

# Investigation of Bond Graphs for Nuclear Reactor Simulations

by

Eugeny Sosnovsky

B.S. Mechanical Engineering and Physics  
Worcester Polytechnic Institute, 2008

Submitted to the Department of Nuclear Science and Engineering  
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Nuclear Science and Engineering

at the

Massachusetts Institute of Technology

May 2010

Copyright © 2010 Massachusetts Institute of Technology  
All rights reserved.

Signature of Author: \_\_\_\_\_

Department of Nuclear Science and Engineering  
May 24, 2010

Certified by: \_\_\_\_\_

Benoit Forget, Ph.D.  
Assistant Professor of Nuclear Science and Engineering  
Thesis Supervisor

\_\_\_\_\_  
Edward Pilat, Ph.D.  
Research Scientist  
Thesis Reader

Accepted by: \_\_\_\_\_

Jacquelyn Yanch, Ph.D.  
Professor of Nuclear Science and Engineering  
Chair, Department Committee on Graduate Students



# **Investigation of Bond Graphs for Nuclear Reactor Simulations**

by

Eugeny Sosnovsky

Submitted to the Department of Nuclear Science and Engineering  
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Nuclear Science and Engineering

May 2010

## **Abstract**

This work proposes a simple and effective approach to modeling multiphysics nuclear reactor problems using bond graphs. The conventional method of modeling the coupled multiphysics transients in nuclear reactors is operator splitting, which treats the single physics individually and exchanges the information at every time step. This approach has limited accuracy, and so there is interest in the development of methods for fully coupled physics simulation.

The bond graph formalism was first introduced to solve the multiphysics problem in electromechanical systems. Over the years, it has been used in many fields including nuclear engineering, but with limited scope due to its perceived impracticality in large systems. In this work, the bond graph formalism is for the first time applied to neutron transport, and coupled to heat transfer in a nuclear reactor. Fully coupled 1D diffusion reaction model is derived using bond graphs, and the transient solution obtained using a proof-of-concept bond graph processing code. The bond graph-based approach to coupled nuclear reactor simulation was shown to be accurate and stable. Suggestions are made for the expansion of the approach to larger problems and higher fidelity simulations.

Thesis Supervisor: Benoit Forget

Title: Assistant Professor of Nuclear Science and Engineering



## **Acknowledgements**

I would first like to acknowledge and deeply thank my advisor, Professor Benoit Forget, for his delicate but immensely helpful support and patience over the course of this project. I do not believe I would be able to have the degree of autonomy and creativity that I enjoyed under any other advisor, and for that I am endlessly grateful.

It was well understood from the start, that this research is in its earliest stages, and was not yet ready to produce a finished product. For this reason, I would like to acknowledge and thank my sponsors over the course of this project. Firstly, the National Academy of Nuclear Training for the NANT fellowship that I was awarded. Secondly, the Department of Energy for the LDRD grant that was given in the early stages of this research. Lastly, I would like to thank the donors that made the Thomson fellowship that I enjoyed possible. I would also like to acknowledge and thank Christopher Newman and Glen Hansen of Idaho National Laboratory, who helped me frame the project, as well as receive the LDRD grant that made the project possible.

Lastly, I would like to deeply thank my parents, Ray and Olga Hayes, without whose support I simply would not have been able to succeed. I must also thank them for taking care of Poirot, my pug, whose nonchalant attitude was more of an inspiration to me throughout this project than he may ever care to know. And finally, thank you Yana, my fiancé, for the love and support you have given me over the years.



# Table of Contents

Abstract.....	3
Acknowledgements.....	5
Table of Contents.....	7
List of Figures.....	9
List of Tables.....	11
Nomenclature.....	13
1. Introduction.....	17
1.1. Background on Coupled Transient Core Modeling.....	17
1.2. Background on Multiphysics Modeling with Bond Graphs.....	17
1.3. Objectives.....	18
2. Background.....	19
2.1. Neutron Transport.....	19
2.2. Heat Transfer.....	22
2.3. Neutron-Thermal Coupling.....	27
2.4. Bond Graph Formalism Theory.....	30
2.5. Bond Graph Formalism Example.....	39
3. Coupled Neutron and Thermal Diffusion via Bond Graphs.....	43
3.1. Thermal Diffusion via Bond Graphs.....	43
3.2. Neutron Diffusion via Bond Graphs.....	51
3.3. Coupled Diffusion via Bond Graphs.....	57
3.4. Multidimensional Multigroup Neutron Diffusion via Bond Graphs.....	63
4. Bond Graph Processing Code Development.....	67
4.1. General Algorithm Description.....	68
4.2. Symbolic and Numeric Expressions Summary.....	72
4.3. Sorting Procedure Description.....	74
4.4. Final Code Description.....	75
4.5. Possible Code Acceleration for Large Problems.....	75
5. Benchmark Problems.....	77
5.1. Method of Manufactured Solutions Theory.....	77
5.2. Benchmark Problem Construction.....	78
5.3. Benchmark Simulation Results.....	80
5.4. Conclusions.....	80
6. Summary and Recommendations for Future Work.....	81
6.1. Summary.....	81
6.2. Recommendations for Future Work.....	81
Appendix A. Bond Graph Processing Code Documentation.....	83
Appendix B. BGSD File Format Documentation.....	89
Appendix C. BGSD File Creator Documentation.....	97
Glossary.....	101
References.....	107





## List of Figures

Figure 1. Uranium-235 Total Microscopic Cross-Section.....	29
Figure 2. Bond Graph Formalism Summary.....	32
Figure 3. Series RLC Circuit Schematic.....	39
Figure 4. Series RLC Circuit Bond Graph Representation.....	40
Figure 5. Series RLC Circuit Augmented Bond Graph Representation .....	40
Figure 6. Discretized 1D Domain .....	44
Figure 7. Heat Diffusion Bond Graph Representation.....	46
Figure 8. Discretized 1D Domain with Flat Thermal Shape Functions.....	47
Figure 9. Discretized 1D Domain with Flat Neutron Shape Functions .....	52
Figure 10. Neutron Diffusion Bond Graph Representation.....	56
Figure 11. Discretized 1D Domain with Flat Coupled Shape Functions.....	57
Figure 12. Coupled Diffusion Bond Graph Representation .....	62
Figure 13. 1D Slice of a Two-Group Neutron Diffusion Bond Graph Representation .....	64
Figure 14. 2D Diffusion Bond Graph Schematic Representation.....	65
Figure 15. Bond Graph Processing Code Summary .....	67
Figure 16. Benchmark Simulation Results .....	80
Figure 17. Bond Directionality Convention for 2-Port Elements.....	95
Figure 18. Annotated Bond Graph System Diagram.....	97
Figure 19. BGSD_Creator Additional Information Specification Screen .....	98
Figure 20. BGSD_Creator Element Type Selection Screen.....	99
Figure 21. BGSD_Creator Expression Entry Screen.....	99
Figure 22. BGSD_Creator Bond Directionality Entry Screen.....	100



## List of Tables

Table 1. Range of Application of Modeling Formalisms .....	31
Table 2. Causality-Direction Configurations .....	32
Table 3. Bond Variables in Various Physical Domains.....	33
Table 4. Basic Elements.....	34
Table 5. Basic Elements in Various Physical Domains .....	39
Table 6. Series RLC Circuit Equations.....	41
Table 7. Time-modulated Source Elements.....	45
Table 8. Heat Diffusion Bond Graph Constituent Expressions with Uniform $k$ .....	46
Table 9. Heat Diffusion Bond Graph Constituent Expressions with Heterogeneous $k$ .....	50
Table 10. Bond Graph Variables for Finite Volume-Discretized Neutron Diffusion.....	55
Table 11. Neutron Diffusion Bond Graph Constituent Expressions.....	56
Table 12. Modulated and 2-Port Resistors.....	60
Table 13. Signal Bonds .....	61
Table 14. Coupled Diffusion Bond Graph Constituent Expressions .....	63
Table 15. BGSolver Element Types .....	69
Table 16. BGSolver Expression Types.....	72
Table 17. Benchmark Problem's Material and Geometric Properties .....	78
Table 18. Source, Storage and Junction Element and Expression Type Compatibility .....	84
Table 19. Resistive Element Expression and Causality Type Compatibility .....	85
Table 20. Modulated Resistive Element Expression and Causality Type Compatibility .....	86



## Nomenclature

In this chapter, all symbols, expressions, notations, abbreviations and acronyms used in this text are defined. For symbols that have more than one meaning, all meanings are listed in a bullet-point list.

### Mathematical Notation

Symbol	Meaning	Expression
$\mathbb{R}^N$	$N$ -dimensional real space	
$a \in S$	$a$ is a member of set $S$	
$a \cong b$	$a$ is approximately equal to $b$	
$a \equiv b$	$a$ is defined as $b$	
$\bar{\mathbf{x}}$	Vector $\bar{\mathbf{x}}$	$\bar{\mathbf{x}} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \in \mathbb{R}^N$ unless declared otherwise
$x_i$	Element $i$ of vector $\bar{\mathbf{x}}$	
$\mathbf{A}$	Matrix $\mathbf{A}$	$\mathbf{A} = \begin{bmatrix} A_{11} & \dots & A_{1N} \\ \vdots & \ddots & \vdots \\ A_{M1} & \dots & A_{MN} \end{bmatrix} \in \mathbb{R}^{M \times N}$ unless declared otherwise
$A_{ij}$	Element $i, j$ of matrix $\mathbf{A}$	
$\mathbf{A}^T$	Transpose of matrix $\mathbf{A}$	$\mathbf{A}^T = \begin{bmatrix} A_{11} & \dots & A_{M1} \\ \vdots & \ddots & \vdots \\ A_{1N} & \dots & A_{MN} \end{bmatrix} \in \mathbb{R}^{N \times M}$ unless declared otherwise
$\dot{\bar{\mathbf{x}}}$	Time derivative of vector $\bar{\mathbf{x}}$	$\dot{\bar{\mathbf{x}}} = \frac{\partial \bar{\mathbf{x}}}{\partial t}$
$\nabla f(\bar{\mathbf{x}})$	Gradient of scalar field $f(\bar{\mathbf{x}})$ ; unless stated otherwise, the gradient is over only the geometric space	$\nabla f(\bar{\mathbf{x}}) = \left[ \frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \dots \quad \frac{\partial f}{\partial x_N} \right]^T$ for a Cartesian $\bar{\mathbf{x}}$
$\nabla_{\bar{\mathbf{x}}} f(\bar{\mathbf{x}}, \bar{\mathbf{y}})$	Gradient of a scalar field $f(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ over only the $\bar{\mathbf{x}}$ -space	$\nabla_{\bar{\mathbf{x}}} f(\bar{\mathbf{x}}, \bar{\mathbf{y}}) = \left[ \frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \dots \quad \frac{\partial f}{\partial x_N} \right]^T$ for a Cartesian $\bar{\mathbf{x}}$
$\nabla \cdot \bar{\mathbf{f}}(\bar{\mathbf{x}})$	Divergence of a vector field $\bar{\mathbf{f}}(\bar{\mathbf{x}})$ ; unless stated otherwise, the divergence is over only the geometric space	$\nabla \cdot \bar{\mathbf{f}}(\bar{\mathbf{x}}) = \frac{\partial f_1}{\partial x_1} + \frac{\partial f_2}{\partial x_2} + \dots + \frac{\partial f_N}{\partial x_N}$ for a Cartesian $\bar{\mathbf{x}}$
$\nabla_{\bar{\mathbf{x}}} \cdot \bar{\mathbf{f}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$	Divergence of a vector field $\bar{\mathbf{f}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ over only the $\bar{\mathbf{x}}$ -space	$\nabla_{\bar{\mathbf{x}}} \cdot \bar{\mathbf{f}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) = \frac{\partial f_1}{\partial x_1} + \frac{\partial f_2}{\partial x_2} + \dots + \frac{\partial f_N}{\partial x_N}$ for a Cartesian $\bar{\mathbf{x}}$

$\iint_{4\pi} d\hat{\Omega} f(\hat{\Omega})$	Integral of scalar field $f(\hat{\Omega})$ over all directions $\hat{\Omega}$	$\iint_{4\pi} d\hat{\Omega} f(\hat{\Omega}) = \int_0^{2\pi} d\varphi \int_0^\pi d\theta \sin(\theta) f(\theta, \varphi)$ for $\hat{\Omega}$ in spherical coordinates
$\iiint_{\forall} d\forall f(\vec{x})$	Integral of scalar field $f(\vec{x})$ over volume $\forall$	$\iiint_{\forall} d\forall f(\vec{x}) = \int_{x_0}^{x_1} dx \int_{y_0}^{y_1} dy \int_{z_0}^{z_1} dz f(x, y, z)$ for a Cartesian $\vec{x}$
$\oiint_S dS f$	Surface integral of quantity $f$ over a surface $S$	$\oiint_S dS f = \iint_S \vec{dS} \cdot \vec{f}$
$(f)_a$	Expression $f$ evaluated while holding $a$ constant	
$f^0$	Initial value of quantity $f$	$f^0 = f(t=0)$
$\forall^1$	For all	$\forall i \in 1 \dots 6$ means “for all $i$ from 1 to 6”
$\vec{x} \cdot \vec{y}$	Scalar/dot/inner product of vectors $\vec{x}$ and $\vec{y}$	$\vec{x} \cdot \vec{y} = \sum_{i=1}^N x_i y_i$ for Cartesian $\vec{x}$ and $\vec{y}$

## Latin Symbols

Symbol	Meaning
$A$	Cross-sectional area
$A_j$	Mass number of isotope $j$
$\vec{b}$	Vector of bond variables
$c$	Precursor concentration
$c_p$	Specific heat at constant pressure
$c_v$	Specific heat at constant volume
$Co$	Neutron confusion coefficient
$D$	Neutron diffusion coefficient
$E$	Neutron energy
$e$	Effort
$f$	Flow
$h$	Specific enthalpy
$J$	Current density
$k$	Thermal conductivity
$\vec{m}$	Vector of modulating variables
$N$	Number of neutrons
$N_b$	Number of bonds in the system
$N_e$	Number of bond graph elements in the system
$N_j$	Number density of isotope $j$
$n$	Neutron density
$\vec{n}$	Vector of numeric variables
$o$	Thermal resistivity

<sup>1</sup> Different from italicized  $\forall$ , which means “volume.”

$P$	· Power · Pressure
$p$	Generalized momentum
$s_{ex}$	External neutron source
$T$	Temperature
$t$	Time
$U$	Thermal energy
$U_{int}$	Total internal energy of the system
$U_{sens}$	Sensible energy of the system
$U_{lat}$	Latent energy of the system
$U_{chem}$	Chemical energy of the system
$U_{nucl}$	Nuclear energy of the system
$u$	Thermal energy density
$u_m$	Specific thermal energy
$u_v$	Thermal energy density in volume
$\bar{\mathbf{u}}_v''$	Heat flux vector
$V$	Velocity
$V_n$	Neutron velocity
$\forall$	Volume
$v$	Specific volume
$w$	Thermal energy generated per fission
$\bar{\mathbf{x}}$	· Position in geometric space · State vector at time $t$

## Greek Symbols

Symbol	Meaning
$\beta$	Total delayed neutron fraction
$\beta_i$	Delayed group $i$ delayed neutron fraction
$\theta$	Inclination angle from the $z$ -axis
$\theta_d$	Inclination angle from the $z$ -axis of the direction vector
$\theta_p$	Inclination angle from the $z$ -axis of the position vector
$\lambda$	· Decay constant · Eigenvalue
$\lambda_i$	Delayed group $i$ decay constant
$\mu$	Angle cosine
$\mu_0$	Scattering angle cosine
$\bar{\mu}_0$	Mean scattering angle cosine
$\nu$	Average number of neutrons born per fission
$\rho$	Mass density in volume
$\sigma_j$	Microscopic cross-section of type $j$
$\Sigma_j$	Macroscopic cross-section of type $j$

$\varphi$	· Angular flux · Azimuthal angle
$\varphi_d$	Azimuthal angle of the direction vector
$\varphi_p$	Azimuthal angle of the position vector
$\phi$	Scalar flux
$\chi(E)$	Spectrum function
$\chi_f(E)$	Prompt fission neutron spectrum function
$\chi_{di}(E)$	Delayed group $i$ neutron spectrum function
$\hat{\Omega}$	Direction unit vector
$d\hat{\Omega}$	Differential element of the direction space $d\hat{\Omega} = \sin(\theta_d) d\theta_d d\varphi_d$

## Abbreviations

Abbreviation	Meaning
AE	Algebraic Equation
BGS	Bond Graph System
BGSD	Bond Graph System Descriptor
CAS	Computer Algebra System
DAE	Differential-Algebraic Equation
FEA	Finite Element Analysis
JFNK	Jacobian-Free Newton-Krylov
MMS	Method of Manufactured Solutions
MOL	Method Of Lines
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
PIDE	Partial Integro-Differential Equation
SCAP	Sequential Causality Assignment Procedure
SME	Symbolic Math Engine
SMT	Symbolic Math Toolbox



# 1. Introduction

The important physics for reactor simulation include neutron transport, thermal hydraulics and mechanical response, which are all inherently strongly coupled. Most current efforts typically only model a single physics where the coupled data is determined externally. This is an imperfect approach, particularly for transient analysis, because the strong interdependence of the physics involved creates stability and accuracy issues. (Ref. [1]). The inability to accurately predict transients under accident scenarios is the primary reason for highly conservative safety limits imposed on reactors. Due to computational limitations, this approach was employed for many years. Therefore, there is a need for faster, more accurate fully coupled codes capable of handling coupled transient systems. The introduction of such codes would allow reducing the safety margins on reactor operation and manufacturing, and increase predictive capabilities for reactor simulations.

A similar single physics approach was used in electromechanical systems until a new formalism was proposed and developed by Henry Paynter of MIT in the 1960s (Ref. [2]). This formalism is known as Bond Graphs, and it is a way of representing engineering systems as combinations of storage elements (capacitive and inertial), resistive elements, and junction elements, connected by bonds which transfer energy (or a similar conserved quantity) between the elements.

Bond graph formalism has never been applied to the simulation of coupled multiphysics in nuclear reactors. In this work, this approach is developed, and the feasibility of such approach is investigated.

## 1.1. Background on Coupled Transient Core Modeling

Neutron-induced fission in nuclear reactors is the main source of heat in the system. Heat travels through the system, and in time, adjusts the temperature field in the reactor. Macroscopic neutron cross-sections are affected by temperature; the cross-sections themselves determine the scalar neutron flux field in the nuclear reactor. This field, in turn, determines the fission reaction rate, which determines the heat generation.

The physical system is clearly tightly coupled, however, each individual type of physics is complicated enough to only be analyzed individually. This approach is known as operator splitting, and is the basis of most coupled nuclear reactor transient analysis. However, operator splitting has strong associated computational limitations, and so there is significant interest in the field to move away from operator splitting and towards fully coupled simulation.

The approach proposed in this work for doing so is summarized in the next section. A more detailed discussion of the physical nature of the coupling in nuclear reactor simulation is given in section 2.3 (p. 27).

## 1.2. Background on Multiphysics Modeling with Bond Graphs

As mentioned above, bond graph formalism is a technique for modeling engineering systems as combinations of connected elements. Bond graphs were originally introduced for mechatronics, but over time grew from a comprehensive methodology to model mechatronic systems into a complete research field, concerned with modeling mechanical, electrical, magnetic, hydraulic, thermal, and even optical and financial systems. Bond graphs have been used for modeling various field problems, such as thermal diffusion, but have never been applied to neutron transport.

The basic idea of modeling a system with bond graphs is to represent it using bond graphs, and then to apply a bond graph processing algorithm to the resulting bond graph system. This algorithm results in the formulation of the state derivative vector, which can then be integrated to obtain full information about the system's dynamics.

The bond graph processing algorithm is rigorous enough to be automated; however, there exists no available code capable of processing large, nonlinear and automatically generated bond graph systems. Such a code would need to be developed to model nuclear reactors using bond graphs. A detailed description of the bond graph formalism, the bond graph processing algorithm and other related material is provided in sections 2.4 (p. 30) and 2.5 (p. 39).

### **1.3.Objectives**

As stated above, bond graph formalism has never been applied to neutron transport, and therefore to neutron transport and thermal hydraulic coupling. Furthermore, there currently exists no available bond graph processing code powerful enough to process the bond graphs that may arise from modeling the nuclear reactor multiphysics. For these reasons, to explore the feasibility of using bond graphs to model nuclear reactors, the following objectives need to be accomplished:

1. To identify a method to represent neutron diffusion via bond graph formalism.
2. To identify a method to couple neutron diffusion and thermal diffusion via bond graph formalism.
3. To develop a code fit for automatic processing of systems modeled via bond graph formalism.
4. To construct and run a benchmark and draw conclusions about the method's feasibility.

## 2. Background

Several different types of physical effects take place in nuclear reactors. They include neutron transport, fuel depletion, (Refs. [3-5]), one and two-phase fluid dynamics, (Refs. [6,7]), heat transfer in fluids and heat diffusion in solids (Refs. [6,8]), materials stress mechanics (Ref. [9]), materials degradation under irradiation and materials chemistry (Ref. [10]).

Fundamentally, all these phenomena interact with each other, and therefore can be considered *coupled*.<sup>2</sup> This essentially means that changing the properties or the dynamics of one or more of the phenomena directly or indirectly affects all others. However, this coupling can be very weak; for example, the dynamics of the materials degradation under irradiation very weakly affect the thermal conductivity of the fuel material. For this reason, by inspection of the references above, these physical effects are generally all treated separately.

The important physics in transient safety analysis of nuclear reactors are the ones that occur over time scales from milliseconds to several hours, but not longer. Of the list above, these include neutron transport, fluid dynamics and related heat transfer phenomena, and, if accounting for materials failure, the materials stress mechanics. In this work, neutron transport and heat transfer are considered; this occurs under the implicit assumption that the materials do not fail and therefore the reactor geometry does not change, except possibly under thermal expansion. This is a typical approach in modern reactor analysis. (Ref. [11]).

In this chapter, the physics and common numerical approaches to these phenomena are first discussed individually in sections 2.1 and 2.2, followed by the discussion of the nature and mathematics of their coupling in section 2.3. The purpose of this work is to apply the bond graph formalism to modeling these phenomena, so a detailed introduction to bond graph formalism then follows in section 2.4. An example of application of bond graph formalism is then provided in section 2.5.

### 2.1. Neutron Transport

Typical assumptions in neutron transport analysis are (Ref. [3]):

- Neutrons can be treated as point particles with no wave-like quantum mechanical effects, which is valid on the macroscopic scale considered.
- High enough neutron density for the deterministic approach to be valid, which holds for at-power nuclear reactor, with the neutron density on the order of  $10^{14} n/cm^3$ .
- No neutron-to-neutron interactions. Valid because atom density is on the order of  $10^{23} a/cm^3$ , which is much greater than the neutron density on the order of  $10^{14} n/cm^3$ .
- Collisions are well-defined 2-body events which occur instantaneously. This is an experimentally validated fact for  $(n, a)$  collisions for any nuclide  $a$ .
- Between collisions, neutrons stream with constant velocity. Valid because neutrons are neutral elementary particles, which only undergo weak nuclear, strong nuclear and gravitational interactions.
- Material properties are unaffected by neutron interactions with the nuclides. Valid because of time scales considered; if depletion time scales were considered, macroscopic cross-sections would vary.

Additionally, as discussed above, in this work completely stationary geometry and constant material composition in solids is assumed. Material composition in fluids can still vary,

---

<sup>2</sup> *Italicized* terms are defined in the Glossary chapter, p. 101.

due to the motion of the fluid. In this section, the temperature dependence of the macroscopic cross-sections is neglected; thermal feedback is discussed in section 2.3. Additionally, both prompt and delayed neutron productions from all precursor groups are assumed to be isotropic.

Under all of the above assumptions, the neutron transport partial integro-differential equation (PIDE) becomes Eq. (2.1). The 6-precursor group delayed neutron precursor equations become Eqs. (2.2) (Ref. [5]):

$$\begin{aligned} \frac{1}{V_n(E)} \frac{\partial}{\partial t} \varphi(t, \bar{\mathbf{x}}, E, \hat{\Omega}) = & -\hat{\Omega} \cdot \nabla \varphi(t, \bar{\mathbf{x}}, E, \hat{\Omega}) - \Sigma_t(\bar{\mathbf{x}}, E) \varphi(t, \bar{\mathbf{x}}, E, \hat{\Omega}) + \\ & + \int_0^\infty dE' \iint_{4\pi} d\hat{\Omega}' \Sigma_s(\bar{\mathbf{x}}, E' \rightarrow E, \hat{\Omega}' \rightarrow \hat{\Omega}) \varphi(t, \bar{\mathbf{x}}, E', \hat{\Omega}') + \\ & + (1-\beta) \frac{\chi_f(E)}{4\pi} \int_0^\infty dE' \iint_{4\pi} d\hat{\Omega}' \nu(E') \Sigma_f(\bar{\mathbf{x}}, E') \varphi(t, \bar{\mathbf{x}}, E', \hat{\Omega}') + \\ & + \frac{1}{4\pi} \sum_{i=1}^6 \chi_{di}(E) \lambda_i c_i(t, \bar{\mathbf{x}}) + s_{ex}(t, \bar{\mathbf{x}}, E, \hat{\Omega}) \end{aligned} \quad (2.1)$$

$$\frac{\partial}{\partial t} c_i(t, \bar{\mathbf{x}}) = \beta_i \int_0^\infty dE' \iint_{4\pi} d\hat{\Omega}' \nu(E') \Sigma_f(\bar{\mathbf{x}}, E') \varphi(t, \bar{\mathbf{x}}, E', \hat{\Omega}') - \lambda_i c_i(t, \bar{\mathbf{x}}) + \dot{c}_{ex,i}(t, \bar{\mathbf{x}}) \quad \forall i=1\dots 6 \quad (2.2)$$

using the following notation<sup>3</sup>:

$t$	Time
$\bar{\mathbf{x}}$	Position in geometric space
$E$	Neutron energy
$\hat{\Omega}$	Direction unit vector
$\varphi(t, \bar{\mathbf{x}}, E, \hat{\Omega})$	Angular flux density in energy <sup>4</sup>
$dE$	Energy differential
$d\hat{\Omega}$	Differential element of the direction space
$V_n(E)$	Neutron velocity of a neutron with energy $E$
$\Sigma_t(\bar{\mathbf{x}}, E)$	Macroscopic total cross-section
$\Sigma_f(\bar{\mathbf{x}}, E)$	Macroscopic fission cross-section
$\Sigma_s(\bar{\mathbf{x}}, E' \rightarrow E, \hat{\Omega}' \rightarrow \hat{\Omega})$	Double differential macroscopic scattering cross-section
$\iint_{4\pi} d\hat{\Omega} f(\hat{\Omega})$	Integral of scalar field $f(\hat{\Omega})$ over all directions $\hat{\Omega}$
$\nu(E)$	Average number of neutrons born per fission caused by neutron with energy $E$
$\chi_f(E)$	Prompt fission neutron spectrum function
$\chi_{di}(E)$	Delayed group $i$ neutron spectrum function
$\beta$	Total delayed neutron fraction
$\beta_i$	Delayed group $i$ delayed neutron fraction

<sup>3</sup> All symbols, mathematical expressions, acronyms and abbreviations used in this work are described in the Nomenclature chapter, p. 13.

<sup>4</sup> Note: angular neutron flux density in energy and external angular neutron source density in energy are both densities in the entire 7-phase space: they are densities in angle, energy, geometric space and rates in time. However, the terminology used here is the standard terminology and notation used to denote these quantities.

$\lambda_i$	Delayed group $i$ decay constant
$s_{ex}(t, \bar{x}, E, \hat{\Omega})$	External angular neutron source density in energy
$c_i(t, \bar{x})$	Delayed group $i$ precursor concentration
$\dot{c}_{ex,i}(t, \bar{x})$	Delayed group $i$ external precursor concentration source

The external neutron source  $s_{ex}$  is a partially physical quantity – in certain reactors, external sources are used for start-up and other purposes. The external precursor concentration source  $\dot{c}_{ex,i}$ , however, is a nonphysical quantity, introduced in the equation only to be used as a corrective source if the *method of manufactured solutions* (MMS) is used.

In this text, a significantly simplified neutron transport model is considered. A model this simple is inapplicable except for a preliminary analysis of real nuclear reactors. However, it can be used to test the potential of using the new approach explored in this work, thus making the simplified model ideal for a proof-of-concept study. The simplified model used in this text is a 1D one-group neutron diffusion model with no delayed neutrons. The underlying additional assumptions for this simplified model are:

- Neutron scattering is linearly anisotropic or fully isotropic.
- All fission-born neutrons are prompt.
- All macroscopic cross-sections in the model are one-group, constructed using an appropriate energy condensation technique (Ref. [4]).
- The reactor geometry is that of an infinite slab reactor, thus reducing the problem to one-dimensional.

The resulting 1D one-group neutron diffusion equation is:

$$\frac{\partial}{\partial t} n(t, x) = -\frac{\partial}{\partial x} J(t, x) + \nu \Sigma_f(x) \phi(t, x) - \Sigma_a(x) \phi(t, x) + s_{ex}(t, x) \quad (2.3)$$

using the following notation<sup>5</sup>:

$n(t, x)$	Neutron density in 3D geometric space
$\phi(t, x)$	One-group scalar flux
$J(t, x)$	Scalar net current density in the $+x$ direction
$\Sigma_f(x)$	One-group macroscopic fission cross-section
$\Sigma_a(x)$	One-group macroscopic absorption cross-section
$s_{ex}(t, x)$	Neutron source density in 3D geometric space

Here all relevant quantities have been integrated in energy and angle.  $\phi(t, x)$  is given by the definition of the one-group scalar flux:

$$\phi(t, x) = V_n n(t, x) \quad (2.4)$$

in which:

$V_n$	One-group neutron velocity
-------	----------------------------

$V_n$  is the flux-weighted average neutron velocity, which will be assumed given like the one-group macroscopic cross-sections.

Neutron current density  $J(t, x)$  is given by Fick's law (Ref. [3]):

$$J(t, x) = -D(x) \frac{\partial}{\partial x} \phi(t, x) \quad (2.5)$$

<sup>5</sup> From here onward: only quantities previously unused will be defined.

in which:

$D(x)$	One-group neutron diffusion coefficient
--------	---

$D(x)$  is also a material property, which can be obtained according to:

$$D(x) = \frac{1}{3(\Sigma_t(x) - \bar{\mu}_0(x)\Sigma_s(x))} \quad (2.6)$$

in which:

$\Sigma_t(x)$	One-group macroscopic total cross-section
$\Sigma_s(x)$	One-group macroscopic scattering cross-section
$\bar{\mu}_0(x)$	Mean scattering angle cosine

$\Sigma_t(x)$  and  $\Sigma_s(x)$  are both data which can be assumed to be given.  $\Sigma_t(x)$  can also be related to other macroscopic cross-sections:

$$\Sigma_t(x) = \Sigma_s(x) + \Sigma_a(x) = \Sigma_s(x) + \Sigma_c(x) + \Sigma_f(x) \quad (2.7)$$

in which:

$\Sigma_c(x)$	One-group macroscopic capture cross-section
---------------	---

Equations (2.6) and (2.7) will generally hold for group-specific quantities as well. The mean scattering angle cosine  $\bar{\mu}_0(x)$  is, in general, a material property and will be assumed to be given. For elastic scattering, mean scattering angle cosine from isotope  $j$  is approximately given by (from Ref. [5]):

$$\bar{\mu}_{0j} \cong \frac{2}{3A_j} \quad (2.8)$$

in which:

$\bar{\mu}_{0j}$	Mean scattering angle cosine from isotope $j$
$A_j$	Mass number of isotope $j$

Using Eq. (2.8), the mean scattering angle cosine  $\bar{\mu}_0(x)$  can be estimated:

$$\bar{\mu}_0(x) \cong \sum_{\text{all } j} \left( \frac{\frac{2}{3A_j} \Sigma_{sj}(x)}{\Sigma_s(x)} \right) \quad (2.9)$$

in which:

$\Sigma_{sj}(x)$	One-group macroscopic scattering cross-section of isotope $j$
------------------	---

Equation (2.9) is essentially a weighted average of the macroscopic scattering angle cosines from all present isotopes. From the inspection of Eq. (2.8), it is easy to see that  $\bar{\mu}_0(x)$  is dominated by light nuclei, the most important one being Hydrogen-1 with  $\bar{\mu}_0 = 2/3$ .

Equations (2.3)-(2.5) summarize the most basic neutron transport model analyzed in this text. More complicated models will be discussed, but for proof-of-concept analysis and basic coupling, this primitive model will suffice.

## 2.2.Heat Transfer

Three types of heat transfer occur in typical nuclear reactors. The first type of heat transfer is thermal conduction, which occurs due to the diffusion of thermal energy in solids. Thermal conduction is the primary mechanism responsible for heat transfer through the fuel and the cladding. The second type of heat transfer is convection, which occurs due to the mechanical motion and possibly boiling of the fluid in contact with the cladding. Convection is the primary

mode of heat removal from the fuel elements. Closely related to convection is advection, which refers to the transport of thermal energy by the fluid due to the bulk motion of the fluid, once the thermal energy has been collected by convection. The third type of heat transfer is thermal radiation, which occurs as infrared radiation by the heated cladding surfaces. Thermal radiation is almost always neglected in reactor analysis, since the materials are primarily opaque, and diffusion followed by convection dominate the heat removal from the fuel. (Ref. [8]).

The simplest mode of heat transfer is pure thermal diffusion. In this text, it will be the primary heat transfer mode of interest, for several reasons. Firstly, it occurs in fuel, and is therefore coupled most directly to neutron transport in the fuel. Secondly, it is the simplest type of heat transfer to model, and is therefore a good starting point.

Thermal diffusion without advection occurs only in solids, such as fuel pins. Fundamentally, thermal diffusion is an expression of thermal energy balance equation with a possible external source and a diffusive operator.

In general the *internal energy* of a system consists of 4 components:

- Sensible energy: energy associated with the particles’ kinetic energies.
- Latent energy: energy associated with the systems’ materials’ phases.
- Chemical energy: energy associated with the chemical bonds of the particles in the system.
- Nuclear energy: energy associated with the nuclear bonds in the system’s nuclei.

Most thermodynamic texts utilize the internal energy in some form (Ref. [12]), but an explicit discussion of its components is less common. Reference [13] contains such discussion, which is summarized below.

The relation between the 4 components of the internal energy can be summarized in the following expression:

$$U_{int} = U_{sens} + U_{lat} + U_{chem} + U_{nucl} \quad (2.10)$$

in which:

$U_{int}$	Total internal energy of the system
$U_{sens}$	Sensible energy of the system
$U_{lat}$	Latent energy of the system
$U_{chem}$	Chemical energy of the system
$U_{nucl}$	Nuclear energy of the system

In vast majority of thermodynamic analysis, nuclear energy of the system does not vary. Even in nuclear reactors, while nuclei fission and large amount of energy is released, the total binding energy in the remaining nuclei is still many orders of magnitude higher than the sensible, latent and chemical energy of the system. This is the case because the strong nuclear interaction forces are generally much stronger than the electromagnetic interaction forces which primarily account for the sensible, latent and chemical energies. (Ref. [14]). Furthermore, while sensible, latent and chemical energies can be measured relatively easily, nuclear energy is very difficult to measure since it requires splitting atoms. For these reasons, the total internal energy of the system is very rarely treated directly. Unless chemical reactions occur in a system, chemical energy also does not change. However, sensible energy changes any time the system temperature changes, and latent energy changes any time a phase change occurs. Both temperature and phase changes occur all the time in a nuclear reactor and other heat exchanging systems. Therefore, when analyzing the transient thermodynamics of the system, it is best to concentrate on the sensible and latent energies of the system. This combination of sensible and latent energies of the system is called the *thermal energy* of the system:

$$U = U_{sens} + U_{lat} \quad (2.11)$$

in which:

$U$	Thermal energy of the system
-----	------------------------------

In practice, and in this text, the quantity referred to as the “internal energy” of the system is usually the thermal energy of the system. For that reason, the terms “internal energy” and “thermal energy” will be used interchangeably.

All of the above quantities are extensive system properties, which means that they characterize the system as a whole, and not the internal distribution of each property within the system. The intensive property corresponding to the thermal energy  $U$  is the specific thermal energy, which is given by (assuming isotropic homogeneous system in thermodynamic equilibrium):

$$U = u_m m \quad (2.12)$$

in which:

$u_m$	Specific internal energy of the system’s material
$m$	Mass of the material in the system

Typically, in literature, the symbol  $u$  is used to mean specific internal energy. Specific internal energies of engineering materials and working fluids are tabulated, and are generally functions of the material’s thermodynamic state, that is:

$$u_m = u_m(P, T) \quad (2.13)$$

in which:

$P$	Pressure
$T$	Temperature

Equation (2.13) is also known as the caloric equation of state (Ref. [7]). The thermodynamic state of any substance is fixed (determined) by a pair of any two values of its properties. Therefore, instead of defining  $u_m$  as a function of pressure and temperature, it can be defined as a function of temperature and specific volume. However, thermodynamic tables for most substances normally accept pressure and temperature as the inputs, so it is best to leave Eq. (2.13) in its present form.

Specific internal energy can be related to the volumetric thermal energy density by:

$$u_v = \rho u_m = u_m / v \quad (2.14)$$

in which:

$u_v$	Volumetric thermal energy density
$\rho$	Material density
$v$	Fluid specific volume

The specific thermal energy, pressure and specific volume of a fluid are often treated using a combined thermodynamic property, called enthalpy:

$$h = u_m + Pv \quad (2.15)$$

in which:

$h$	Fluid specific enthalpy
-----	-------------------------

Equation (2.15) can, in principle, be used for a solid, but solids do not do significant work by expansion, therefore most energy added to or removed from a solid is by heat transfer.

It should be noted, that while a change in thermal energy is a measurable quantity, thermal energy itself is not. (Ref. [12]). However, in a heat transfer analysis of a nuclear reactor, and of vast majority of physical systems in general, interest is limited to the temperature field in



the system as a function of space and time, and possibly in the energy production and transfer rates. For this reason, as long as the rate of change of thermal energy density in volume can be related to the temperature field in the reactor, the absolute value of the thermal energy density is unimportant.

Equations (2.13) and (2.14) relate the temperature and the volumetric thermal energy density of the substance, but are normally more convenient for use with fluids, since fluid specific thermal (internal) energies are tabulated in this form. For solids, as well as for incompressible liquids and ideal gases, instead of working with specific thermal energies, specific heat capacities are used instead. Specific heat capacity is a measure of the amount of thermal energy required to change the material's temperature. Two types of specific heat capacity are most commonly found for working fluids and engineering materials: specific heat at constant pressure, and specific heat at constant volume. Their formal definitions are given below (from Ref. [7]):

$$c_p \equiv \left( \frac{\partial h}{\partial T} \right)_P \quad (2.16)$$

$$c_v \equiv \left( \frac{\partial u_m}{\partial T} \right)_v \quad (2.17)$$

in which:

$c_p$	Specific heat at constant pressure
$c_v$	Specific heat at constant volume
$(f)_P$	Expression $f$ evaluated while holding $P$ constant

Both specific heats are intensive thermodynamic properties fixed by the state, similarly to  $u_m$  and  $h$ .

For most liquids and solids, it is nearly impossible to hold the substance at a constant volume while heating it, so primarily, specific heat at constant pressure  $c_p$  is used. For most engineering materials, if  $P$  is held constant, enthalpy gain due to thermal expansion is generally small compared to the thermal energy gain from heating. The enthalpy loss due to thermal compression is also small if the material is being cooled.  $c_p$  does not vary considerably with temperature, so in most heat transfer analyses, it can be held constant for a solid, and used to relate temperature and thermal energy density by:

$$u_v \cong \rho(T) c_p T \quad (2.18)$$

An implicit assumption in Eq. (2.18) is that  $c_p$  is constant for all temperatures  $T$ . In reality, while  $c_p$  may be nearly constant around temperature  $T$ , it is likely very different for temperatures significantly different from  $T$ . For this reason, the thermal energy density of a solid material in Eq. (2.18) may actually be significantly different from the true thermal energy density.

Fortunately,  $u_v$  itself is not an important quantity for us since the goal is to compute the temperature field in the system as a function of time, and to compute the various heat transfer and generation rates in the system as functions of time. Temperature field can be related to the heat transfer and generation rates by appropriate partial differential equations (PDEs), depending on the phase of the substance being modeled. These PDEs are generally all forms of thermal energy balance laws, but the ways in which thermal energy is transferred varies depending on the substance phase.

As stated above, in this text, primary interest lies in thermal conduction which occurs in solids, such as the fuel pins. In a solid, the temperature field can be related to the thermal energy density in volume by the thermal energy diffusion equation (Ref. [13]):

$$\frac{\partial}{\partial t} u_v(t, \bar{\mathbf{x}}) = -\nabla \cdot \bar{\mathbf{u}}_v''(t, \bar{\mathbf{x}}) + \dot{u}_{v,ex}(t, \bar{\mathbf{x}}) \quad (2.19)$$

in which:

$\bar{\mathbf{u}}_v''(t, \bar{\mathbf{x}})$	Heat flux vector
$\dot{u}_{v,ex}(t, \bar{\mathbf{x}})$	External heat source density

The heat flux vector depends on the temperature gradient. In most engineering materials, the linear thermal conductivity model is completely sufficient, which leads to Fourier's law (Ref. [13]):

$$\bar{\mathbf{u}}_v''(t, \bar{\mathbf{x}}) = -k(\bar{\mathbf{x}}, T) \nabla T(t, \bar{\mathbf{x}}) \quad (2.20)$$

in which:

$k(\bar{\mathbf{x}}, T)$	Material thermal conductivity
--------------------------	-------------------------------

Equation (2.19) does not contain a  $u_v$  term, only a rate of change of  $u_v$  and temperature terms are present. For this reason, as long as a base temperature is selected such that Eq. (2.18) holds well about that temperature, the above-mentioned discrepancy between the  $u_v$  being treated in Eq. (2.19) and the true  $u_v$  does not make a difference. All properties are functions of temperature, not thermal energy density. Inverting Eq. (2.18) yields temperature as a function of thermal energy density, assuming  $\rho(u_v)$  can be derived from the material properties table:

$$T(u_v) \cong \frac{1}{\rho(u_v) c_p} u_v \quad (2.21)$$

For reasons discussed above, equations (2.19)-(2.21) together provide an accurate model for pure thermal diffusion in a solid, as long as the temperature does not depart too far from the temperature at which  $c_p$  was evaluated. For most engineering materials, as long as the base temperature is well chosen, this is a good assumption. Furthermore, while thermal expansion can occur in a solid, the rate of change of  $\rho(u_v)$  is generally much slower in a fractional sense than the rate of change of  $u_v$  itself. Therefore, in a solid, a very good simplification, which neglects thermal expansion, is given by:

$$T(u_v) \cong \frac{1}{\rho c_p} u_v \quad (2.22)$$

Since 1D geometry is being treated for reasons discussed in section 2.1, Eqs. (2.19) and (2.20) reduce to:

$$\frac{\partial}{\partial t} u_v(t, x) = -\frac{\partial}{\partial x} u_v''(t, x) + \dot{u}_{v,ex}(t, x) \quad (2.23)$$

$$u_v''(t, x) = -k(x, T) \frac{\partial}{\partial x} T(t, x) \quad (2.24)$$

Equations (2.22)-(2.24) form the basic thermal diffusion model used in this text.

Besides the heat transfer in a solid, which is primarily analyzed in this text, heat transfer in a nuclear reactor also occurs in the working fluid. Fluid motion accounts for the convection of thermal energy from the cladding to the working fluid, and the advection of the thermal energy out of the nuclear reactor by the moving fluid. These processes are generally modeled by coupled Navier-Stokes equations, mass conservation equation and the equations of state (Ref. [7]). Heat

transfer by fluids is not modeled in this text, but it will have to be considered when moving beyond the proof-of-concept stage.

### 2.3. Neutron-Thermal Coupling

The two types of physics analyzed in this text are neutron transport and heat transfer. Uncoupled neutron transport was discussed in section 2.1, and uncoupled heat transfer was discussed in section 2.2. The primary models of interest in this text are the 1D neutron and thermal diffusion equations, so coupling for these equations will be treated in this section.

The purpose of sustained fission in a power reactor is to create a heat generation rate, called “fission heat.” Therefore, part of the external source term in the heat balance equation (Eq. (2.23)) is the fission heat source, which models the heat generation due to fission reactions. Other types of neutron-nucleus reactions, such as radiative capture, also generate heat, but fission reactions are the dominant source of heat, so other absorptions are negligible for the purposes of heat generation. (Ref. [4]). The heat generation rate density due to fission can therefore be written as (temperature dependence of the cross-section is discussed below):

$$\dot{u}_{v,f}(t, x) = w \Sigma_f(x, T) \phi(t, x) \quad (2.25)$$

in which:

$\dot{u}_{v,f}(t, x)$	Heat generation rate density in volume due to fission
$w$	Thermal energy released per fission event

Typically, in thermal hydraulic analysis of nuclear reactors,  $\dot{u}_{v,f}$  is assumed given as an external source, independent of the temperature field in the reactor. From Eq. (2.25), it is clear that this approximation is of limited validity, particularly in transient analysis, in which both flux and cross-sections are guaranteed to change.

The more complicated type of coupling present in the coupled nuclear reactor is the temperature dependence of the macroscopic cross-sections.

For the following treatment, only one isotope is assumed; it is easily extendable to multiple isotopes by combining their individual contributions to the macroscopic cross-sections. Total cross-sections are treated below; however, exactly the same analysis is applicable to any type of cross-section.

A one-group total macroscopic cross-section is given by (from Ref. [4]):

$$\Sigma_t(\bar{\mathbf{x}}, T) = \frac{\sum_{\text{all } j} \left[ \int_0^\infty dE N^j(\bar{\mathbf{x}}, T) \sigma_t^j(T, E) \phi(\bar{\mathbf{x}}, E) \right]}{\int_0^\infty dE \phi(\bar{\mathbf{x}}, E)} \quad (2.26)$$

in which:

$N^j(\bar{\mathbf{x}}, T)$	Atom (number) density of isotope $j$ at temperature $T$
$\sigma_t^j(T, E)$	Total microscopic cross-section of isotope $j$ at temperature $T$ and energy $E$
$\phi(\bar{\mathbf{x}}, E)$	Scalar flux density in energy at energy $E$

The atom densities and microscopic cross-sections depend on temperature, which results in the temperature dependence of the one-group macroscopic cross-section. These dependencies are for two separate reasons. Consider the following expression for the atom density of isotope  $j$ :

$$N^j(\bar{\mathbf{x}}, T) = NA^j x^k N_A \frac{\rho^{comp}(\bar{\mathbf{x}}, T)}{M_m^{comp}} \quad (2.27)$$

in which:

$NA^j$	Isotope abundance of isotope $j$
$x^k$	Molar fraction of element $k$ of which $j$ is an isotope
$N_A$	Avogadro's number
$\rho^{comp}(T)$	Compound mass density at temperature $T$
$M_m^{comp}$	Molar mass of the compound

Equation (2.27) assumes that isotope abundances and molar fractions of the elements in the compound are spatially homogeneous. As long as  $\bar{x}$  is within the compound, this is a good assumption.

By inspection of Eq. (2.27), it is clear that the atom density depends on temperature due to the temperature dependence of the compound's mass density. For isotropic materials, like most solids in the nuclear reactor, the dependence of density on temperature can be expressed using the coefficient of thermal expansion:

$$\frac{\partial}{\partial T} \rho(T) = -\alpha_v(T) \rho(T) \quad (2.28)$$

in which:

$\alpha_v(T)$	Material's volumetric thermal expansion coefficient at temperature $T$
---------------	--

Neglecting thermal variation of  $\alpha_v(T)$ , similarly to how thermal variation of  $c_p$  was neglected in section 2.2, Eq. (2.28) can be integrated to:

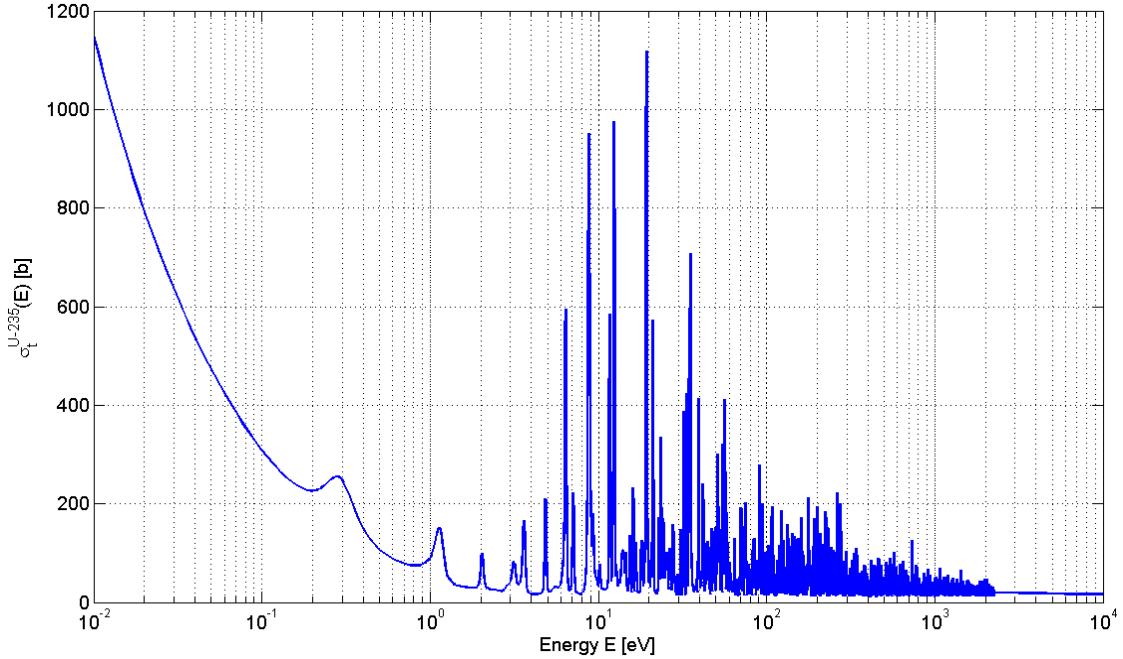
$$\rho(T) = \rho_0 e^{\alpha_v(T_0 - T)} \quad (2.29)$$

in which:

$\rho_0$	Material's mass density at some reference temperature $T_0$
$T_0$	Reference temperature about which the equation holds accurately

For most engineering materials, Eq. (2.29) holds well for a wide temperature range, as long as  $\alpha_v$  is chosen at an appropriate average temperature. However, at higher temperatures, oxide materials can restructure, which can produce a jump in density.  $UO_2$  is an example of such material; up to 3 zones of different density can exist in a hot fuel pellet. (Ref. [8]). For these reasons, Eq. (2.29) is a good starting point, but should be used with caution.

Regardless of the density relation used, it is well understood that the atom density in a solid material is generally weakly, but not negligibly, dependent on temperature. However, the stronger dependence of macroscopic one-group cross-section on temperature comes from the temperature dependence of the microscopic cross-section on temperature. Consider a sample total microscopic cross-section plotted in Figure 1 below.



**Figure 1. Uranium-235 Total Microscopic Cross-Section**  
(from Ref. [15])

Figure 1 clearly shows resonance peaks in the energy dependence of the microscopic cross-section. All cross sections of all isotopes exhibit this resonant structure to some degree. The reasons are related to the neutron matter wave function constructive interference with the standing wave of the nucleus. (Ref. [16]). However, for the present analysis it is sufficient to recognize the existence of these peaks, regardless of their physical nature. An increase in temperature of the material results in the broadening of these peaks, called Doppler broadening. This can be associated with the increased vibration frequencies of the nuclides. Therefore, in general, Doppler broadening accounts for an increase in the microscopic cross-sections of the material if temperature is increased, and vice versa. (Ref. [5]).

Few-group macroscopic cross-sections for a reactor are normally generated using a variety of homogenization procedures, resonance absorption treatments, and other energy condensation techniques. Normally, to generate the coarse-mesh few-group cross-sections, a very fine group 1D pin-cell model is first run, which is used to generate the spectrum for a 2D lattice calculation. The lattice calculation then generates the spectrum function  $\phi(\vec{x}, E)$  for the coarse mesh calculations in Eq. (2.26), which results in several few-group cross-section functions. (Ref. [5]). This procedure can, in principle, be repeated several times, at different temperatures, and the resulting cross-section database interpolated, usually using low-order polynomials, to construct an approximate  $\Sigma_i(\vec{x}, T)$  function. A similar approach is taken in most operator splitting-type multiphysics code packages, such as Ref. [17].

Other approaches to the construction of  $\Sigma_i(\vec{x}, T)$  are also possible; in general, the few-group cross-section generation is an important and complicated part of reactor analysis. However, in this work, the focus will be on the transient coupling of the neutron and heat transport, not on the cross-section generation. For that reason, in this text, the temperature-dependent cross section functions like  $\Sigma_i(\vec{x}, T)$  will be assumed to be given. This is also consistent with many benchmark problems, in which the few-group cross-sections are typically

given explicitly. (Ref. [18]). For real reactor analysis, the cross-section generation will have to be incorporated in the modeling code; however, this is not part of the present work.

To summarize: while the atom density dependence on temperature can be modeled and/or tabulated relatively easily, the microscopic cross-section thermal dependence, which accounts for the bulk of thermal feedback in a nuclear reactor, is far more complicated. For that reason, one-group macroscopic cross-sections used in this text will be assumed to be given as explicit functions; in the future, their construction will have to be incorporated into the model.

Recognizing the two types of coupling described above, Eqs. (2.3), (2.5), (2.23) and (2.24) can be rewritten in their coupled form:

$$\frac{\partial}{\partial t} n(t, x) = -\frac{\partial}{\partial x} J(t, x) + \nu \Sigma_f(x, T) \phi(t, x) - \Sigma_a(x, T) \phi(t, x) + s_{ex}(t, x) \quad (2.30)$$

$$J(t, x) = -D(x, T) \frac{\partial}{\partial x} \phi(t, x) \quad (2.31)$$

$$\frac{\partial}{\partial t} u_v(t, x) = -\frac{\partial}{\partial x} u_v''(t, x) + w \Sigma_f(x, T) \phi(t, x) + \dot{u}_{v,ex}(t, x) \quad (2.32)$$

$$u_v''(t, x) = -k(x, T) \frac{\partial}{\partial x} T(t, x) \quad (2.33)$$

Equations (2.30)-(2.33), and their corresponding supportive expressions for  $D$ ,  $\phi$  and  $T$  constitute the primary model of interest in this work. More complicated models will be discussed, but this 1D one-group coupled thermal and neutron diffusion model will be studied in most detail.

## 2.4. Bond Graph Formalism Theory

In this text, nuclear reactor core, the nuclear plant itself, and multiple other engineering structures are viewed as dynamic engineering systems. The definition of the word “*system*” is adapted from Ref. [19], and implies that the system is composed of connected interacting parts and is conceptually or physically separate from its environment. Examples of dynamic systems, not necessarily limited to engineering, include an animal (composed of organs and fluids), a financial system, a *mechatronic* device or a reactor core. The components of larger systems, such as the nuclear reactor core which is a component of a nuclear power plant, can be viewed to be engineering systems themselves. In theory, this subdivision is nearly endless, and can continue down to individual atoms which themselves are physical systems.

Numerous algorithms exist for constructing and analyzing mathematical models for various classes of engineering systems. In this context, a “*mathematical model*” is a set of mathematical relations which can be solved to obtain a meaningful description of the system’s dynamic or stationary behavior. This definition is adapted from Ref. [19].

Generally, in the analysis of an engineering system, a necessary step is the representation of the system via some more or less abstract formalism, from which the mathematical model can be derived. (Ref. [19]) Brown (Ref. [20]) refers to these formalisms as “modeling languages.” Some examples of these formalisms include:

- Operational block diagram
- Electric circuit schematic
- Kinematic diagram
- Dynamic schematic
- Free-body diagram
- Piping network schematic

– General schematic diagram

These formalisms were generally developed for specific types of systems, thus each formalism can only cover a specific *physical domain*. The range of application of the formalisms listed above is given in Table 1 below.

**Table 1. Range of Application of Modeling Formalisms**

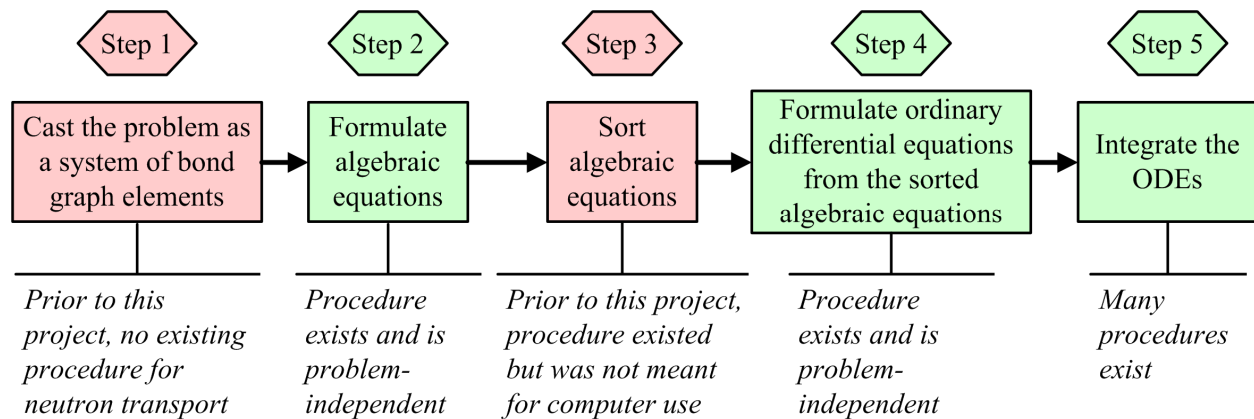
<b>Formalism</b>	<b>Types of systems modeled</b>	<b>Physical effects modeled</b>
Operational block diagram	Directed signal flow systems (i.e., automatic control systems)	All possible in directed signal flow systems
Electric circuit schematic	Electric circuits and networks	All present in electric circuits
Single-body diagram	Bodies under applied forces and torques	Free-body dynamics or loaded-body stresses
Kinematic schematic diagram	Rigid-body linkages and mechanisms	Rigid-body kinematics (position, velocity, acceleration) only
Dynamic schematic diagram	Rigid-body linkages and mechanisms with compliant and damping elements	Rigid-body dynamics (kinematics and loads)
Piping network schematic	Systems of pipes	Pressure, flow rate and thermal fluid dynamics
General schematic diagram	Systems with multiple physical domains	Varies

All of the above formalisms, except for the general schematic diagram, have algorithmic procedures for formulating the mathematical models describing the system. For example, the implementation of Kirchoff's laws or nodal voltage methods results in the mathematical models for electric circuits. This is the basis of many codes for circuit analysis, like SPICE. (Ref. [21]) As a rule of thumb, the implementation of the more general formalisms is less algorithmic. The general schematic diagram is the only formalism out of the ones listed here capable of describing multi-domain systems. However, it is not standardized and there exists no formal algorithm for formulating equations from a general schematic diagram. For these reasons, the general schematic diagrams are primarily only used for illustrating complex systems, and not for actual mathematical modeling.

None of the conventional formalisms, such as the ones listed above, are fit for detailed dynamic modeling of mechatronic systems. For this reason, in the 1960s, a new formalism for modeling engineering system dynamics was developed by Henry Paynter of MIT. (Ref. [2]) This formalism was named the Bond Graph Method, or simply Bond Graphs.

Over time, the bond graph formalism grew from a comprehensive methodology to model mechatronic systems into a complete research field, concerned with modeling mechanical, electrical, magnetic, hydraulic, thermal, and even optical and financial systems. (Ref. [20]). In this section, the formalism is described in its current form, and it is developed further in the next chapters.

Fundamentally, the process involved in modeling a system via bond graph formalism is summarized in Figure 2 below. One can see that the formalism is generally similar to those listed above. The main difference is in the first step: the bond graph system is capable of representing multiple physical domains, without restriction to a particular type of engineering system. This makes the bond graph formalism ideal for multiphysics simulations.



**Figure 2. Bond Graph Formalism Summary**

A *bond graph system* (BGS), like the formalisms listed above, completely describes the engineering system it models. It does so by representing the system using two types of fundamental entities: *bond graph elements* connected by *bonds*. The bonds carry bond variables between the elements' ports, and the elements' constituent equations impose the bond variables onto the bonds as functions of time, other bond variables and state variables.

Every bond in a bond graph system is associated with distinct “effort” and “flow” variables. Bonds are typically numbered, for convenience. Effort on bond  $i$  is denoted  $e_i$ , and flow on that same bond is denoted  $f_i$ . All efforts and flows in a BGS are referred to in this text as *bond variables*.

A bond always points from one element to another element. In an *augmented* bond graph system a bond also has an associated *causality*, which determines which way the bond delivers the effort and the flow. A bond graph system without assigned causalities is called *acausal*. The meaning of the direction in which the bond points is explained in detail below; in general, the direction of positive flow (of conserved quantity – usually energy) across a bond is in the direction the bond points. The four possible configurations of bond direction and causality are given in Table 2 below. The direction of the half-arrow (by convention, regular bonds have half-arrows, and special *signal bonds* have full arrows) is the direction of the positive flow. The direction of the causal stroke (the short mark at the end of the bond) is the direction of causality, explained in Table 2.

**Table 2. Causality-Direction Configurations**

Configuration	Causality	Direction of positive flow
A $\longrightarrow$ B	Effort is delivered from A to B Flow is delivered from B to A	Positive flow is from A to B
A $\longleftarrow$ B	Effort is delivered from B to A Flow is delivered from A to B	Positive flow is from A to B
A $\longleftarrow$ B	Effort is delivered from B to A Flow is delivered from A to B	Positive flow is from B to A
A $\longrightarrow$ B	Effort is delivered from A to B Flow is delivered from B to A	Positive flow is from B to A



Generally, causality is not inherently a part of the model, and is instead assigned to the bonds in a BGS using an augmentation procedure. Augmentation procedures and signal bonds are used in a later part of this text. For now, it suffices to say that augmentation procedures are automatic, and require no user input, while signal bonds are special bonds often representing nonlinear coupling. The most common and simplest augmentation procedure is SCAP: Sequential Causality Assignment Procedure, described in detail in Ref. [19]. It is summarized below in this section, following a brief discussion of the bond graph elements. Its usage is implied when constructing bond graphs of physical systems throughout this text.

The most important decision about modeling a new physical domain with bond graphs is deciding what the “effort” and “flow” variables represent. In most energy-related domains, these choices have long been established, and are summarized in Table 3 below.

**Table 3. Bond Variables in Various Physical Domains**

<b>Domain</b>	<b>Effort variable</b>	<b>Flow variable</b>
Mechanical translational	Force	Velocity
Mechanical rotational	Torque	Angular velocity
Electric circuit	Voltage	Electric current
Hydraulic	Pressure	Volumetric flow rate
Thermal conduction	Temperature	Heat flow

This table shows that generally, flow goes from concentrations of higher to lower effort, which is how energy transfer occurs. For all of the above physical domains except thermal conduction, the following equation also holds:

$$P_i = e_i f_i \tag{2.34}$$

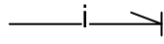
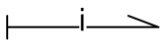
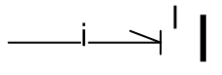
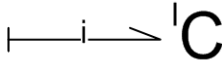
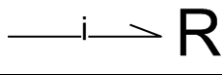
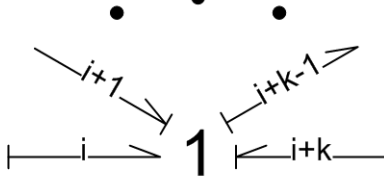
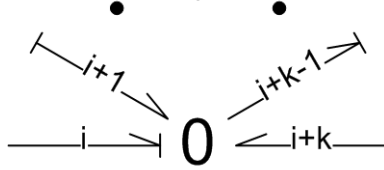
in which:

$P_i$	Power carried by bond $i$
$e_i$	Effort on bond $i$
$f_i$	Flow on bond $i$

Equation (2.34) does not hold for the choice of variables for thermal conduction as described in Table 3, because  $T\dot{U}$  ( $T$  – temperature,  $\dot{U}$  – heat flow) does not have dimensions of power. Rather, the flow variable itself – the heat flow – is the power carried by the bond. For this reason, when the thermal bonds were first introduced, they were called pseudo-bonds. (Refs. [22-24]). An alternative way to represent thermal conduction, and more generally, transient thermodynamics, is to use  $T$  as the effort variable, and  $\dot{S}$  – entropy flow rate – as the flow variable. This representation is used in many conventional bond graph texts, such as Refs. [20,25]. It is particularly convenient for lumped thermodynamic analysis, in which it has been extended to account for both energy and mass conservation using *convection bonds*. (Ref. [26]). However, the thermodynamic representation is less convenient for treating discretized field problems important in nuclear analysis. Therefore, in this text, the pseudo-bond graph approach shown in Table 3 is used, because it is simpler and generally easier to work with.

Table 4 below summarizes the basic types of elements used in this text, along with the number of ports they have and the general form of their constituent equations. The specific values/functions used in these equations will be discussed further in the text. The notation used in this table and in this text is generally adapted from Refs. [19,20], with exceptions in the storage element notation and the element numbering. Also, in Ref. [20], the symbol  $\dot{q}$  is used instead of  $f$  to represent flow.

**Table 4. Basic Elements**

Element	Element name	Type of element	Constituent equation(s)
SE 	Source of Effort	Source	$e_i = A$
SF 	Source of Flow	Source	$f_i = A$
	Inertial Element	Storage	$f_i = I(p_i)$ $\dot{p}_i = e_i$
	Capacitive Element	Storage	$e_i = C(q_i)$ $\dot{q}_i = f_i$
	Resistive Element	Damping	$e_i = R(f_i)$ or $f_i = G(e_i)$ depending on causality
	1-junction	Junction	$f_i = f_{i+1} = \dots = f_{i+k-1} = f_{i+k}$ $\sum_{l=i}^{i+k} d_l e_l = 0$
	0-junction	Junction	$e_i = e_{i+1} = \dots = e_{i+k-1} = e_{i+k}$ $\sum_{l=i}^{i+k} d_l f_l = 0$

Note:  $A$  is a constant.  $d_l \equiv \begin{cases} +1 & \text{for bonds directed toward the junction} \\ -1 & \text{for bonds directed away from the junction} \end{cases}$

Many special elements also exist, a lot of which are described in Refs. [19,20]. Generally, they are combinations of the basic 7 elements listed in Table 4 above. They are introduced later in the text. Bond graphs with these special elements are analyzed in the same way as bond graphs with only the basic elements.

Here every element type is discussed. All elements are shown in the table in integral/appropriate causality. Integral causality is discussed below.

### Source Elements

Source elements impose either a flow (SF) or an effort (SE) on the bond connected to them. This implies that the bonds connected to these elements are automatically assigned the causality shown in Table 4. In certain texts, regular source elements can sometimes be time-modulated, that is,  $A = A(t)$ , however, in this work, time-modulated elements are referred to as *modulated elements*, and will be introduced in section 3.3 (p. 57).

Source elements have exactly 1 bond connected to them, and therefore have exactly 1 constituent equation associates to them.

In energy bond graphs, source elements represent the physical inputs of energy into the system – either by setting a constant potential (sources of effort) or by imposing a constant current (sources of flow). Since the source elements’ imposed quantities are independent of the resistance encountered, the sources are potentially capable of putting out infinite power. Thus, in energy bond graphs, they represent idealized generalizations, similar to ideal batteries in an electrical circuit (SE element) or perfect thermal insulators (SF element imposing zero flow).

During SCAP, the source elements’ bonds are augmented first, since their causality is prescribed. If a causal conflict arises during this assignment, this corresponds to a major modeling flaw, like trying to enforce two dissimilar temperatures at one point.

### Storage Elements

Storage elements are the only elements that have variables associated to them, same way bonds do. These variables are called displacement variables for Capacitive elements, and momentum variables for Inertial elements. The symbols used for these are conventionally  $q$  for displacement and  $p$  for momentum. These variables are sometimes affiliated with the bond connected to the element (Ref. [19]), and not with the element itself, but in this text the storage variable  $q_i$  (or  $p_i$ ) corresponds to the storage element  $^iC$  (or  $^iI$ ).

A general capacitive element imposes an effort on its bond as a function of its displacement. A general inertial element imposes a flow on its bond as a function of its momentum. Storage elements have exactly 1 bond connected to them, but have 2 constituent equations. One of these equations is algebraic, similarly to all other elements, while the other is differential, and relates the bond variable delivered to the element to the displacement or momentum of the element. Therefore, the number of differential state equations in a bond graph system corresponds to the number of storage elements in integral causality.

The storage elements in Table 4 are shown in integral causality. This means that each element contributes a state variable to the system’s state vector, which means that there will be an ordinary differential equation associated with this element. In integral causality, each storage element has two associated equations: one differential equation which becomes a state equation for the system, and one algebraic equation. For a capacitive element in integral causality, these equations are (from Table 4):

$$e = C(q) \quad (2.35)$$

$$\dot{q} = f \quad (2.36)$$

We can see that, for a storage element in integral causality, the state derivative is integrated in time to evaluate the bond variable enforced by the capacitive element. This is the origin of the term “integral causality.”

Storage elements can also be in derivative causality – in that case, effort  $e$  is delivered to a capacitive element, and flow  $f$  is delivered to an inertial element (see Table 2 above for associated causal strokes). In general, storage elements in derivative causality are much more complicated to treat, since while they store a conserved quantity, the bond variable that is delivered to them does not serve as a state derivative variable. Instead, to compute their “state” variable’s derivative, the time derivative of the bond variable supplied to the storage element has to be evaluated. This is the origin of the term “derivative causality.” For a capacitive element in derivative causality, Eqs. (2.35) and (2.36) are inverted:

$$q = C^{-1}(e) \quad (2.37)$$

$$f = \dot{q} \quad (2.38)$$

Equation (2.37) assumes that the capacitive element's function is invertible; if it's not, an additional algebraic equation which evaluates  $q$  implicitly is required. Equation (2.38) is the differential equation associated with every capacitive element, but because the element is in integral causality, the time derivative term is treated as the known term in the equation. To relate the time derivative to other terms in the system, we take a time derivative of Eq. (2.37) and plug it into Eq. (2.38):

$$f = \dot{q} = \frac{\partial C^{-1}(e)}{\partial t} = \frac{\partial C^{-1}(e)}{\partial e} \dot{e} \quad (2.39)$$

The time derivative  $\dot{e}$  generally results in making implicit one or more of the state equations of the system, since  $e$  itself depends on the system's state. Additionally, the derivative  $\partial C^{-1}(e)/\partial e$  may be a complicated and/or unstable numerically.

Implicit ordinary differential equations (ODEs) are generally more computationally expensive to integrate, since an explicit ODE vector can be turned implicit for a more efficient time integration, but the reverse is generally impossible.

For these reasons, if possible, derivative causality is avoided. In this text, the bulk of the work is with BGSs that arise from field problem discretizations, and therefore all storage elements end up in integral causality. This is a very significant and useful restriction on the nature of problems tackled.

During SCAP, the storage elements receive their causal assignments (preferably integral) immediately after the storage elements receive theirs and the assignments are propagated through the junction elements. Derivative causality assignments can happen if either a storage element's causality assignment forces a derivative causality (when propagated through the junction elements) onto a storage element, or when an integral causality assignment on a storage element propagate the derivative causality onto another storage element.

Another rare occurrence is what's called a "bond graph mesh," described in detail in Ref. [20]. In a mesh, attempting to put an integral causality on a storage element can result in a contradiction – causalities propagate through the junction elements and contradict each other on some bond. This is a case of SCAP failure, and a more complicated augmentation procedure becoming necessary. This case, and how to handle it, is described in Ref. [20]; in vast majority of physical models, including all models in this text, bond graph meshes do not happen, and so we can assume SCAP to always succeed.

Storage elements can also be multiport, but such storage elements are not used in this text. Interested readers are referred to Refs. [19,20].

## Damping Elements

Damping elements, or resistive elements, are elements which control the rate of transfer and dissipation of the conserved quantity. Resistive elements in general do not have a preferred causality: as shown in Table 4, the resistive element can enforce either an effort or a flow. For a resistive element without a preferred causality, the following holds:

$$R(f) = G^{-1}(f) \quad (2.40)$$

$$G(e) = R^{-1}(e) \quad (2.41)$$

However, physical constraints will sometimes force the algebraic equation to a given form, thus only one causality can be written; this happens when the resistive element's constitutive function  $R(f)$  or  $G(e)$  is not invertible and so Eqs. (2.40) and (2.41) are undefined. This causality restriction is not reflected in the bond graph, but is simply a property of the element's constitutive relation. A modified version of SCAP, described in Ref. [27], allows the

user to force specific causalities on the resistive elements. This comes at the cost of additional algebraic equations accompanying the state derivative vector, thus turning the system of ODEs into a system of Differential-Algebraic Equations (DAEs). The addition of algebraic equations also happens when causality to resistive elements remain unassigned. SCAP has provisions for handling these unassigned elements, and the modified SCAP from Ref. [27] handles those elements formally, with the same effect. In the context of this work, the nature of the bond graphs handled has no instance of this algebraic loop problem caused by unassigned resistive elements, since all BGSs handled are fully causal.

Multiport resistive elements also exist, and are described in section 3.3 (p. 57). They are essentially connected single-port resistive elements with the individual constituent equations replaced by several of them, one for every port. It should be noted that resistive elements are, by definition, non-conservative – they may dissipate the conserved quantity provided to them. Multiport elements are also capable of doing so, and are sometimes called “non-conservative couplers.”

### **Junction elements**

The junction elements used in this text are the 1-junction and the 0-junction. In general, junction elements propagate and distribute the conserved quantity, without storing, generating or dissipating any of it, and without impeding its rate of transfer. As with the sources of effort/flow, and the capacitor/inertial element, the 1 and 0-junctions are “dual elements.” Specifically, across a 1-junction, all efforts add to 0 and all flows are equal, and over a 0-junction, all flows add to 0 and all efforts are equal. For this reason, sometimes the 1-junction is referred to as the “series junction” (a single current over all elements, net potential over the circuit is zero), and the 0-junction is referred to as the “parallel junction” (the currents entering the node are the currents coming out, and the potential drop is the same across all connected parallel branches). For this reason, these elements are sometimes denoted by “s-junction” and “p-junction,” respectively.

The causality requirements on junction elements are stringent: exactly one bond on a 1-junction may deliver flow to the junction, and, symmetrically, exactly one bond on a 0-junction may deliver effort to the junction. So, for example, a source of effort connected to 0-junction automatically makes the junction enforce efforts on all other bonds connected to it. The causality assignments may thus travel through the BGS’s junction structure, and may in principle conflict, if a specific bond graph mesh is encountered.

Junction elements are by definition multiport elements. In fact, once the augmentation procedure is completed, the entire junction structure can be thought of as a large multiport junction element. This view was utilized by Rosenberg in an early attempt to sufficiently formalize step 2 of the algorithm in Figure 2 – the equation formulation step. (Ref. [28]). The algorithm developed in Ref. [28] does not work for nonlinear systems, and so cannot be utilized in this text.

The other two basic junction element types are the transformer (TF) and gyrator (GY) elements. These are not used in this work, and are not described in Table 4 above, but together with the 7 elements described in Table 4 they form the so-called “9 basic bond graph elements.” (Ref. [19]). It is worth noting that some of the 9 basic elements themselves can be viewed as combinations of other elements: for example, an appropriately picked SE and GY pair is identical to an SF element, and a C and GY pair is identical to an I element. Nonetheless, these 9 elements are considered the basic building blocks of bond graph models; more complicated elements are very frequently combinations or multiport generalizations of the basic 9 elements.

## Sequential Causality Assignment Procedure

The sequential causality assignment procedure, SCAP, is quoted verbatim from Ref. [19]:

1. Choose any source (SE, SF), and assign its required causality. Immediately extend the causal implications through the graph as far as possible, using the constraint elements (0, 1, GY, TF).
2. Repeat step 1 until all sources have been assigned.
3. Choose any storage element (C or I), and assign its preferred (integration) causality. Immediately extend the causal implications through the graph as far as possible, using the constraint elements (0, 1, GY, TF).
4. Repeat step 3 until all storage elements have been assigned a causality. In many practical cases all bonds will be causally oriented after this stage. In some cases, however, certain bonds will not yet have been assigned. We then complete the causal assignment as follows:
5. Choose any unassigned R-element and assign a causality to it (basically arbitrary). Immediately extend the causal implications through the graph as far as possible, using the constraint elements (0, 1, GY, TF).
6. Repeat step 5 until all R-elements have been used.
7. Choose any remaining unassigned bond (joined to two constraint elements), and assign a causality to it arbitrarily. Immediately extend the causal implications through the graph as far as possible, using the constraint elements (0, 1, GY, TF).
8. Repeat step 7 until all remaining bonds have been assigned.

...

There are several situations that can arise when applying causality according to the procedure given:

1. All storage elements have integral causality, and the graph is complete after step 4...
2. Causality is completed by using R-elements or bonds, as indicated in steps 5-8...
3. Some storage elements are forced into differentiation causality at step 3...

In this text, only the simple case of “full causality” is encountered, due to the physical origin of the bond graphs built. For this reason, we do not have to explicitly run an augmentation procedure during the analysis: we only need to know the causalities on the ports of the resistive elements, which can be declared manually when building the bond graph. The reason is, upon inspection of Table 4, that if we assume integral causality, the equations of all other elements except for resistors do not explicitly depend on the causality of the connected bonds. Therefore, if we know that a BGS is fully causal in advance, and we know the causality on resistive elements’ bonds, SCAP can be omitted.

The basic elements described in this section can be viewed as generalizations of the various equivalence concepts, such as the mass and heat diffusion analogy, the electrical circuit analogy used in heat transfer, or the analogy between oscillating mechanical systems and oscillating electrical circuits. The roles of the basic elements in several physical domains, adapted from Ref. [20], are listed in Table 5 below:

**Table 5. Basic Elements in Various Physical Domains**

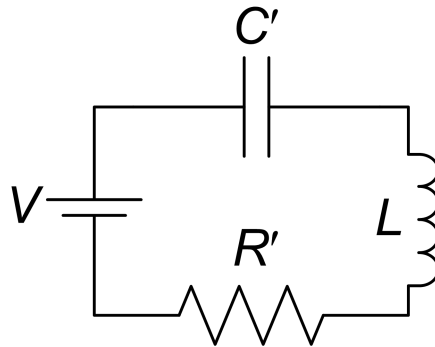
Element type	Corresponding physical element <sup>6</sup>		
	Electrical circuit	Mechanical translational	Mechanical rotational
Source of Effort	Voltage source	Linear actuator	Torque motor actuator
Source of Flow	Current source	Linear servo actuator	Servo motor
Inertial element	Inductor	Mass	Flywheel
Capacitive element	Capacitor	Linear spring	Torsional spring
Resistive element	Resistor	Damper	Rotational damper
1-junction	In-series wiring	No one-to-one correspondence, but the junctions are still used to connect other elements	
0-junction	In-parallel wiring		

In continuous systems that first need to be discretized, the elements play a more mathematical role, and do not have one-to-one analogies with physical devices. However, in discrete physical systems, the one-to-one correspondences exist and allow for direct construction of bond graph representations of physical systems. Most engineering systems are combinations of discrete networks of devices and continuous fields, and the bond graph formalism can represent these systems after the continuous fields have been discretized.

The principles described in this section will now be illustrated using an example.

### 2.5. Bond Graph Formalism Example

Consider a series RLC circuit in Figure 3 below. The values  $V$ ,  $C'$ ,  $L$  and  $R'$  denote the battery voltage, capacitor's capacitance, inductor's inductance and resistor's resistance, respectively. In this section, this system is fully analyzed using the bond graph formalism, following the basic steps of the bond graph process summarized in Figure 2 above.



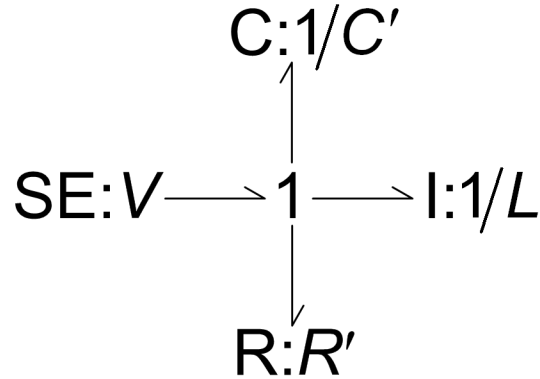
**Figure 3. Series RLC Circuit Schematic**

#### Step 1. Casting the problem as a bond graph system

In general, the casting of a physical system as a bond graph system starts with a discretization of the system. In many cases, including this one, and networks in general, the system is already fully discrete, so there is no need for discretization.

References [19,20] describe algorithms for constructing bond graph representations for general electrical schematics. In this simple case, it is sufficient to recognize that the four circuit elements are connected in series, and by inspection of Table 5 above, construct the following bond graph representation:

<sup>6</sup> These are examples only, other physical roles are possible.



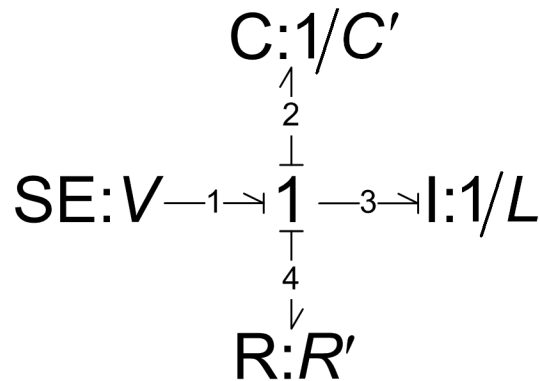
**Figure 4. Series RLC Circuit Bond Graph Representation**

Here the 1-junction represents the in-series connection, the source of effort represents the fixed-voltage battery, the capacitive element represents the electrical capacitor, the inertial element represents the inductor, and the resistive element represents the resistor.

**Step 2. Formulation of algebraic equations**

Normally, before we formulate the algebraic equations we should attempt to reduce the bond graph. This is a complicated process which pays off later by reducing computational complexity. It relies heavily on graph theory. There is a strong mathematical theory behind the equivalences of various bond graph representations, studied in detail in Refs. [29,30], and applied to an example in Ref. [31]. Fundamentally, bond graph reduction involves replacing the bond graph diagram with a diagram that will yield the same solutions for the same state variables. References [19,20] list several collections of equivalent bond graph structures, which involve replacing junction structures with smaller junction structures. Linear constant coefficient elements also allow combining several resistive, capacitive or inertial elements into one element. In circuit theory this corresponds to equivalent resistances, capacitances and inductances of series and parallel circuits.

In this case, the BGS in Figure 4 is already fully reduced, so we do not need to reduce it further. The next step is to augment the BGS. Using the SCAP described in section 2.4, and numbering the bonds for convenience, we arrive at the following augmented BGS:



**Figure 5. Series RLC Circuit Augmented Bond Graph Representation**

By comparing the augmented BGS above to Table 4 (p. 34), we can see that this BGS has full integral causality. Therefore, we can now follow Table 4 to formulate the algebraic



equations. Typically the state variables are numbered, but here we just refer to them as  $q$  and  $p$ , since there is only 1 capacitor and 1 inertial element. Physically, the displacement  $q$  corresponds to the charge stored on its capacitor, and the momentum  $p$  corresponds to the flux linkage on the inductor (Ref. [21]).

The algebraic equations for this bond graph are formulated using Table 4, and are listed in Table 6 below. Here, the relationships are linear and constant coefficient, but in general, more complex algebraic relationships may exist; the most general relationship would involve a table look-up function. One of the virtues of the bond graph representation is that the representation itself allows us to identify the kinds of connections between the elements, without dealing directly with the underlying equations. In this case, one of the elements – often the resistor – could be nonlinear, possibly due to a large expected current, or extreme operating conditions. However, the representation in Figure 5 would not change, since the element is still there; the constituent equation of the element would be different, but that can be addressed later in the modeling.

Notice that the differential equations are not yet involved – those will come into play in step 4, after the algebraic equations are sorted in step 3.

**Table 6. Series RLC Circuit Equations**

<b>Element</b>	<b>Equation(s)</b>
SE	$e_1 = V$
1-junction	$f_1 = f_2$
	$f_1 = f_3$
	$f_1 = f_4$
	$0 = e_1 - e_2 - e_3 - e_4$
C	$e_2 = (1/C')q$
I	$f_3 = (1/L)p$
R	$e_4 = R'f_4$

### Step 3. Sorting of algebraic equations

During the sorting step, we are trying to solve the equations from step 2 for the bond variables in terms of time and state variables ( $p$  and  $q$ ). In this case, the system is small and simple to solve:

$$\begin{aligned}
 e_1 &= V \\
 e_2 &= \frac{1}{C'}q \\
 e_3 &= V - \frac{1}{C'}q - \frac{R'}{L}p \\
 e_4 &= \frac{R'}{L}p \\
 f_1 &= f_2 = f_3 = f_4 = \frac{1}{L}p
 \end{aligned} \tag{2.42}$$

#### Step 4. Formulation of ODEs

Now we can formulate the ODEs, following the rules for storage elements from Table 4. This involves only retaining the algebraic equations whose flows and efforts are the time derivatives of the state variables. This yields:

$$\begin{aligned} \dot{q} &= \frac{1}{L} p \\ \dot{p} &= V - \frac{1}{C'} q - \frac{R'}{L} p \end{aligned} \tag{2.43}$$

These are the state equations for this system. They can alternatively be written in the form  $\dot{\bar{\mathbf{x}}} = \bar{\mathbf{f}}(t, \bar{\mathbf{x}})$  where  $t$  is time and  $\bar{\mathbf{x}} = [q \ p]^T$  is the state vector. Integrating these ODEs yields  $\bar{\mathbf{x}}(t)$ , which is the state of the system. Finding  $\bar{\mathbf{x}}(t)$  completes the model of the system.

#### Step 5. Integration of ODEs

The ODEs are usually integrated numerically; the method is independent of the bond graph formalism. Their integration yields  $\bar{\mathbf{x}}(t)$ . In this case, the ODEs are simple enough to be integrated analytically. Applying analytical linear ODE system solution techniques (Ref. [32]), we can obtain the general solution.

The solution can then be post-processed. When the state of the system as a function of time  $t$  is known, the solutions of algebraic equations (see Eq. (2.42)) can be used to obtain any information about the system. In this case, variables  $q$  and  $p$  are the charge stored on the capacitor and the flux linkage on the inductor, so if they are known as functions of time, other quantities, like the current through the battery  $f_1$ , or the voltage drop across the resistor  $e_4$ , can be algebraically evaluated at any time.

This example concludes the summary of bond graphs. Although not obvious from the example, the real benefit of the bond graphs is the capability to couple any number of physical domains together, and process them all simultaneously, processing their bond graph system. This is generally superior to the operator-splitting approach typically implemented in most modern multiphysics codes.

### 3. Coupled Neutron and Thermal Diffusion via Bond Graphs

As discussed in section 2.4 (p. 30), to model coupled physics using bond graphs, the physics are individually represented, and the bond graph representations are then connected by coupling bonds. In this chapter, bond graph representations of 1D thermal diffusion are first constructed in section 3.1. Then, 1D neutron diffusion is represented with bond graphs in section 3.2, and the models are coupled in section 3.3. In section 3.4, potential representations for more complicated physics are discussed.

#### 3.1. Thermal Diffusion via Bond Graphs

Two types of bond graph representations of thermal diffusion exist in literature: thermodynamic true bond graphs, and thermal pseudo-bond graphs. For reasons discussed in section 2.4 (p. 30), in this work thermal pseudo-bond graph representation is used.

As discussed in section 2.2 (p. 22), 1D thermal diffusion in a solid material is modeled by the following equations:

$$\frac{\partial}{\partial t} u_v(t, x) = -\frac{\partial}{\partial x} u_v''(t, x) + \dot{u}_{v,ex}(t, x) \quad (3.1)$$

$$u_v''(t, x) = -k(x, T) \frac{\partial}{\partial x} T(t, x) \quad (3.2)$$

$$T(u_v) = \frac{1}{\rho c_p} u_v \quad (3.3)$$

Boundary conditions of some sort are required; in this study, Dirichlet boundary conditions will be used. Neumann and mixed boundary conditions could also be easily implemented, so Dirichlet boundary conditions are only treated here as an example. Initial condition function  $u_v^0(x)$  is also assumed given. If an initial temperature distribution  $T^0(x)$  is specified instead, Eq. (3.3) can be used to derive  $u_v^0(x)$ .

The primary application of bond graphs has always been for discrete systems modeling with well-defined element boundaries. A lumped element heat conduction model is a form of such discrete system. Even prior to bond graph formalism being introduced, the notion of thermal resistances and capacitances has been used widely in thermal system modeling. The lumped volume discussion of thermal pseudo-bond graph representations below is adapted from Ref. [19].

The lumped element approach typically used in thermal pseudo-bond graph models works only for uniform material properties; that is, constant  $k$ ,  $\rho$  and  $c_p$ . In lumped heat transfer analysis, thermal capacitance of a lumped volume is defined as the ratio of the volume's average temperature to the thermal energy stored in the volume. This definition of capacitance is the reciprocal of the electrical definition, which is the ratio of the charge (thermal energy) stored on the capacitor to the voltage imposed by it (temperature). Thermal resistance between two lumped volumes is defined as the ratio of the difference in temperatures of the two volumes to the heat flow rate between them. The corresponding equations for these definitions are:

$$T = CU \quad (3.4)$$

$$\dot{U}_{j \rightarrow i} = \frac{\Delta T_{j \rightarrow i}}{R} \quad (3.5)$$

in which:

$T$	Average temperature in the lumped volume
$C$	Thermal capacitance

$U$	Total thermal energy in the lumped volume
$\dot{U}_{j \rightarrow i}$	Heat flow rate from lumped volume $j$ to lumped volume $i$
$\Delta T_{j \rightarrow i}$	Temperature difference between lumped volume $j$ and lumped volume $i$
$R$	Thermal resistance

Using the above assumptions about the material properties, the values of the capacitance and the resistance are given by:

$$C = \frac{1}{\Delta V \rho c_p} \quad (3.6)$$

$$R = \frac{\Delta x}{kA} \quad (3.7)$$

in which:

$\Delta V$	Volume of the lumped volume
$\Delta x$	Distance across which the heat transfer occurs
$A$	Cross-sectional area through which the heat transfer occurs

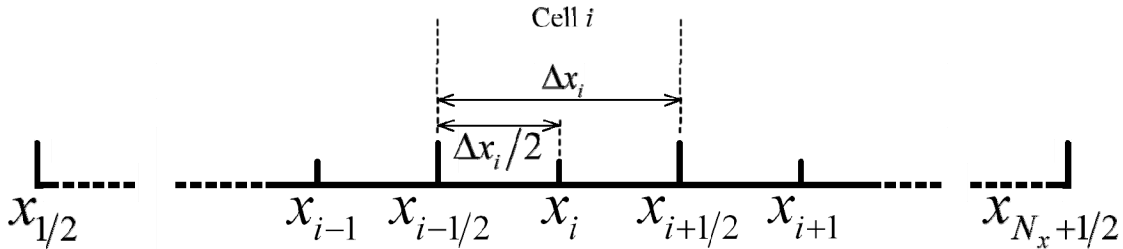
By conservation of thermal energy, the overall rate of change of thermal energy in a lumped volume is the sum of heat flows in and out of the volume, that is:

$$\dot{U}_i = \sum_{\text{all } j} \dot{U}_{j \rightarrow i} - \sum_{\text{all } k} \dot{U}_{i \rightarrow k} + \dot{U}_{ex} \quad (3.8)$$

in which:

$\dot{U}_i$	Rate of change of thermal energy stored in lumped volume $i$
$\dot{U}_{j \rightarrow i}$	Heat flow from lumped volume $j$ into lumped volume $i$
$\dot{U}_{i \rightarrow k}$	Heat flow from lumped volume $i$ into lumped volume $k$
$\dot{U}_{ex,i}$	Heat generation rate in lumped volume $i$ from an external heat source

The lumped parameter approach can be applied to a discretized 1D domain. Consider the discretized domain in Figure 6:



**Figure 6. Discretized 1D Domain**

Each finite cell  $i$  can be viewed as a separate lumped volume, connected with thermal resistors to neighboring lumped volumes. In this case, the average distance across which the heat transfer occurs is the distance between the centers of the nearby cells, that is:

$$\Delta x_{i-1/2} = x_i - x_{i-1} = \frac{\Delta x_{i-1}}{2} + \frac{\Delta x_i}{2} \quad (3.9)$$

For boundary cells, the average distance across which the heat transfer occurs is the distance between the boundary cell center and the boundary:

$$\Delta x_{1/2} = \frac{\Delta x_1}{2} \quad (3.10)$$

$$\Delta x_{N_x+1/2} = \frac{\Delta x_{N_x}}{2} \quad (3.11)$$

Using this discretization, the heat generation rate by the external heat source can be computed as a volume integral of the external heat source density over the lumped volume:

$$\dot{U}_{ex,i}(t) = A \int_{x_{i-1/2}}^{x_{i+1/2}} dx u_{v,ex}(t, x) \quad (3.12)$$

The initial conditions  $U_i^0$  can be similarly computed by:

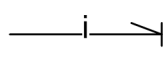
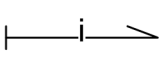
$$U_i^0 = A \int_{x_{i-1/2}}^{x_{i+1/2}} dx u_v^0(x) \quad (3.13)$$

In pseudo-bond graph representation, effort corresponds to temperature, and flow corresponds to heat flow rate. Using this convention and comparing Eqs. (3.4) and (3.5) to Table 4 (p. 34), we can see that they correspond to the constituent equations of a capacitor and a resistor, respectively.  $\Delta T_{j \rightarrow i}$  here is a difference of efforts imposed on a resistor, which itself sets a flow  $\dot{U}_{j \rightarrow i}$ . The heat flows supplying and removing thermal energy from a lumped volume are the flows to and from the neighboring lumped volumes, or the boundary. Therefore, we can view the thermal resistors as connecting two neighboring finite cells. The flow entering the thermal resistor is the flow leaving it, by energy conservation, so a thermal resistor is represented by a resistor connected to a 1-junction. The resistor sets its flow by Eq. (3.5), while the 1-junction provides the difference of efforts.

$U$  is a displacement on a capacitor, whose rate of change is the flow supplied to the capacitor, which is set by Eq. (3.8). Equation (3.8) is a form of flow balance equation, which is imposed by a 0-junction.

As stated above, Dirichlet boundary conditions are assumed. This means that functions  $T_{left}(t)$  and  $T_{right}(t)$ , the temperatures at the left and right boundaries, respectively, are given. These are externally imposed efforts, dependent only on time. Similarly, the flow contributed to a volume by the external source  $\dot{U}_{ex,i}(t)$  is also an externally imposed quantity, dependent only on time. Both of these have the form similar to sources of effort and flow, respectively, except that the efforts and flows imposed here are time dependent. Therefore, time-modulated source elements are now introduced:

**Table 7. Time-modulated Source Elements**

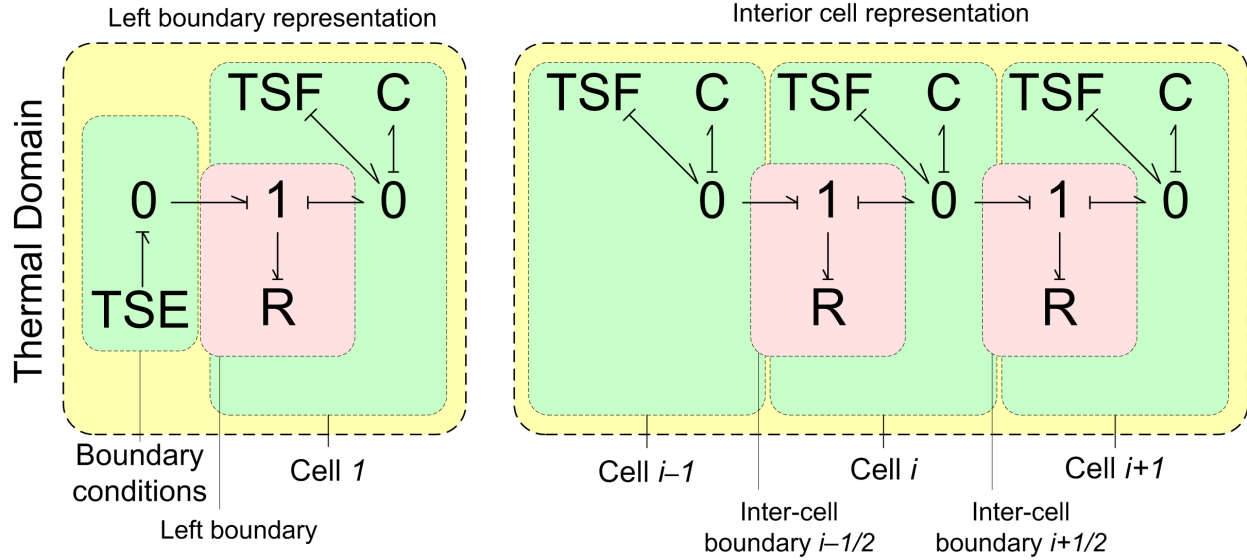
Element	Element name	Type of element	Constituent equation(s)
TSE 	Time-modulated Source of Effort	Source	$e_i = A(t)$
TSF 	Time-modulated Source of Flow	Source	$f_i = A(t)$

The Dirichlet boundary conditions and the external sources clearly have the form of the time-modulated sources of effort and flow, respectively.

Combining all these interpretations, the bond graph representation of a 1D thermal diffusion problem using lumped finite volumes can be constructed. It is given in Figure 7 below.

The constituent expressions for all elements in this bond graph representation are summarized in Table 8 below. Figure 7 and Table 8 together summarize the bond graph

representation algorithm which arises from lumped volume representation. The initial conditions for the capacitors' displacements have been specified in Eq. (3.13).



**Figure 7. Heat Diffusion Bond Graph Representation**

**Table 8. Heat Diffusion Bond Graph Constituent Expressions with Uniform  $k$**

Element <sup>7</sup>	Constituent Expressions <sup>8</sup>
$TSE_{left}$	$T_{left}(t)$
$R_{left}$	$\frac{\Delta x_1}{2kA}$
$C_i$	$\frac{1}{\Delta V_i \rho c_p}$
$R_{i-1/2}$	$\frac{\Delta x_{i-1} + \Delta x_i}{2kA}$
$TSF_i$	$\dot{U}_{ex,i}(t)$

As stated above, the problem with the lumped volume representation is that it assumes spatially uniform and constant  $k$ ,  $\rho$  and  $c_p$ . For certain problems, these assumptions are unacceptable. Fortunately, a more rigorous mathematical treatment of Eqs. (3.1)-(3.3) can be used, which utilizes the finite volume discretization method. (Ref. [33]). It results in a discretized formulation which can be represented with bond graphs. Such a treatment accounts for variable coefficient problems, and therefore for heterogeneous material properties.

The only simplifying assumption to make for use with finite volume discretization is to assume that thermal conductivity, while it may vary in space, is not temperature dependent:

$$k(x, T) \cong k(x) \quad (3.14)$$

<sup>7</sup> Element subscripts indicate which cell or inter-cell boundary the element belongs to.

<sup>8</sup> The constituent expressions are either the linear elements' moduli (for capacitive and resistive elements), or the expressions the elements impose on their bonds (for source elements).

Strictly speaking, this assumption is not necessary. However, retaining the temperature dependence of the thermal conductivity introduces nonlinearity for heat flux evaluation between cells, which would be preferable to avoid. Therefore, for the present analysis, temperature dependence of the material properties will be neglected.

Fundamentally, the finite volume discretization method relies on breaking the domain into finite cells, and approximating the surface integrals across the cell boundaries to evaluate the inter-cell fluxes. These approximate fluxes are then utilized to construct a system of rate balance ordinary differential equations, thus completing the system discretization. (Ref. [33]).

Flux functions, by definition, are functions of spatial derivatives of the unknown variable; Eq. (3.2) is one example. The approximate integral fluxes in finite volume method are integrals of these flux functions over the surface of the cell. In 1D, the “surface” between two nearby cells is simply the domain’s cross-sectional area, over which the flux does not vary. Therefore, the surface integral of  $u_v''$  at cell interface  $x_{i-1/2}$  is simply given by:

$$\oint_{S_{i-1/2}} dS u_v''(t, x) = A u_v''(t, x_{i-1/2}) \quad (3.15)$$

Using finite volume discretization, there are two ways to approximate the surface integrals at cell boundaries. The first way is to evaluate the derivatives under the integrals using interpolations based on cell-averaged (nodal) values, and then evaluate the integrals. The second way is to do it in reverse order: using piecewise shape functions within the cells, imposing constraints on the shape functions’ coefficients using the nodal values, and integrating the resulting shape functions. The second approach is the basis of spatial homogenization used in virtually all neutron transport codes and approaches, and will be utilized here as well.

Consider the following piecewise-constant shape functions for the flux function and the temperature:

$$T(t, x) \cong \bar{T}_i(t) \text{ for } x_{i-1/2} < x < x_{i+1/2} \quad (3.16)$$

$$u_v''(t, x) \cong \bar{u}_{v,i-1/2}''(t) \text{ for } x_{i-1} < x < x_i \quad (3.17)$$

in which:

$\bar{T}_i(t)$	Average temperature in finite cell $i$ at time $t$
$\bar{u}_{v,i-1/2}''(t)$	Average heat flux in the interval $x_{i-1} < x < x_i$ at time $t$

These shape functions are similar to the shape functions used by Stacey in his coarse mesh FEA derivations (Ref. [5]). They are illustrated in Figure 8 below. Notice, that the temperature shape function in Eq. (3.16) implies a similar shape function for  $u_v$  itself, because of Eq. (3.3).

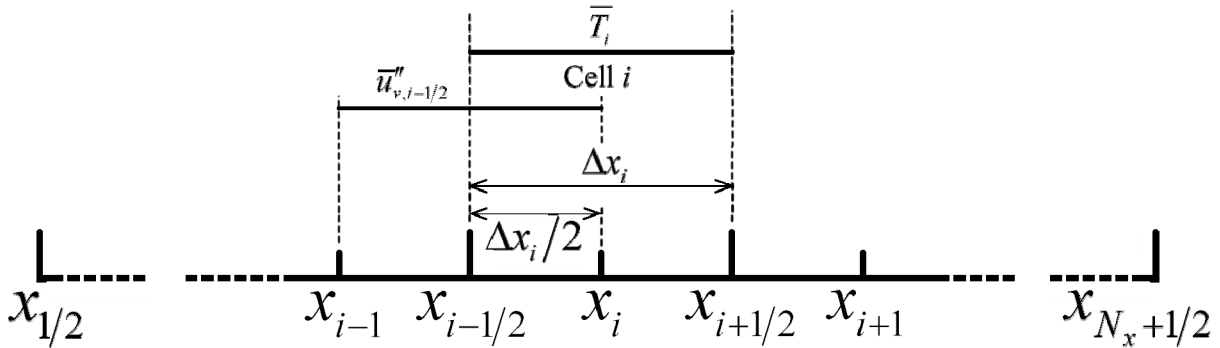


Figure 8. Discretized 1D Domain with Flat Thermal Shape Functions

To obtain a solution in terms of these functions, first Eq. (3.1) has to be adjusted. Integrating both sides of the equation over the finite cell volume defined by the interval  $x \in (x_{i-1/2}, x_{i+1/2})$ , using the notation from the lumped volume treatment above:

$$\dot{U}_i(t) = -A \int_{x_{i-1/2}}^{x_{i+1/2}} dx \frac{\partial}{\partial x} u_v''(t, x) + \dot{U}_{ex,i}(t) \quad (3.18)$$

As stated above, the cell average temperature  $\bar{T}_i$  can be related to cell total energy  $U_i$ . To do so, integrating Eq. (3.3) over the finite cell volume defined by the interval in Eq. (3.18) and rearranging:

$$\bar{T}_i(t) = \frac{1}{\Delta V_i \rho c_p} U_i(t) \quad (3.19)$$

The total cell external source rate  $\dot{U}_{ex,i}(t)$  is given by Eq. (3.12). The initial conditions  $U_i^0$  are given by Eq. (3.13).

To simplify the integral on the right hand side of Eq. (3.18), the first fundamental theorem of calculus can be used. (Ref. [34]). In 2D or 3D, the Gauss-Ostrogradsky theorem<sup>9</sup> would be used instead. (Ref. [32]). Writing out the integral from Eq. (3.18):

$$-A \int_{x_{i-1/2}}^{x_{i+1/2}} dx \frac{\partial}{\partial x} u_v''(t, x) = A \left[ u_v''(t, x_{i-1/2}) - u_v''(t, x_{i+1/2}) \right] \quad (3.20)$$

Equation (3.20) can simply be thought of as the difference in inflow and outflow of heat into the finite volume  $i$ . Equation (3.17) illustrates, that a flat flux function is assumed between nearby cell centers. Implementing such flat flux function in Eq. (3.20), and plugging the resultant equation into Eq. (3.18):

$$\dot{U}_i(t) = A \bar{u}_{v,i-1/2}''(t) - A \bar{u}_{v,i+1/2}''(t) + \dot{U}_{ex,i}(t) \quad (3.21)$$

To evaluate the heat flux integrals at the cell edges, the flat flux functions have to be related to the temperature averages. To do so, the shape functions in Eqs. (3.16) and (3.17) are again utilized. As was stated above, temperature dependence of the material properties is neglected. Rearranging Eq. (3.2):

$$\frac{1}{k(x)} u_v''(t, x) = -\frac{\partial}{\partial x} T(t, x) \quad (3.22)$$

$k(x)$  is thermal conductivity, so the quantity  $1/k(x)$  can be interpreted as thermal resistivity. In this text, the symbol  $o(x)$  will be used for it; plugging it into Eq. (3.22) yields:

$$o(x) u_v''(t, x) = -\frac{\partial}{\partial x} T(t, x) \quad (3.23)$$

Integrating Eq. (3.23) over the finite volume defined by the interval  $x \in (x_{i-1}, x_i)$ :

$$A \int_{x_{i-1}}^{x_i} dx o(x) u_v''(t, x) = -A \int_{x_{i-1}}^{x_i} dx \frac{\partial}{\partial x} T(t, x) \quad (3.24)$$

Recognizing that the flat shape function of  $u_v''(t, x)$  in this interval (Eq. (3.17)) results in  $u_v''(t, x)$  being constant under the integral allows us to simplify the left side of the equation:

$$A \int_{x_{i-1}}^{x_i} dx o(x) u_v''(t, x) = A \bar{u}_{v,i-1/2}''(t) \int_{x_{i-1}}^{x_i} dx o(x) \quad (3.25)$$

Denoting the average thermal resistivity in the interval  $x \in (x_{i-1}, x_i)$   $\bar{o}_{i-1/2}$ :

$$\bar{o}_{i-1/2} = \left[ \frac{\Delta x_{i-1}}{2} + \frac{\Delta x_i}{2} \right]^{-1} \int_{x_{i-1}}^{x_i} dx o(x) \quad (3.26)$$

<sup>9</sup> Often called the divergence theorem.



Notice, that instead of averaging the thermal conductivity over the interval of interest, the thermal resistivity was averaged. This is to be expected: it is a well-known fact in circuit theory that resistances, not conductances, add in series to yield the overall resistance, which can then be inverted to yield the overall conductance. (Ref. [21]). Equation (3.26) is a continuous form of this fact.

Using Eq. (3.26) allows us to further simplify Eq. (3.25):

$$A \int_{x_{i-1}}^{x_i} dx o(x) u_v''(t, x) = A \bar{u}_{v,i-1/2}''(t) \bar{o}_{i-1/2} \left[ \frac{\Delta x_{i-1}}{2} + \frac{\Delta x_i}{2} \right] \quad (3.27)$$

Simplifying the right hand side of Eq. (3.24):

$$-A \int_{x_{i-1}}^{x_i} dx \frac{\partial}{\partial x} T(t, x) = A [T(t, x_{i-1}) - T(t, x_i)] \quad (3.28)$$

Implementing the flat shape function of  $T(t, x)$  (Eq. (3.16)) simplifies the equation further:

$$-A \int_{x_{i-1}}^{x_i} dx \frac{\partial}{\partial x} T(t, x) = A [\bar{T}_{i-1}(t) - \bar{T}_i(t)] \quad (3.29)$$

Combining Eqs. (3.27) and (3.29) to construct the flux surface integral estimate as a function of temperatures:

$$A \bar{u}_{v,i-1/2}''(t) = A \frac{\bar{T}_{i-1}(t) - \bar{T}_i(t)}{\left[ \frac{\Delta x_{i-1}}{2} + \frac{\Delta x_i}{2} \right] \bar{o}_{i-1/2}} \quad (3.30)$$

For boundary cells, the flux shape functions on the boundary side only occupy the half of the cell adjacent to the boundary, that is:

$$u_v''(t, x) \cong \bar{u}_{v,1/2}''(t) \text{ for } x_{1/2} < x < x_1 \quad (3.31)$$

$$u_v''(t, x) \cong \bar{u}_{v,N_x+1/2}''(t) \text{ for } x_{N_x} < x < x_{N_x+1/2} \quad (3.32)$$

Running the analysis similar to Eqs. (3.24)-(3.30) for these boundary flux integrals, and recognizing that the temperatures at the boundaries are given as exact functions  $T_{left}(t)$  and  $T_{right}(t)$ , yields the following flux surface integral estimates as function of temperatures:

$$A \bar{u}_{v,1/2}''(t) = A \frac{T_{left}(t) - \bar{T}_1(t)}{\left[ \frac{\Delta x_1}{2} \right] \bar{o}_{1/2}} \quad (3.33)$$

$$A \bar{u}_{v,N_x+1/2}''(t) = A \frac{\bar{T}_{N_x}(t) - T_{right}(t)}{\left[ \frac{\Delta x_{N_x}}{2} \right] \bar{o}_{N_x+1/2}} \quad (3.34)$$

Similarly to Eq. (3.26), the averaged thermal resistivities are given by:

$$\bar{o}_{1/2} = \left[ \frac{\Delta x_1}{2} \right]^{-1} \int_{x_{1/2}}^{x_1} dx o(x) \quad (3.35)$$

$$\bar{o}_{N_x+1/2} = \left[ \frac{\Delta x_{N_x}}{2} \right]^{-1} \int_{x_{N_x}}^{x_{N_x+1/2}} dx o(x) \quad (3.36)$$

The formal finite volume discretization with heterogeneous material properties is now complete and can be summarized. Cell total energy and temperature are related by Eq. (3.19). Equation (3.12) yields the cell total external heat rate. Equation (3.21) relates the rate of change of total stored energy in a cell to other rates, and the heat flux integrals are evaluated using Eq.

(3.30). The averaged thermal resistivity is given by Eq. (3.26). The boundary heat flux integrals and averaged thermal resistivities are given by Eqs. (3.33)-(3.36), respectively.

As in the lumped volume interpretation, temperatures are efforts, heat flows are flows and cell total thermal energies are displacements. With these assumptions, equations from the above summary can be compared to Table 4 (p. 34) and Table 7 above. Equation (3.19) is a standard capacitor equation and Eq. (3.12) is a flow imposed by a time-modulated source of flow. The addition of flows in Eq. (3.21) is imposed by a 0-junction, and the inter-cell flow imposed in Eq. (3.30) proportional to a difference in efforts is a combination of a resistive element and a 1-junction, which is connected to the 0-junctions that set the neighbor-cell temperatures (efforts). The boundary temperatures in Eqs. (3.33) and (3.34) are externally imposed efforts, imposed onto 0-junctions by time-modulated sources of effort. Equations (3.33) and (3.34) themselves are identical, in their form, to Eq. (3.30), which, as stated previously, has the form of a resistive element connected to a 1-junction. The initial conditions for the capacitors' displacements are defined in Eq. (3.13).

Combining all these interpretations, the bond graph representation of a 1D thermal diffusion problem using formally derived finite volume discretization can be constructed. The bond graph system itself happens to be identical to the one obtained for spatially uniform material properties problem using lumped volume representation, given in Figure 7 above. However, the element constituent expressions are now slightly different, resulting from the formal derivation which was necessary for the spatially heterogeneous thermal conductivity. The new constituent expressions are given in Table 9 below:

**Table 9. Heat Diffusion Bond Graph Constituent Expressions with Heterogeneous  $k$**

Element	Constituent Expressions
$TSE_{left}$	$T_{left}(t)$
$R_{left}$	$\frac{\Delta x_1 \bar{\sigma}_{1/2}}{2A}$
$C_i$	$\frac{1}{\Delta V_i \rho c_p}$
$R_{i-1/2}$	$\frac{(\Delta x_{i-1} + \Delta x_i) \bar{\sigma}_{i-1/2}}{2A}$
$TSF_i$	$\dot{U}_{ex,i}(t)$

It is notable, that with a uniform thermal conductivity  $k$ , the constituent expressions in Table 8 and Table 9 are the same, since the averaged thermal resistivities would all reduce to  $1/k$ . This indicates that the physical arguments that lead to the lumped volume formulation are equivalent to the formal mathematical arguments which resulted in the finite volume discretization, allowing for a spatially heterogeneous conductivity.

It is also interesting, that the integral rate balance Eq. (3.21), together with the heat flux in Eq. (3.30), is very similar to a more common finite difference discretization of Eq. (3.1). Such finite difference discretization, along with more complicated ones, are studied in detail in Ref. [35]. If an explicit forward Euler time stepping is used, and the material properties are assumed uniform and cell widths all equal, the local numerical stability requirement is given by:

$$\frac{k}{\rho c_p} \frac{\Delta t}{\Delta x^2} \leq \frac{1}{2} \quad (3.37)$$

in which:

---

$\Delta t$	Time step
------------	-----------

---

This stability requirement is called “local,” because it does not account for the boundary conditions, which may destabilize the problem. However, this is rarely the case in practical heat diffusion problems.

For finite volume discretization, numerical stability is less well defined than for finite difference discretization, because only the error in the integral properties of the solution can be recovered (and is of any interest, if a finite volume discretization is used in the first place). Reference [33] provides a rigorous mathematical study of the numerical stability and convergence properties of finite volume discretizations for hyperbolic (advection) problems; for parabolic (transient diffusive) problems, criteria similar to Eq. (3.37) can be constructed using von Neumann analysis and other techniques normally used in finite difference methods.

Rigorous proofs of convergence using finite volume methods are generally unavailable in multidimensional hyperbolic problems. (Ref. [33]). Even for the more simple parabolic problems, such analysis can be very complicated, and typically involves numerous assumptions of limited physical validity. For coupled problems, such analysis is generally impossible.

Finite difference-type convergence criteria can often be used to provide an order-of-magnitude estimate for the size of time step required for the stability of an explicit method. However, in general, we will be more interested in experimentally proving the consistency, stability and convergence of the method on a well-studied problem. Due to the currently limited body of theoretical numerical analysis tools available for nonlinear coupled problems, and the extreme complexity of such problems, this is how most practical multiphysics codes are currently tested. (Ref. [17]).

The fundamental difference between finite volume and finite difference methods is that the finite volume methods are integral-conservative, while finite difference methods are not. Furthermore, finite volume methods, similarly to how it was done in the study above, are prone to averaging properties, while finite difference methods essentially sample them out of the domain. In heat transfer either technique is usually sufficient. However, in the more sensitive neutron diffusion studies, integral rate conservation, at least within the model, is a necessary requirement.

The above similarity between finite volume and finite difference discretization methods is the reason for why methods which rely on homogenized (averaged) data are often called “finite difference methods” in neutron transport. They often happen to be finite volume methods instead.

The general mathematical techniques and bond graph representation developed in this section will now be used in neutron diffusion representation, and then for coupling the two types of physics.

### **3.2. Neutron Diffusion via Bond Graphs**

Neutron transport of any kind, including neutron diffusion, has never been represented with bond graphs. In this section, the first such representation is proposed, in close parallel with heat diffusion in section 3.1. The equations treated are very similar to the heat diffusion equations, so the accompanying discussion from the formal treatment of the finite volume discretization in section 3.1 is applicable here as well.

As discussed in section 2.1 (p. 19), 1D one-group neutron diffusion is modeled by the following equations:

$$\frac{\partial}{\partial t} n(t, x) = -\frac{\partial}{\partial x} J(t, x) + \nu \Sigma_f(x) \phi(t, x) - \Sigma_a(x) \phi(t, x) + s_{ex}(t, x) \quad (3.38)$$

$$J(t, x) = -D(x) \frac{\partial}{\partial x} \phi(t, x) \quad (3.39)$$

$$\phi(t, x) = V_n n(t, x) \quad (3.40)$$

These equations are very similar to Eqs. (3.1)-(3.3), except for the absorption and fission generation terms, which do not exist in heat diffusion.

As with heat diffusion, some type of boundary conditions are required. Again, Dirichlet boundary conditions,  $\phi_{left}(t)$  and  $\phi_{right}(t)$  will be used, because zero Dirichlet boundary conditions are often used to approximate vacuum boundaries. An extrapolation-distance boundary condition would again be of Dirichlet type, but with modified geometry. (Ref. [3]). Initial conditions are assumed specified by the function  $n^0(x)$ , which can be easily derived from an initial flux function  $\phi^0(x)$  using Eq. (3.40).

As with thermal diffusion, consider the following piecewise-constant shape functions for the scalar flux and scalar net current density:

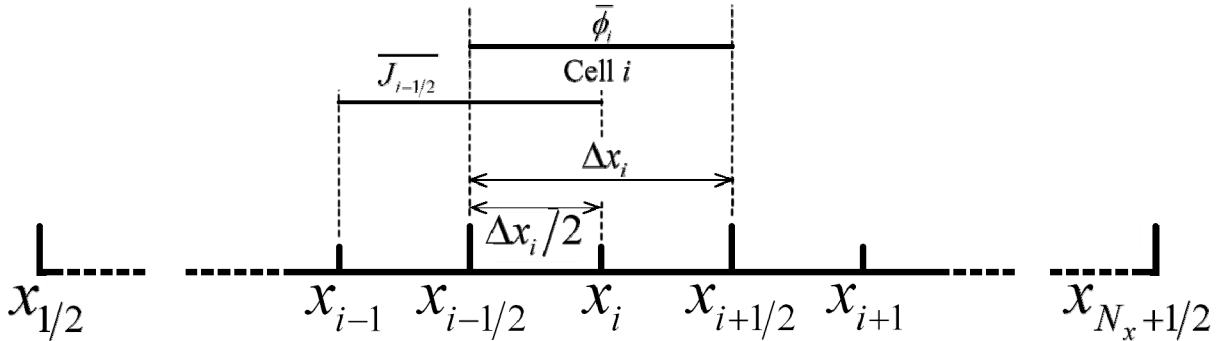
$$\phi(t, x) \cong \bar{\phi}_i(t) \text{ for } x_{i-1/2} < x < x_{i+1/2} \quad (3.41)$$

$$J(t, x) \cong \bar{J}_{i-1/2}(t) \text{ for } x_{i-1} < x < x_i \quad (3.42)$$

in which:

$\bar{\phi}_i(t)$	Average scalar flux in finite cell $i$ at time $t$
$\bar{J}_{i-1/2}(t)$	Average net current density in the interval $x_{i-1} < x < x_i$ at time $t$

These shape functions are identical to the shape functions used by Stacey in his coarse mesh FEA derivations (Ref. [5]). They are illustrated in Figure 9 below. Similar arguments to temperature and thermal energy density shape function hold, and so the flux shape function in Eq. (3.41) implies a similar shape function for  $n$ .



**Figure 9. Discretized 1D Domain with Flat Neutron Shape Functions**

Strictly, 1D analysis is only applicable to an infinite slab reactor, because otherwise neutrons will leak through the other two dimensions, which 1D analysis cannot readily account for. However, to stay consistent with the fixed cross-sectional area type treatment from section 3.1, the analysis here will also assume a fixed cross-sectional area  $A$ . A dimensionless number like  $A = 1$  can be used, thus making the treatment equivalent to an infinite 1D slab reactor.

To relate the cell average flux  $\bar{\phi}_i$  to the total number of neutrons in a cell, we integrate Eq. (3.40) over the finite cell volume defined by the interval  $x \in (x_{i-1/2}, x_{i+1/2})$ :

$$\bar{\phi}_i(t) = \frac{V_n}{\Delta V_i} N_i(t) \quad (3.43)$$

in which:

$N_i(t)$	Rate of change of the total number of neutrons in finite cell $i$ at time $t$
----------	---

Integrating both sides of Eq. (3.38) over the finite cell volume defined by the interval  $x \in (x_{i-1/2}, x_{i+1/2})$ :

$$\dot{N}_i(t) = -A \int_{x_{i-1/2}}^{x_{i+1/2}} dx \frac{\partial}{\partial x} J(t, x) + A \int_{x_{i-1/2}}^{x_{i+1/2}} dx [\nu \Sigma_f(x) - \Sigma_a(x)] \phi(t, x) + S_{ex,i}(t) \quad (3.44)$$

in which:

$\dot{N}_i(t)$	Rate of change of the number of neutrons in finite cell $i$ at time $t$
$S_{ex,i}(t)$	Neutron generation rate in finite cell $i$ from an external neutron source

The external neutron source rate  $S_{ex,i}(t)$  is given by:

$$S_{ex,i}(t) = A \int_{x_{i-1/2}}^{x_{i+1/2}} dx s_{ex}(t, x) \quad (3.45)$$

The initial conditions are similarly given by:

$$N_i^0 = A \int_{x_{i-1/2}}^{x_{i+1/2}} dx n^0(x) \quad (3.46)$$

To simplify the first (streaming) integral on the right hand side of Eq. (3.44), using the first fundamental theorem of calculus:

$$-A \int_{x_{i-1/2}}^{x_{i+1/2}} dx \frac{\partial}{\partial x} J(t, x) = A [\overline{J_{i-1/2}}(t) - \overline{J_{i+1/2}}(t)] \quad (3.47)$$

Simplifying the second (reaction) integral on the right hand side of Eq. (3.44):

$$A \int_{x_{i-1/2}}^{x_{i+1/2}} dx [\nu \Sigma_f(x) - \Sigma_a(x)] \phi(t, x) = \Delta V_i [\nu \overline{\Sigma_{f,i}} - \overline{\Sigma_{a,i}}] \bar{\phi}_i(t) \quad (3.48)$$

in which:

$\overline{\Sigma_{f,i}}$	Homogenized one-group fission cross-section in finite cell $i$
$\overline{\Sigma_{a,i}}$	Homogenized one-group absorption cross-section in finite cell $i$

The homogenized cross-sections are given by:

$$\overline{\Sigma_{f,i}} = \frac{A}{\Delta V_i} \int_{x_{i-1/2}}^{x_{i+1/2}} dx \Sigma_f(x) \quad (3.49)$$

$$\overline{\Sigma_{a,i}} = \frac{A}{\Delta V_i} \int_{x_{i-1/2}}^{x_{i+1/2}} dx \Sigma_a(x) \quad (3.50)$$

Equation (3.48) can be thought of as the net rate of neutron production due to absorption. Plugging it back into Eq. (3.44) simplifies the rate balance equation further:

$$\dot{N}_i(t) = A \overline{J_{i-1/2}}(t) - A \overline{J_{i+1/2}}(t) + \Delta V_i [\nu \overline{\Sigma_{f,i}} - \overline{\Sigma_{a,i}}] \bar{\phi}_i(t) + S_{ex,i}(t) \quad (3.51)$$

To relate the flat current density functions to the flux averages, rearranging Eq. (3.39):

$$\frac{1}{D(x)} J(t, x) = -\frac{\partial}{\partial x} \phi(t, x) \quad (3.52)$$

Similarly to comparing thermal conductivity and its reciprocal, thermal resistivity, the reciprocal of  $D(x)$  also needs a name. In this text, the name *confusion coefficient*, and the symbol  $Co$  are proposed:

$$Co(x) \equiv \frac{1}{D(x)} \quad (3.53)$$

Plugging Eq. (3.53) into Eq. (3.52) and integrating it over the finite volume defined by the interval  $x \in (x_{i-1}, x_i)$ :

$$A \int_{x_{i-1}}^{x_i} dx Co(x) J(t, x) = -A \int_{x_{i-1}}^{x_i} dx \frac{\partial}{\partial x} \phi(t, x) \quad (3.54)$$

Using the flat shape function of  $J(t, x)$  from Eq. (3.42) in this interval to simplify the left hand side of the equation:

$$A \int_{x_{i-1}}^{x_i} dx Co(x) J(t, x) = \overline{AJ}_{i-1/2}(t) \int_{x_{i-1}}^{x_i} dx Co(x) \quad (3.55)$$

Denoting the average confusion coefficient in the interval  $x \in (x_{i-1}, x_i)$   $\overline{Co}_{i-1/2}$ :

$$\overline{Co}_{i-1/2} = \left[ \frac{\Delta x_{i-1}}{2} + \frac{\Delta x_i}{2} \right]^{-1} \int_{x_{i-1}}^{x_i} dx Co(x) \quad (3.56)$$

Plugging Eq. (3.56) into Eq. (3.55):

$$A \int_{x_{i-1}}^{x_i} dx Co(x) J(t, x) = \overline{AJ}_{i-1/2}(t) \left[ \frac{\Delta x_{i-1}}{2} + \frac{\Delta x_i}{2} \right] \overline{Co}_{i-1/2} \quad (3.57)$$

Using the first fundamental theorem of calculus to simplify the right hand side of Eq. (3.54):

$$-A \int_{x_{i-1}}^{x_i} dx \frac{\partial}{\partial x} \phi(t, x) = A [\phi(t, x_{i-1}) - \phi(t, x_i)] \quad (3.58)$$

Implementing the flat flux shape functions from Eq. (3.41):

$$-A \int_{x_{i-1}}^{x_i} dx \frac{\partial}{\partial x} \phi(t, x) = A [\bar{\phi}_{i-1}(t) - \bar{\phi}_i(t)] \quad (3.59)$$

Combining Eqs. (3.57) and (3.59) to construct the current surface integral as a function of cell fluxes:

$$\overline{AJ}_{i-1/2}(t) = A \frac{\bar{\phi}_{i-1}(t) - \bar{\phi}_i(t)}{\left[ \frac{\Delta x_{i-1}}{2} + \frac{\Delta x_i}{2} \right] \overline{Co}_{i-1/2}} \quad (3.60)$$

Dirichlet boundary conditions are used, therefore the fluxes at the boundaries are given as functions of time. Therefore similar analysis will work to compute the current surface integrals at the boundaries. Applying similar analysis for boundary cells yields the following averaged confusion coefficients:

$$\overline{Co}_{1/2} = \left[ \frac{\Delta x_1}{2} \right]^{-1} \int_{x_{1/2}}^{x_1} dx Co(x) \quad (3.61)$$

$$\overline{Co}_{N_x+1/2} = \left[ \frac{\Delta x_{N_x}}{2} \right]^{-1} \int_{x_{N_x}}^{x_{N_x+1/2}} dx Co(x) \quad (3.62)$$

The current surface integrals at the boundaries are then given by:

$$\overline{AJ}_{1/2}(t) = A \frac{\bar{\phi}_{left}(t) - \bar{\phi}_1(t)}{\left[ \frac{\Delta x_1}{2} \right] \overline{Co}_{1/2}} \quad (3.63)$$

$$\overline{AJ}_{N_x+1/2}(t) = A \frac{\bar{\phi}_{N_x}(t) - \bar{\phi}_{right}(t)}{\left[ \frac{\Delta x_{N_x}}{2} \right] \overline{Co}_{N_x+1/2}} \quad (3.64)$$

The finite volume discretization of the neutron diffusion equations is now complete and can be summarized. Cell total number of neutrons and flux are related by Eq. (3.43). The

external cell total neutron source rate is given by Eq. (3.45). Equation (3.51) relates the rate of change of the total number of neutrons in a cell to other rates. The net rate of neutron production due to absorption is given by Eq. (3.48). The current surface integrals are evaluated using Eq. (3.60), and the homogenized confusion coefficient is given by Eq. (3.56). The boundary current surface integrals are evaluated using Eqs. (3.63) and (3.64), and the boundary homogenized confusion coefficients are given by Eqs. (3.61) and (3.62).

As stated above, neutron diffusion has never been represented with bond graphs. However, by comparing the above listed equations to the similar equations in heat diffusion, in section 3.1, one can clearly see an analogy between the cell temperatures and the cell scalar fluxes, as well as cell total thermal energies and cell total numbers of neutrons. It is proposed to use this analogy from now on in the bond graph representation of neutron diffusion. The proposal is summarized in Table 10.

**Table 10. Bond Graph Variables for Finite Volume-Discretized Neutron Diffusion**

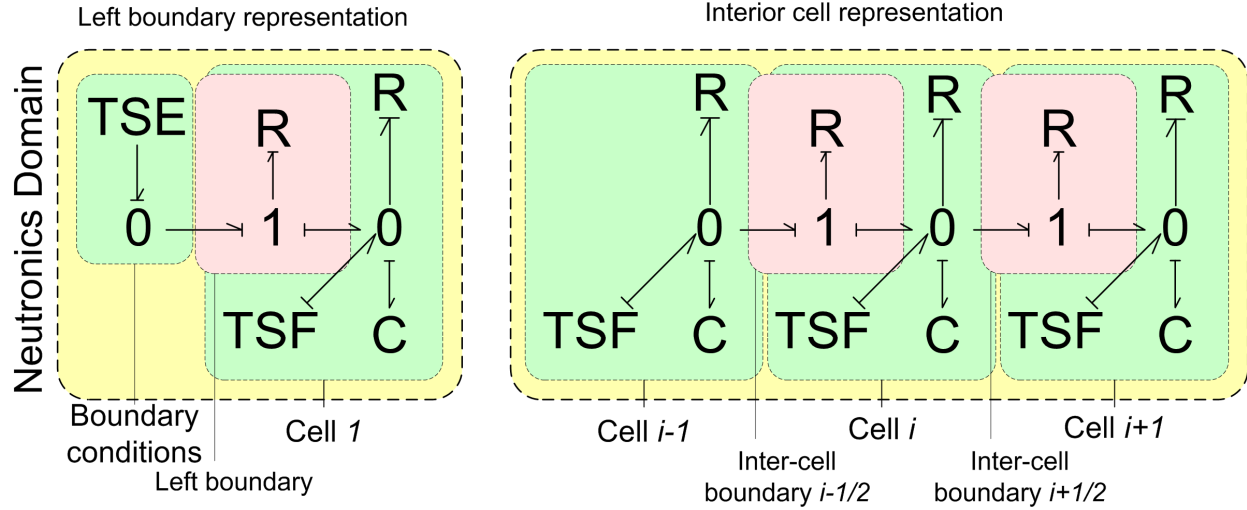
<b>Bond graph variable</b>	<b>Physical meaning</b>
Effort	Scalar flux
Flow	Neutron current <sup>10</sup>
Displacement	Number of neutrons

With these variable representations, equations from the above summary can be compared to Table 4 (p. 34) and Table 7 above. Equation (3.43) is a capacitor equation. Equation (3.45) is a flow imposed by a time-modulated source of flow. The rate balance in Eq. (3.51) is an addition of flows imposed by a 0-junction. The net rate of neutron production, Eq. (3.48), which does not have an analogy in the heat diffusion equations, yields a flow proportional to an effort, which is the function of a linear resistor. The inter-cell flow from Eq. (3.60), which is proportional to a difference in efforts, is a combination of a resistive element and a 1-junction, which is connected to the 0-junctions in the neighboring cells that set the neighboring cells' scalar fluxes. The boundary scalar fluxes in Eqs. (3.63) and (3.64) are externally imposed efforts, imposed onto 0-junctions by time-modulated sources of effort. Equations (3.63) and (3.64) are, similarly to Eq. (3.60), resistive elements connected to a 1-junction.

Combining all these interpretations, the bond graph representation of a 1D one-group finite-volume discretized neutron diffusion problem can be constructed. It is given in Figure 10 below.

---

<sup>10</sup> The neutron current has the dimensions of, and is in many cases equivalent to, a reaction rate.



**Figure 10. Neutron Diffusion Bond Graph Representation**

The constituent expressions for the elements in this representation are given in Table 11 below.

**Table 11. Neutron Diffusion Bond Graph Constituent Expressions**

Element	Constituent Expressions
$TSE_{left}$	$\phi_{left}(t)$
$R_{left}$	$\frac{\Delta x_1 \overline{C_{0_{1/2}}}}{2A}$
$C_i$	$\frac{V_n}{\Delta V_i}$
$R_{i-1/2}$	$\frac{(\Delta x_{i-1} + \Delta x_i) \overline{C_{i-1/2}}}{2A}$
$R_i$	$\frac{1}{\Delta V_i (\overline{\Sigma_{a,i}} - \nu \overline{\Sigma_{f,i}})}$
$TSF_i$	$S_{ex,i}(t)$

Applying the finite difference stability analysis for forward Euler stepping, similar to the one in section 3.1, and neglecting the net production term, the local stability criterion comes out to:

$$DV_n \frac{\Delta t}{\Delta x^2} \leq \frac{1}{2} \quad (3.65)$$

In a coupled problem, both the heat and neutron diffusion phenomena would take place simultaneously. The local stability criteria in Eqs. (3.37) and (3.65) would both have to be satisfied for local stability. This would not necessarily guarantee convergence, but gives a good order of magnitude estimate for the time step  $\Delta t$ . Generally, in most engineering materials, the requirement in Eq. (3.65) is a more stringent one, due to the large value of  $V_n$ . Again, it must be stressed, that these are only order of magnitude estimates for stability of a coupled problem's time integrator, and that the net production term in the neutron diffusion equation significantly complicates the stability analysis.



Now, that the heat and neutron diffusion problems have been represented with bond graphs, a coupled problem can be represented.

### 3.3. Coupled Diffusion via Bond Graphs

As discussed in section 2.3 (p. 27), the coupled 1D heat and one-group neutron diffusion equations are:

$$\frac{\partial}{\partial t} n(t, x) = -\frac{\partial}{\partial x} J(t, x) + \nu \Sigma_f(x, T) \phi(t, x) - \Sigma_a(x, T) \phi(t, x) + s_{ex}(t, x) \quad (3.66)$$

$$J(t, x) = -D(x, T) \frac{\partial}{\partial x} \phi(t, x) \quad (3.67)$$

$$\phi(t, x) = V_n n(t, x) \quad (3.68)$$

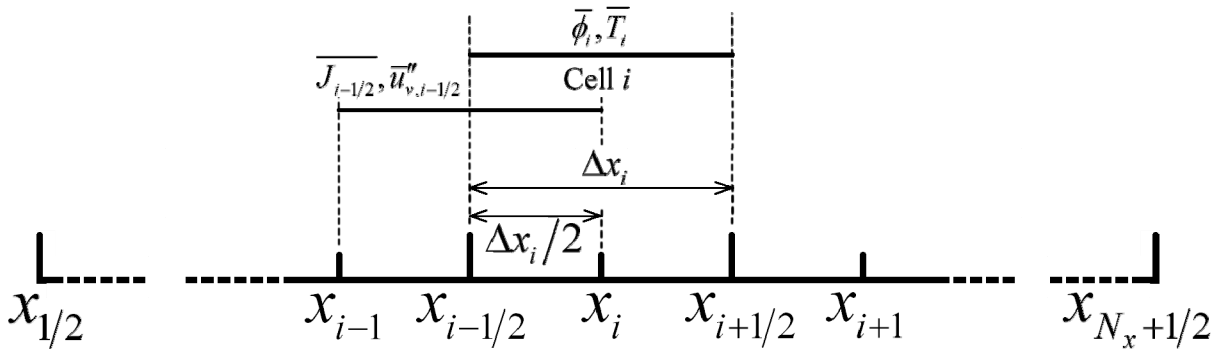
$$\frac{\partial}{\partial t} u_v(t, x) = -\frac{\partial}{\partial x} u_v''(t, x) + w \Sigma_f(x, T) \phi(t, x) + \dot{u}_{v,ex}(t, x) \quad (3.69)$$

$$u_v''(t, x) = -k(x, T) \frac{\partial}{\partial x} T(t, x) \quad (3.70)$$

$$T(u_v) = \frac{1}{\rho c_p} u_v \quad (3.71)$$

The boundary conditions are still assumed to be Dirichlet for both temperature and scalar flux; therefore, functions  $\phi_{left}(t)$ ,  $\phi_{right}(t)$ ,  $T_{left}(t)$  and  $T_{right}(t)$  are assumed given. The initial conditions are specified by functions  $u_v^0(x)$  and  $n^0(x)$ , which can be derived from initial flux and temperature distributions using Eqs. (3.68) and (3.71).

The same shape functions for temperature, scalar flux, heat flux and current density are used in the coupled model discretization as in the individual model discretization. Together, they are shown in Figure 11 below:



**Figure 11. Discretized 1D Domain with Flat Coupled Shape Functions**

The coupling does not greatly affect the discretization, but does modify the homogenized properties in the neutron diffusion equation and fission source is added in the heat diffusion equation. First, the new homogenized properties are treated below. They are now functions of temperature:

$$\overline{\Sigma}_{f,i}(T) = \frac{A}{\Delta V_i} \int_{x_{i-1/2}}^{x_{i+1/2}} dx \Sigma_f(x, T(t, x)) \quad (3.72)$$

$$\overline{\Sigma}_{a,i}(T) = \frac{A}{\Delta V_i} \int_{x_{i-1/2}}^{x_{i+1/2}} dx \Sigma_a(x, T(t, x)) \quad (3.73)$$

$$\overline{Co}_{i-1/2}(T) = \left[ \frac{\Delta x_{i-1}}{2} + \frac{\Delta x_i}{2} \right]^{-1} \left[ \int_{x_{i-1}}^{x_{i+1/2}} dx Co(x, T(t, x)) + \int_{x_{i-1/2}}^{x_i} dx Co(x, T(t, x)) \right] \quad (3.74)$$

$$\overline{Co}_{1/2}(T) = \left[ \frac{\Delta x_1}{2} \right]^{-1} \left[ \int_{x_{1/2}}^{x_1} dx Co(x, T(t, x)) \right] \quad (3.75)$$

$$\overline{Co}_{N_x+1/2}(T) = \left[ \frac{\Delta x_{N_x}}{2} \right]^{-1} \left[ \int_{x_{N_x}}^{x_{N_x+1/2}} dx Co(x, T(t, x)) \right] \quad (3.76)$$

A flat temperature shape function was assumed in section 3.1, which means that under each of the integrals in Eqs. (3.72)-(3.76), temperature is constant. Therefore, the expressions can be rewritten as functions of cell-averaged temperatures:

$$\overline{\Sigma}_{f,i}(\overline{T}_i) = \frac{A}{\Delta V_i} \int_{x_{i-1/2}}^{x_{i+1/2}} dx \Sigma_f(x, \overline{T}_i) \quad (3.77)$$

$$\overline{\Sigma}_{a,i}(\overline{T}_i) = \frac{A}{\Delta V_i} \int_{x_{i-1/2}}^{x_{i+1/2}} dx \Sigma_a(x, \overline{T}_i) \quad (3.78)$$

$$\overline{Co}_{i-1/2}(\overline{T}_{i-1}, \overline{T}_i) = \left[ \frac{\Delta x_{i-1}}{2} + \frac{\Delta x_i}{2} \right]^{-1} \left[ \int_{x_{i-1}}^{x_{i-1/2}} dx Co(x, \overline{T}_{i-1}) + \int_{x_{i-1/2}}^{x_i} dx Co(x, \overline{T}_i) \right] \quad (3.79)$$

$$\overline{Co}_{1/2}(\overline{T}_1) = \left[ \frac{\Delta x_1}{2} \right]^{-1} \left[ \int_{x_{1/2}}^{x_1} dx Co(x, \overline{T}_1) \right] \quad (3.80)$$

$$\overline{Co}_{N_x+1/2}(\overline{T}_{N_x}) = \left[ \frac{\Delta x_{N_x}}{2} \right]^{-1} \left[ \int_{x_{N_x}}^{x_{N_x+1/2}} dx Co(x, \overline{T}_{N_x}) \right] \quad (3.81)$$

Equations (3.77) and (3.78) yield a convenient expression for the new net rate of neutron production due to absorption in cell  $i$ :

$$\begin{aligned} A \int_{x_{i-1/2}}^{x_{i+1/2}} dx \left[ \nu \Sigma_f(x, T(t, x)) - \Sigma_a(x, T(t, x)) \right] \phi(t, x) = \\ = \Delta V_i \left[ \nu \overline{\Sigma}_{f,i}(\overline{T}_i(t)) - \overline{\Sigma}_{a,i}(\overline{T}_i(t)) \right] \overline{\phi}_i(t) \end{aligned} \quad (3.82)$$

With these modified homogenized properties, the rest of the neutron diffusion discretization from section 3.2 stays the same. Now the integrated heat source can be treated. Integrating it over the cell volume defined by the interval  $x \in (x_{i-1/2}, x_{i+1/2})$  yields the temperature-dependent fission heat source:

$$A \int_{x_{i-1/2}}^{x_{i+1/2}} dx w \Sigma_f(x, T(t, x)) \phi(t, x) = \Delta V_i w \overline{\Sigma}_{f,i}(\overline{T}_i(t)) \overline{\phi}_i(t) \quad (3.83)$$

Notice, that all averaged temperatures and homogenized fluxes are dependent on time.

Together, Eqs. (3.77)-(3.83), can be implemented into the original single physics discretizations (sections 3.1 and 3.2). The discretized neutron diffusion is now given by Eqs. (3.84)-(3.88):

$$\dot{N}_i(t) = A \overline{J}_{i-1/2}(t) - A \overline{J}_{i+1/2}(t) + \Delta V_i \left[ \nu \overline{\Sigma}_{f,i}(\overline{T}_i(t)) - \overline{\Sigma}_{a,i}(\overline{T}_i(t)) \right] \overline{\phi}_i(t) + S_{ex,i}(t) \quad (3.84)$$

$$\overline{\phi}_i(t) = \frac{V_n}{\Delta V_i} N_i(t) \quad (3.85)$$

$$\overline{J}_{i-1/2}(t) = A \frac{\overline{\phi}_{i-1}(t) - \overline{\phi}_i(t)}{\left[ \frac{\Delta x_{i-1}}{2} + \frac{\Delta x_i}{2} \right] \overline{Co}_{i-1/2}(\overline{T}_{i-1}, \overline{T}_i)} \quad (3.86)$$

$$A\overline{J}_{1/2}(t) = A \frac{\overline{\phi}_{left}(t) - \overline{\phi}_1(t)}{\left[ \frac{\Delta x_1}{2} \right] \overline{Co}_{1/2}(\overline{T}_1(t))} \quad (3.87)$$

$$A\overline{J}_{N_x+1/2}(t) = A \frac{\overline{\phi}_{N_x}(t) - \overline{\phi}_{right}(t)}{\left[ \frac{\Delta x_{N_x}}{2} \right] \overline{Co}_{N_x+1/2}(\overline{T}_{N_x})} \quad (3.88)$$

The discretized heat diffusion is now given by Eqs. (3.89)-(3.93):

$$\dot{U}_i(t) = A\overline{u}_{v,i-1/2}''(t) - A\overline{u}_{v,i+1/2}''(t) + \Delta V_i w \overline{\Sigma}_{f,i}(\overline{T}_i(t)) \overline{\phi}_i(t) + \dot{U}_{ex,i}(t) \quad (3.89)$$

$$\overline{T}_i(t) = \frac{1}{\Delta V_i \rho c_p} U_i(t) \quad (3.90)$$

$$A\overline{u}_{v,i-1/2}''(t) = A \frac{\overline{T}_{i-1}(t) - \overline{T}_i(t)}{\left[ \frac{\Delta x_{i-1}}{2} + \frac{\Delta x_i}{2} \right] \overline{\sigma}_{i-1/2}} \quad (3.91)$$

$$A\overline{u}_{v,1/2}''(t) = A \frac{\overline{T}_{left}(t) - \overline{T}_1(t)}{\left[ \frac{\Delta x_1}{2} \right] \overline{\sigma}_{1/2}} \quad (3.92)$$

$$A\overline{u}_{v,N_x+1/2}''(t) = A \frac{\overline{T}_{N_x}(t) - \overline{T}_{right}(t)}{\left[ \frac{\Delta x_{N_x}}{2} \right] \overline{\sigma}_{N_x+1/2}} \quad (3.93)$$

The finite volume discretization of the coupled diffusion equations is now complete and can be summarized. Total number of neutrons in a cell can be related to the scalar flux by Eq. (3.85). Total energy in a cell and temperature are related by Eq. (3.90). The external neutron and heat source rates remained unaffected by coupling, and are still given by Eqs. (3.45) and (3.12), respectively. The rate of change of the total number of neutrons in a cell is related to other rates by Eq. (3.84). Equation (3.89) relates the rate of change of total stored energy in a cell to other rates. The net rate of neutron production due to absorption is given by Eq. (3.82), and the net rate of heat generation due to fission is given by Eq. (3.83). The corresponding homogenized properties are evaluated using Eqs. (3.77) and (3.78). The neutron current surface integrals between cells are evaluated using Eq. (3.86), and their boundary versions are evaluated using Eqs. (3.87) and (3.88). The corresponding homogenized confusion coefficients are evaluated using (3.79)-(3.81), respectively. The heat flux surface integrals between cells are given by Eq. (3.91), and the boundary heat fluxes by Eqs. (3.92) and (3.93). The corresponding averaged quantities are unchanged from section 3.1, and are given by Eqs. (3.26), (3.35) and (3.36), respectively.

In many of these equations, the specific values of the coefficients changed, due to the introduced temperature dependence. However, the equations' overall forms did not change. This means that the bond graph representation for these equations stays almost the same, although the constituent expressions for the elements may be different. The difference in the bond graph representation comes from the fact that, for example, the flow of neutrons between two cells is now no longer only proportional to the difference in efforts (scalar fluxes), which made it a combination of a linear resistor and a 1-junction. This quantity is still proportional to the difference in efforts, but the constant of proportionality is now a function of two temperatures – two efforts. To treat such dependency, a new element is needed, called modulated resistor, or

MR; in this case, a modulated linear resistor<sup>11</sup> can be used. This element, in its general form, is introduced in Table 12 below.

**Table 12. Modulated and 2-Port Resistors**

Element	Element name	Type of element	Constituent equation(s)
	Modulated Resistor	Modulated damping	$f_i = G(e_i, e_j, e_k)$
	2-port Nonconservative Coupler	Multiport damping	$f_i = R2_1(e_i, e_j)$ $f_j = R2_2(e_i, e_j)$

In this case, the modulated resistor element is in a modulated coefficient form. In this form, the coefficient is the modulated proportionality constant between the effort and the flow of the resistor. The constituent equation therefore is:

$$f_i = \frac{1}{R(e_j, e_k)} e_i \quad (3.94)$$

As described in section 2.4 (p. 30), a bond generally has two associated bond variables – an effort and a flow. However, in this case, while the MR element is modulated by two efforts, it does not impose any flows back on the modulating bonds. Therefore, it can be said, that while these bonds carry information, they do not carry a conserved quantity – similarly to signal bonds in operational block diagrams used a lot in control theory. (Ref. [20]). This introduces a new bond type into the bond graph representation (mentioned in section 2.4) – the signal bond, which carries a single bond variable from a junction. Such junction must impose the causality on the signal bond: a 1-junction makes the bond deliver a flow from the junction, and a 0-junction makes the bond deliver an effort from the junction. For the purposes of SCAP, the junction’s causalities must be set through other bonds connected to it; that is, some element must impose a flow on a 1-junction or an effort on a 0-junction. In this case, the thermal 0-junction is ideal as the source junction of the modulating signal bonds, as all efforts it imposes correspond to the effort imposed on it by the thermal capacitor. (See Figure 7 for details). This effort, in turn, is the average temperature in the finite cell, according to Eq. (3.19). The other bond variable on a signal bond is zero, and does not contribute to any of the algebraic equations in the system.

The standard symbol for signal bonds is a line with an arrow symbol in the middle. Sometimes, a regular full arrow is used instead, similarly to the notation in operational block diagrams. Signal bonds are also sometimes referred to as activated bonds or active bonds, because they were originally used to model active electronic amplifiers. (Refs. [19,20]). The notation and underlying assumed equations for signal bonds are summarized in Table 13 below:

<sup>11</sup> A “modulated linear resistor” refers to the resistor’s equations’ quasilinear form. This means, that the flow imposed by the resistor is proportional to the effort supplied to the resistor through the resistor’s main port, denoted by bond  $i$  in the table. The proportionality constant is nonlinearly dependent on the modulating variables supplied through the signal ports, denoted by signal bonds  $j$  and  $k$  in the table.

**Table 13. Signal Bonds**

Augmented signal bond	Constituent equation(s)
$0 \xrightarrow{i} \text{MR}$	$e_i = e_0$ $f_i = 0$
$1 \xrightarrow{i} \text{MR}$	$f_i = f_1$ $e_i = 0$

For boundary resistors, the modulation is only performed by the boundary cells' temperatures, as seen from Eqs. (3.87) and (3.88).

Besides the newly-introduced modulated resistor, an additional element is required to model the coupling, called the R2 element. It is shown in Table 12, and is defined below. Both resistive elements in Table 12 are shown in the causality in which they will be used, but it must be noted, that other causalities are generally possible. For the purposes of SCAP, each port of the R2 element can be treated as an independent resistor.

The other additions from coupling are the temperature modulation of the net rate of neutron production (originally a resistor) and of the fission heat generation term. By the bond variable definitions specified in sections 3.1 and 3.2, the two terms are both flows, a neutron and a thermal flow, respectively. The other bond variable definitions state that temperature is a thermal effort and scalar flux is a neutron effort. Together, these neutron and heat production terms can be written in the following form:

$$\begin{bmatrix} f_i^N \\ f_i^T \end{bmatrix} = \begin{bmatrix} \Delta \nabla_i \left[ \overline{\Sigma_{a,i}}(e_i^T) - \nu \overline{\Sigma_{f,i}}(e_i^T) \right] e_i^N \\ \Delta \nabla_i w \overline{\Sigma_{f,i}}(e_i^T) e_i^N \end{bmatrix} \quad (3.95)$$

in which:

$f_i^N$	Neutron flow corresponding to net neutron removal rate due to absorption in finite cell $i$
$f_i^T$	Thermal flow corresponding to fission heat generation rate in finite cell $i$
$e_i^N$	Average scalar flux in finite cell $i$
$e_i^T$	Average temperature in finite cell $i$

By comparing Eq. (3.95) to the 2-port nonconservative coupler element in Table 12, it is clear that Eqs. (3.95) and the R2 constituent equations are in the same form. By convention, it will be assumed, that bond  $i$  is the bond coming from the neutron side of the coupler, and bond  $j$  is the bond coming from the thermal side of the coupler. With this assumption, the functions  $R2_1$  and  $R2_2$  are the first and second equations in Eqs. (3.95), the neutron flow and effort are  $f_i$  and  $e_i$ , and the thermal flow and effort are  $f_j$  and  $e_j$ .

The neutron and thermal flow rates set by the R2 coupler are parts of Eqs. (3.84) and (3.89), respectively. These rate balance equations are set by the 0-junctions, which also impose the cell average scalar flux and temperature as their efforts. Therefore, the R2 element will connect the neutron and thermal 0-junctions, with the bond orientation discussed above.

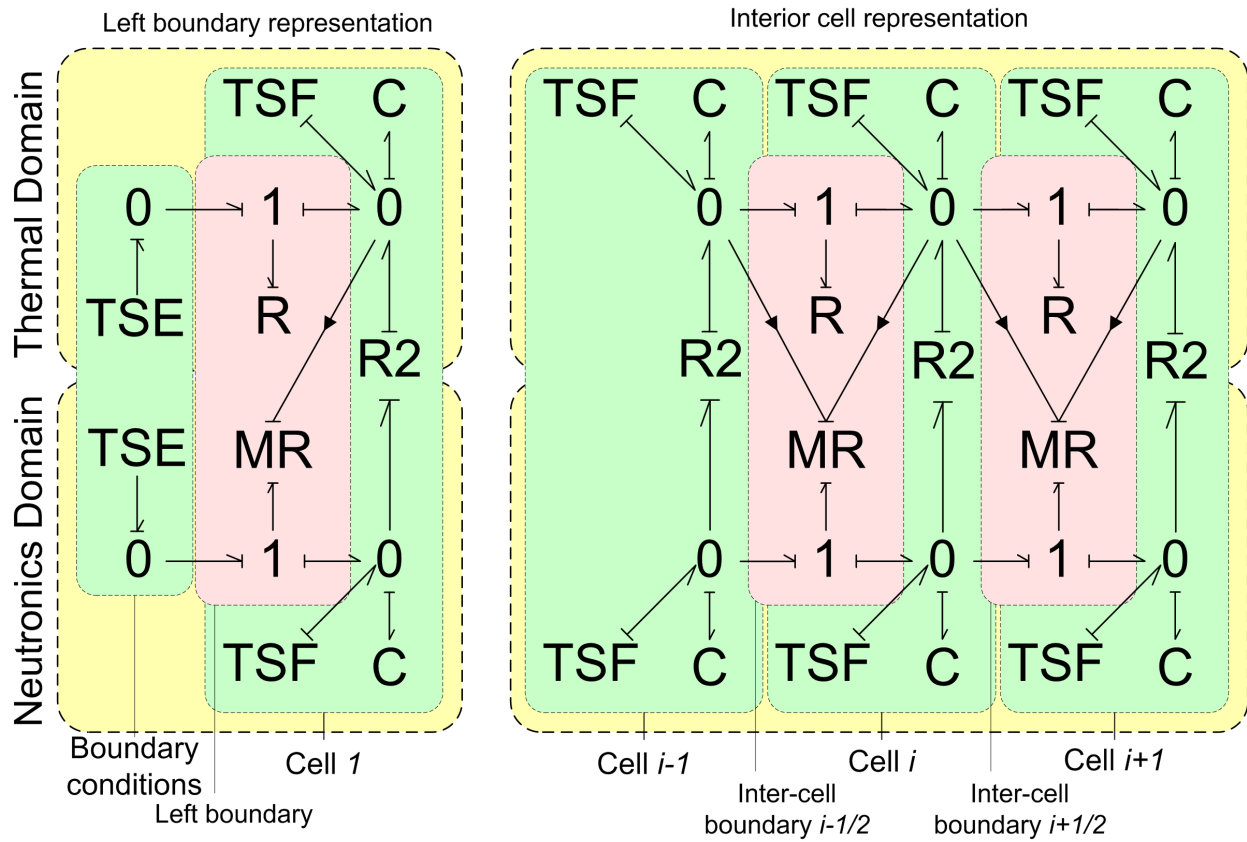
Equations (3.95) are generally nonlinear. However, in the absence of temperature dependence of the cross-sections, they become linear:

$$\begin{bmatrix} f_i^N \\ f_i^T \end{bmatrix} \cong \begin{bmatrix} \Delta V_i [\overline{\Sigma_{a,i}} - \nu \overline{\Sigma_{f,i}}] \\ \Delta V_i w \overline{\Sigma_{f,i}} \end{bmatrix} e_i^N \quad (3.96)$$

To summarize, the modulation of the neutron resistors by the temperatures introduces an MR element, modulated through signal bonds by the cell temperatures from the thermal 0 junctions. The temperature modulation of the net neutron removal rate by absorption, together with the heat generation rate by fission, introduces an R2 element. None of the other elements in the single physics discretization are affected by coupling. The initial conditions are also unchanged from the single physics treatments, and are given by Eq. (3.13) and (3.46) for the initial heat and neutron distributions, respectively.

With these modifications to the bond graph representation, the overall bond graph representation of coupled diffusion equations can finally be constructed. It is shown in Figure 12 below.

The constituent expressions for all elements in this bond graph representation are summarized in Table 14 below. Figure 12 and Table 14 together summarize the bond graph representation algorithm which arises from the finite volume discretization of the coupled diffusion equations. The initial conditions for the capacitors' displacements have been specified in Eqs. (3.13) and (3.46). The quantities which are affected by coupling in the constituent expressions in Table 14 are defined in this section; the quantities which remained unaffected are defined in the corresponding single-physics sections 3.1 and 3.2.



**Figure 12. Coupled Diffusion Bond Graph Representation**

**Table 14. Coupled Diffusion Bond Graph Constituent Expressions**

Domain	Element	Constituent Expressions <sup>12</sup>
Thermal	TSE <sub>left</sub>	$T_{left}(t)$
	R <sub>left</sub>	$\frac{\Delta x_1 \bar{\sigma}_{1/2}}{2A}$
	C <sub>i</sub>	$\frac{1}{\Delta V_i \rho c_p}$
	R <sub>i-1/2</sub>	$\frac{(\Delta x_{i-1} + \Delta x_i) \bar{\sigma}_{i-1/2}}{2A}$
	TSF <sub>i</sub>	$\dot{U}_{ex,i}(t)$
Neutronics	TSE <sub>left</sub>	$\phi_{left}(t)$
	R <sub>left</sub>	$\frac{\Delta x_1 C_{o1/2}(e_1)}{2A}$
	C <sub>i</sub>	$\frac{V_n}{\Delta V_i}$
	R <sub>i-1/2</sub>	$\frac{(\Delta x_{i-1} + \Delta x_i) C_{o_{i-1/2}}(e_{i-1}, e_i)}{2A}$
	TSF <sub>i</sub>	$S_{ex,i}(t)$
Coupled	R2 <sub>i</sub>	$\Delta V_i \left[ \bar{\Sigma}_{a,i}(e_{from}) - \nu \bar{\Sigma}_{f,i}(e_{from}) \right] e_{to}$ $\Delta V_i w \bar{\Sigma}_{f,i}(e_{from}) e_{to}$

This bond graph representation is the main result of this chapter. Constructing it completes step 1 of the bond graph process discussed in section 2.4 (p. 30) and summarized in Figure 2 (p. 32). To continue the analysis of the coupled 1D one-group nuclear reactor, steps 2-5 have to be executed. The whole purpose of using the bond graph formalism is that steps 2-5 are algorithmic, and can be executed automatically. A proof-of-concept code, designed to execute these steps, was developed, and is described in chapter 4 (p. 67).

The bond graph representation presented in this section still represents very limited physics. The main assumptions for this preliminary analysis are that these physics are 1D and one-group. In the next section, the extension to 2D two-group diffusion is discussed.

### 3.4. Multidimensional Multigroup Neutron Diffusion via Bond Graphs

Only neutron diffusion representation with bond graphs will be discussed in this section. The 2D heat diffusion bond graph can be constructed very similarly, and the coupling is also similar to section 3.3.

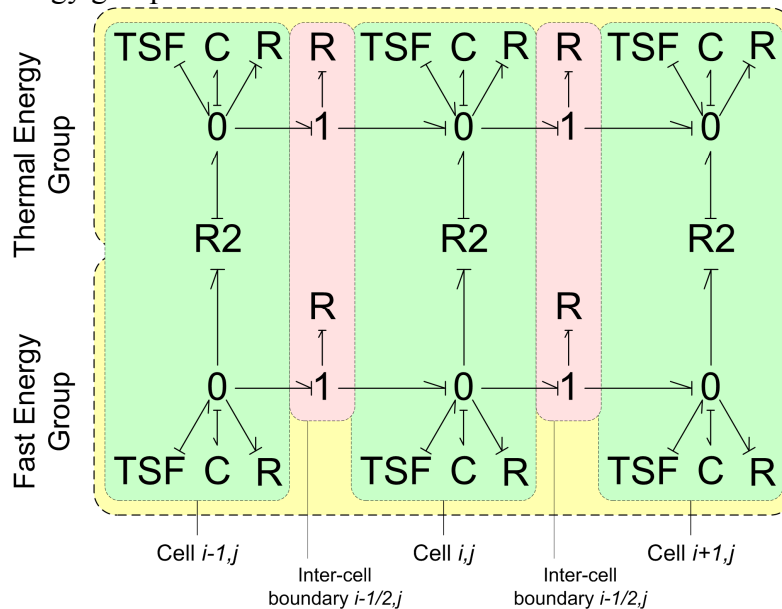
The model derived in this section is not utilized further in this text, due to requiring faster processing than the proof-of-concept code developed in chapter 4 (p. 67) can provide. For this

<sup>12</sup> The constituent expressions are either the linear elements' moduli (for capacitive and 1-port resistive elements), or the expressions the elements impose on their bonds (for source and R2 elements). For the R2 elements,  $f_{to}$ ,  $e_{to}$ ,  $f_{from}$  and  $e_{from}$  refer to the flow and effort on the bond that points toward the element, and the flow and effort on the bond that points away from the element, respectively. As discussed above,  $f_{to}$  is the flow on the neutron side, and  $f_{from}$  is the flow on the thermal side.

reason, instead of giving a detailed derivation, several physical arguments are made from which a bond graph representation is derived. The constituent expressions for the individual elements are not derived in the present work, but will be derived when 2D systems are modeled in the future.

By inspecting the bond graph representation developed for 1D one-group neutron diffusion in section 3.2, it can be seen, that the outcome is essentially a sequence of connected bond graph blocks. With every scalar flux is associated a 0-junction and a capacitor, with every absorption and fission production term is associated an R element connected to the junction. A TSF element imposes an external neutron source on the cell. The cells are connected with combinations of 1-junctions and R elements, which represent the diffusion terms.

In the 2D two-group case, two complications arise and will be discussed individually. First, there are now two neutron groups, which means that there are two scalar fluxes, one for each group. Corresponding to each flux is a neutron density, which, when integrated over the cell finite volume, is the group neutron total. These group fluxes and group neutron totals are related to each other identically to the 1D case, therefore, for each energy group, there will be a neutron capacitor and a 0-junction in each cell. Group-to-group scattering does not explicitly appear in the one-group model, but in a multigroup model, the neutrons may be removed from a group total in a cell not only by streaming or absorption, but also by scattering to another energy group. Scattering rate from group  $g$  to group  $g'$  is only a function of the source group  $g$  scalar flux and the differential scattering cross-section. If upscattering is not neglected, then the reverse is also true. Together, these two statements combine to form a net scattering rate from group  $g$  to group  $g'$ ; this rate is a function of both groups' fluxes, but is not proportional to the difference in them. This is an example of a 2-port coupling in which 2 rates (flows) are functions of 2 scalar fluxes (efforts). Such coupling can be achieved using an R2 element. If the scattering cross-sections are dependent on temperature, then the element would become a modulated MR2 element, and temperature would have to be supplied to it via a modulating bond. There is also now a TSF element for each energy group.



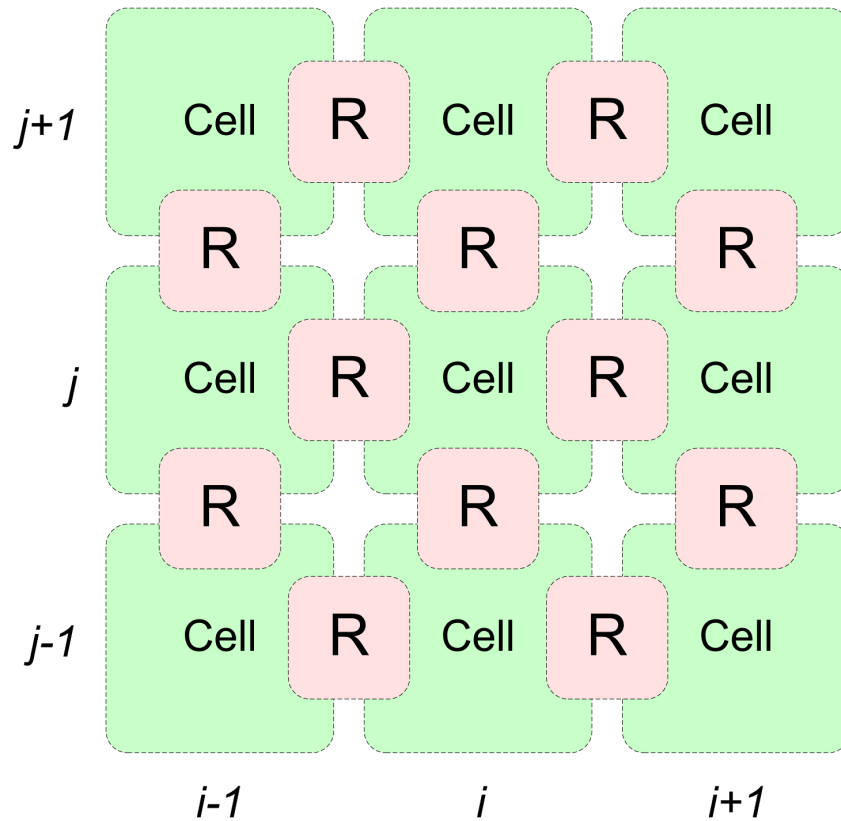
**Figure 13. 1D Slice of a Two-Group Neutron Diffusion Bond Graph Representation**



The second addition to the model is the transition from 1D to 2D. This addition is simpler, and is discussed in detail in literature about the finite volume method (Ref. [33]). The model is two-group, so there are now two rates of diffusion between nearby cells, one for each energy group. These rates are still proportional to the difference in corresponding scalar group fluxes. This means that there is now a set of a 1-junction with a connected R element between every finite cell for every energy group.

Combining these modifications, a 1D slice of the above representation is presented in Figure 13 above.

To expand this representation to 2D, the 1D slices have to be combined, with an inter-cell boundary between each slice. A schematic of the resulting representation is presented in Figure 14 below.



**Figure 14. 2D Diffusion Bond Graph Schematic Representation**

Adding heat diffusion into the model would simply place another layer of connected cells on top of the bond graph system above. This layer would be connected through R2 elements with both energy groups, assuming fast fission is possible. These elements would replace the R elements connected to the 0-junctions in the BGS above. The heat diffusion layer itself would modulate all of the resistive elements in the two neutronics layers in the BGS above.

Overall, the above schematic and representation clearly show that 2D multigroup bond graph representations can be constructed just as easily as 1D one-group, which means that the model studied in this text is an adequate proof-of-concept, but is not by any means a limit of the method.

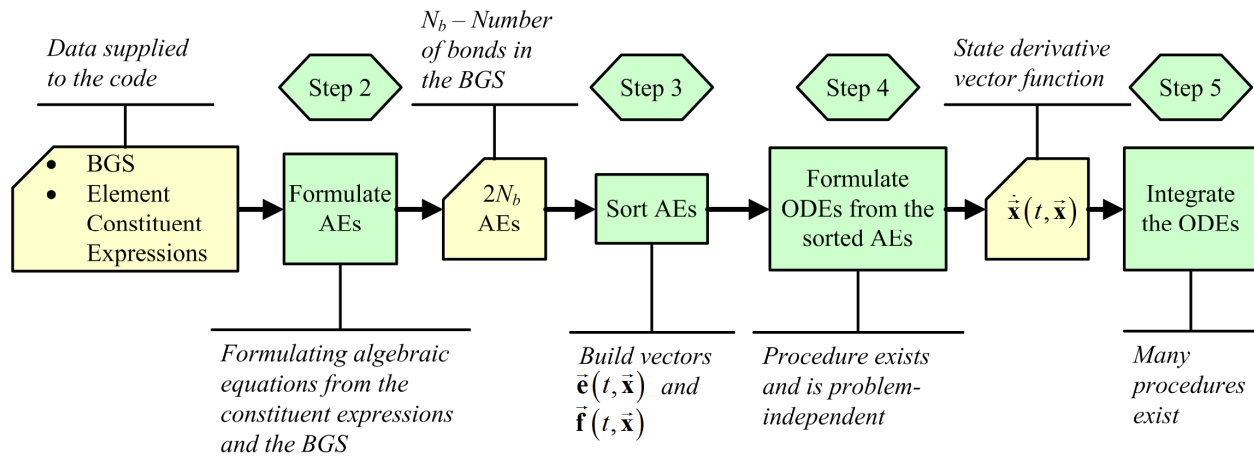
In the next chapter, a code is developed to process general bond graph representations.



## 4. Bond Graph Processing Code Development

As was discussed in section 2.4 (p. 30), after a bond graph representation is constructed for a system, the resulting BGS can be processed. This processing occurs in steps 2-5 of the bond graph method (see Figure 2, p. 32). The major benefit of the bond graph formalism is that the processing of the BGS can be completely automated. In this chapter, a code developed for this project that automates these four steps is described.

The four general steps that a bond graph processing code is supposed to execute are summarized in Figure 15 below.



**Figure 15. Bond Graph Processing Code Summary**

In Figure 15, the following notation is used:

$N_b$	Number of bonds in the BGS
$\bar{\mathbf{x}}$	State vector: vector of all storage variables in the bond graph
$\dot{\bar{\mathbf{x}}}$	State time derivative vector: the ODE system to integrate
$\bar{\mathbf{e}}(t, \bar{\mathbf{x}})$	Vector of all bonds' efforts as a function of time and state
$\bar{\mathbf{f}}(t, \bar{\mathbf{x}})$	Vector of all bonds' flows as a function of time and state

Several attempts have previously been made to automate these steps. A matrix-based vector field-type formulation has been developed for a purely linear BGS processing by Rosenberg (see Ref. [28]). However, in that formulation, along with most other commercial bond graph processing codes, only linear bond graph systems can be processed. This makes the commercially available codes, as well as the algorithmic formulation in Ref. [28] inapplicable for processing the coupled physics bond graphs which arise from nuclear reactor modeling.

Because no fitting bond graph processing code exists to automatically treat the bond graph representations of interest, such code was developed. This bond graph processing code package was named *BGSolver*, and it is a proof-of-concept bond graph processing MATLAB code package. It is important to note, that *BGSolver* was not meant to be a high performance tool for large system analysis. Instead, it is only a proof-of-concept package that demonstrates the possibility of executing the steps in Figure 15 and resolving nonlinear bond graph systems. *BGSolver* was written to satisfy the following requirements:

- To accept a BGS together with its constituent expressions as a file input.
- To be able to execute steps 2-5 in Figure 15 without any input from the user, except for the ODE integration method type and parameters.

- To be able to handle all of the element types necessary for modeling coupled diffusion equations, in all possible causalities.

In this chapter, the underlying algorithms and logic of the code are described in some detail. The mechanics of the implementation of the code, including the specific MATLAB file names and their roles, as well as the complete lists of constituent expressions for all possible element, expression and causality type combinations, are described in Appendix A (p. 83). The file format used to describe a BGS together with its constituent expressions is described in Appendix B (p. 89). A Graphical User Interface (GUI), which can be used to create simple BGS files for BGSolver, is described in Appendix C (p. 97).

The general algorithm underlying BGSolver's operation is presented in section 4.1. Section 4.2 summarizes how the code works simultaneously with symbolic and numeric expressions. The details of the most complicated and restrictive step in the processing algorithm, the sorting step, are described in section 4.3. Concise instructions for how to operate the code are presented in section 4.4, and a short discussion of possible means of acceleration of the code to make it applicable for large problems is given in section 4.5.

### **4.1. General Algorithm Description**

MATLAB was chosen as the environment to develop BGSolver. It is well understood, that for high performance computational applications, MATLAB is inapplicable (Ref. [36]); however, it is an ideal environment for preliminary algorithm development and testing. More important, MATLAB was chosen because of its Symbolic Math Engine (SME), provided as part of its Symbolic Math Toolbox (SMT), and the capability to sort the AEs by simply running a symbolic algebraic solver on them.

The MATLAB SME is a version of a Computer Algebra System (CAS), implemented in the MATLAB environment using the MuPAD symbolic engine. (Ref. [37]). In BGSolver, it was chosen to perform the most crucial step: the sorting of the AEs to yield the bond variable vectors as functions of time and state. Similar sorting capabilities are available in several other software packages, such as Maple (Ref. [38]), but those packages are generally meant exclusively for symbolic manipulation of equations, and are unfit for code development. Implementing a CAS in MATLAB, however, allows BGSolver to use powerful numerical tools while using symbolic sorting to formulate the state derivative vectors.

Many of the elements' constituent expressions, along with the state derivative vector itself, are MATLAB numerical expressions. This means that they are sequences of relatively low-level commands, accept only numerical data as inputs, and most importantly, cannot be manipulated symbolically. However, the sorting procedure requires at least some degree of such manipulation. For this reason, BGSolver had to be able to work with both symbolic and numeric expressions; section 4.2 outlines how this was accomplished. The symbolic sorting procedure itself is discussed in more detail in section 4.3. In this section, the general algorithm of the code is discussed.

Overall, BGSolver supports 18 different element types and 7 element expression types. BGSolver is only capable of treating fully causal bond graphs, which is acceptable, because as was shown in chapter 3 (p. 43), only fully causal bond graphs arise from coupled PDE discretization. For this reason, no augmentation procedure such as SCAP was implemented in BGSolver – the code assumes that causalities are provided for elements which accept several causalities, and that the appropriate/integral causality is used for all other elements.

All possible combinations of element, expression and causality types, and the resultant AEs are listed in Appendix A (p. 83). In this section, it is sufficient to recognize, that every element imposes 1 AE for every bond connected to it.

The 18 element types that BGSolver accepts are summarized in Table 15 below:

**Table 15. BGSolver Element Types**

Type of element	Element symbol	Element name	Admits modulating variables	Admits multiple causalities <sup>13</sup>
Source	SE	Source of Effort	No	No
	SF	Source of Flow	No	No
	MSE	Modulated Source of Effort	Yes	No
	MSF	Modulated Source of Flow	Yes	No
Storage	C	Capacitive element	No	No
	I	Inertial element	No	No
	MC	Modulated Capacitive element	Yes	No
	MI	Modulated Inertial element	Yes	No
Damping	R	Resistor	No	Yes
	R2	2-port Resistor	No	Yes
	MR	Modulated Resistor	Yes	Yes
	MR2	Modulated 2-port Resistor	Yes	Yes
Junction	1	1-junction	No	No
	0	0-junction	No	No
	TF	Transformer	No	No
	GY	Gyrator	No	No
	MTF	Modulated Transformer	Yes	No
	MGY	Modulated Gyrator	Yes	No

In BGSolver, because the systems treated are only fully causal, signal bonds are not explicitly implemented. Instead, the modulated elements that admit modulating variables, besides having associated constituent expressions, can have lists of modulating variables associated with them. These lists contain all bond variables that in a bond graph system would be delivered to the modulated elements via signal bonds; they may also contain time, since time-modulated elements are treated as regular modulated elements by BGSolver. Only resistive (damping) elements admit several different causalities. For reasons discussed below, it is generally necessary to specify the resistive elements' causal configurations to BGSolver, because otherwise it is impossible to write the resistive elements' equations.

The BGS is provided to BGSolver using a Bond Graph System Descriptor (BGSD) file. The BGSD file format was developed specifically for BGSolver and is described in detail in Appendix B (p. 89). The BGSD file can be thought of as a text representation of a bond graph system diagram – it contains information about all elements and bonds in the BGS, as well as the

<sup>13</sup> If an element “admits multiple causalities,” that implies, that the element is capable of imposing several different variable types onto its bond, and so is capable of enforcing several different types of AEs, depending on causality. A junction element admits more than one causal configuration, but they are all equivalent, because the AEs do not change regardless of the junction's causality. BGSolver only works with fully causal bond graphs, so storage elements only admit integral causalities.

initial conditions, but does not necessarily contain all elements' expressions. Some elements' expressions may be separate MATLAB functions, either stored in memory when BGSolver is executed, or saved as MATLAB function .m files. In this case, the BGSD file will only contain the MATLAB function handle that references these functions. For example, such a function handle can be written in the BGSD file as “@sin” and refer to the MATLAB numeric sine function. The numeric<sup>14</sup> functions which are referenced by handles are very important, because among other things, these can include the table interpolation and piecewise polynomial functions, which are essential for working with realistic material properties.

The BGSD file is a text file, and can, in principle, be either written completely by hand, or be generated using a script code. Such a script code generally will not do more than homogenize the material properties, integrate the source functions in space, and construct the BGS according to the bond graph representation algorithm for the physics of interest. Such algorithms were described for coupled neutron and heat diffusion in chapter 3 (p. 43).

A simple Graphical User Interface (GUI) was developed to construct BGSD files for small BGSs. This GUI was primarily used for testing out BGSolver's ability to process various types of bond graph systems, and not for constructing BGSD files for physical systems of interest. Nevertheless, the GUI proved useful both for debugging and for implementing new models for discrete dynamic systems, like electrical circuits. The GUI was named *BGSD\_Creator*, and is described in detail in Appendix C (p. 97).

Both elements and bonds are numbered in a BGSD file, for convenience. Generally, the algorithm is insensitive to how the elements are numbered, however, when creating a BGSD file, it is important to later have a way to interpret the numbered bond and storage variables. To do so, simple functions can be written that, for example, return the bond index  $i$  of the effort that corresponds to the temperature in cell  $j$ .

After the BGSD file is read, the symbolic variables can be instantiated. Generally, MATLAB does not require variable instantiation, as it is a weakly dynamically typed language, but symbolic variables are different, and have to be instantiated. The symbolic variables include time, an appropriate storage variable for every storage element, and a pair of bond variables for every bond. Additionally, every numeric expression has a corresponding symbolic variable, for reasons described in section 4.2.

After the symbolic variables are instantiated, AEs can be formulated. As was described in section 2.5 (p. 39), in step 3 of the bond graph method, the AEs are solved simultaneously to yield the bond variable vectors  $\bar{\mathbf{e}}(t, \bar{\mathbf{x}})$  and  $\bar{\mathbf{f}}(t, \bar{\mathbf{x}})$ . Combined, these bond variable vectors will be referred to as  $\bar{\mathbf{b}}(t, \bar{\mathbf{x}})$ . The algebraic equation system formed in step 2 of the bond graph method can therefore be viewed as a large vector of algebraic equations, of the following general form:

$$\bar{\mathbf{F}}(t, \bar{\mathbf{b}}, \bar{\mathbf{x}}) = \bar{\mathbf{0}} \in \mathbb{R}^{2N_b} \quad (4.1)$$

in which:

$\bar{\mathbf{F}}(t, \bar{\mathbf{b}}, \bar{\mathbf{x}})$	System of algebraic expressions
$\bar{\mathbf{b}}$	Vector of bond variables
$\bar{\mathbf{0}}$	Zero vector of length $2N_b$
$\mathbb{R}^{2N_b}$	$2N_b$ -dimensional real vector space

<sup>14</sup> Expressions which use MATLAB function handles are called “numeric,” and symbolic expressions are called “closed-form” in this chapter.

In step 2 of the bond graph method, BGSolver goes through every element in the BGS, and uses its tables of element types, expression types and causalities to construct the corresponding equations. These tables are provided in Appendix A. By the end of this process,  $\bar{\mathbf{F}}(t, \bar{\mathbf{b}}, \bar{\mathbf{x}})$  is constructed, and can be sorted. The “sorting” really refers to step 3 in the bond graph method, the solution of Eq. (4.1) to find  $\bar{\mathbf{b}}(t, \bar{\mathbf{x}})$ . Because numeric functions are used, the sorting procedure is actually more complicated, but conceptually the description presented here is adequate. The details of the implementation of the numeric functions in  $\bar{\mathbf{F}}(t, \bar{\mathbf{b}}, \bar{\mathbf{x}})$  are discussed in section 4.2, and the details of the sorting with these numeric functions are discussed in 4.3.

After the sorting procedure is complete,  $\bar{\mathbf{b}}(t, \bar{\mathbf{x}})$  is known. The flows on the capacitive elements’ bonds and the efforts on the inertial elements’ bonds form the state derivative vector  $\dot{\bar{\mathbf{x}}}(t, \bar{\mathbf{x}})$ . This allows BGSolver to proceed to step 4, and to trim the large vector  $\bar{\mathbf{b}}(t, \bar{\mathbf{x}})$  to retain only the state derivative vector  $\dot{\bar{\mathbf{x}}}(t, \bar{\mathbf{x}})$ . After the state derivative vector is formed, the only thing that remains is to integrate it in time to construct the state vector  $\bar{\mathbf{x}}(t)$ .

Integrating a first order system of ODEs  $\dot{\bar{\mathbf{x}}}(t, \bar{\mathbf{x}})$  is a standard problem in numerical analysis. When solving hyperbolic and parabolic PDEs, such problems arise when the Method of Lines (MOL) is used. (Ref. [35]). The time integration method chosen has to ensure stable integration, which, if an explicit method is used, normally places certain restrictions on the maximum time step size. An implicit method may also be used to integrate  $\dot{\bar{\mathbf{x}}}(t, \bar{\mathbf{x}})$ ; in that case, it will require solving a generally nonlinear algebraic system with each time step. An implicit time integration method is generally much more stable, which allows using large time steps without destabilizing the integrator. (Ref. [35]).

The main benefit of using the bond graph formalism for nuclear reactor multiphysics is the fact that treating the entire physical system simultaneously allows one to integrate  $\dot{\bar{\mathbf{x}}}(t, \bar{\mathbf{x}})$  using any stable numerical method. Such method may be of 2<sup>nd</sup> or greater order of accuracy in time; adaptive time stepping can also be used. By comparison, operator splitting, the conventional approach to reactor multiphysics, is limited to 1<sup>st</sup> order in time, even if a high order time integrator is used to integrate the individual physics at each time step. (Ref. [1]). Operator splitting is capable of yielding higher order accuracy in time only if iterations between the individual physics are performed, which is computationally expensive, and is generally not done. (Ref. [17]). This means, that assuming an appropriate time integration method is chosen, the bond graph formalism can easily exceed the maximum possible order of accuracy in time that operator splitting can yield.

A particularly important time integration technique involves using an implicit method, like Crank-Nicolson, together with an efficient nonlinear Jacobean-Free Newton-Krylov (JFNK) system solver method. JFNK is based on avoiding the construction of the Jacobian at every time step while still making use of parts of the approximate Jacobian, computed through a variety of means. (Ref. [1]). In the future, JFNK-based implicit time integrators may be one of the possible types of solver used in step 5 for large-scale bond graph processing codes. They are not used in the present work.

MATLAB has several efficient implicit and explicit time integrators. The time integrators, in general, use adaptive time stepping based on the approximate Jacobian of  $\dot{\bar{\mathbf{x}}}(t, \bar{\mathbf{x}})$ ; therefore, these methods do not have a theoretical order of accuracy in time. BGSolver is set up to construct a MATLAB function handle for the state derivative vector  $\dot{\bar{\mathbf{x}}}(t, \bar{\mathbf{x}})$ , and then to either

use the MATLAB time integrator specified in the BGSF file, or to use a custom-written one. It may be useful to custom-write simple methods for theoretical studies, but for practical problems, due to the expected stiffness, more efficient, and likely implicit, time integrators should be used.

While it is not formally a part of the bond graph method, after  $\bar{\mathbf{x}}(t)$  is computed, BGSolver is set up to post-process the results. In doing so, the code evaluates the bond variable vectors  $\bar{\mathbf{e}}(t, \bar{\mathbf{x}})$  and  $\bar{\mathbf{f}}(t, \bar{\mathbf{x}})$  as functions of time. These calculations are generally useful, because in practice, the state variables themselves are normally of little practical value, and the bond variables are truly what interests the user. In the case of coupled nuclear reactors, the reaction rates (neutron flows), temperature distributions (thermal efforts) and power rates (thermal flows) are generally of far greater importance than the neutron and thermal energy totals in finite cells. Quantities like total assembly power as a function of time can then be computed by simple algebraic manipulations of  $\bar{\mathbf{e}}(t, \bar{\mathbf{x}})$  and  $\bar{\mathbf{f}}(t, \bar{\mathbf{x}})$ ; these operations are problem-specific, and are not done by BGSolver.

In the next section, the different expression types that BGSolver accepts are summarized.

## 4.2. Symbolic and Numeric Expressions Summary

All elements listed in Table 15 above have one or more associated constituent expressions. As stated above, 7 types of constituent expressions are supported by BGSolver. Table 16 below summarizes these expression types, and outlines which ones require a specified causality if they are used for a damping element. (Otherwise, causality never needs to be specified). In general, not every expression type is usable with every element type – for example, source elements have no input from the bond that is connected to them, and so cannot use CE, NE, CME or NME expression types. The full table of compatibilities of element and expression types is provided in Appendix A (p. 83).

**Table 16. BGSolver Expression Types**

<b>Expression type's symbol</b>	<b>Expression type's name</b>	<b>Requires causality specification for R and MR elements<sup>15</sup></b>
CC	Constant Coefficient	No
CMC	Closed-form Modulated Coefficient	No
NMC	Numeric Modulated Coefficient	No
CE	Closed-form Expression	Yes
NE	Numeric Expression	Yes
CME	Closed-form Modulated Expression	Yes
NME	Numeric Modulated Expression	Yes

A CC expression is simply a number, either imposed by a source element, or a linear element's modulus. For a simple linear element, like a capacitor, the CC expression takes the following form:

$$e = Cq \tag{4.2}$$

Here  $C$  acts as the constant coefficient. An R2 element with a CC expression is still a linear constant coefficient expression, of the following form:

<sup>15</sup> R2 and MR2 elements always require causality specifications.



$$\begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix} = \begin{bmatrix} R2_{11} & R2_{12} \\ R2_{21} & R2_{22} \end{bmatrix} \begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix} \quad (4.3)$$

In this case, efforts are inputs and flows are outputs. Four different causal configurations for R2 elements are possible, all of which are supported by the code. They are described in detail in Appendix A.

In general, coefficient-type expressions are expressions in which one or more variables set by the element is/are proportional to one or more of the input variables. Equation (4.2) is an example of such proportionality. The input variable may be a storage variable for the storage elements (Eq. (4.2)), or one or more bond variables delivered to the element (Eq. (4.3)). This proportionality constant may be modulated, and is the coefficient itself. For source elements, the “coefficient” is actually the value of the bond variable imposed by the source.

The more general expression types are “expressions” – general functions of the input variable (or variables), which are imposed on the output variables of the element. These expressions may be modulated, in which case the modulating variables act as additional inputs to the functions. Two examples of such expressions are:

$$e = C(q) \quad (4.4)$$

$$\begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix} = \begin{bmatrix} R2_1(e_{to}, e_{from}) \\ R2_2(e_{to}, e_{from}) \end{bmatrix} \quad (4.5)$$

Source elements cannot have CE, NE, CME or NME as their expression types, because they do not have input variables. Modulated source elements are always of either CMC or NMC type.

Both expression-type and coefficient-type expressions can be of two classes: a closed-form expression and a numeric expression. The difference comes from the fact that, as stated above, MATLAB SME is used for sorting the AEs, and later for constructing and evaluating the state derivative vector. All symbolic variables for SME are objects, which can be equated to an expression in terms of other symbolic variables, and therefore can be symbolically manipulated. Generally, any SME expression can be considered closed-form, because of a finite set of algebraic expressions that can be implemented in SME and combined to form other mathematical expressions.

However, MATLAB functions cannot be directly handled by SME, because MATLAB functions are just pointers to sequences of low level commands in memory. MATLAB functions are more general than combinations of closed-form algebraic expressions; most importantly, they can involve piecewise polynomials and table look-up functions. Both of these are necessary for working with practical data. For these reasons, while MATLAB functions may have closed-form equivalents, when making SME work with a MATLAB function, it is generally necessary to do so in several steps. Doing so is illustrated in an example.

First, the following variables are assumed to be in memory:

<code>fun</code>	MATLAB function handle to be evaluated by the SME
<code>v</code>	MATLAB vector which contains the inputs for <code>fun</code> (assumed known)
<code>n</code>	Symbolic variable which represents <code>fun</code> for the SME
<code>expr</code>	Symbolic expression to evaluate, which <code>fun</code> is a part of

In practice, `expr` may be a state derivative function, `fun` a resistive variable’s constituent expression, and `v` a time-dependent input for the constituent expression. When integrating `expr` in time, it would therefore be necessary to evaluate `fun` at every time step, so

a procedure has to be devised for doing so. Once  $V$  is in memory, the following code can be used to evaluate `expr`:

```
F = fun(V); % Evaluating the function numerically
X = subs(expr,n,F); % Plugging the numeric value into the expression
```

In general, it's possible that  $V$  itself depends on other numeric functions' values, so the cycle has to be repeated more than once. Whether or not this is necessary, and if so, what order the numeric variables have to be evaluated in, is taken care of by the sorting procedure, described in the next section.

### 4.3.Sorting Procedure Description

As was described in section 4.1, the sorting procedure involves obtaining the bond variable vectors  $\vec{e}(t, \vec{x})$  and  $\vec{f}(t, \vec{x})$  by solving the AE system  $\vec{F}(t, \vec{b}, \vec{x}) = \vec{0}$ . In section 4.2, it was shown, that some of the constituent expressions provided to the BGSD may in fact be numeric MATLAB functions. The symbol  $\vec{n}$  will be used for these numeric functions. It was also discussed, that while these numeric functions' lists of input variables are provided, they cannot be manipulated directly by the SME. Therefore, more fully, this problem can be stated as follows:

Given	<ul style="list-style-type: none"> <li>· <math>\vec{F}(t, \vec{b}, \vec{x}, \vec{n}) = \vec{0}</math></li> <li>· List of variable inputs to each member of <math>\vec{n}</math>; may include variables in <math>\{t, \vec{b}, \vec{x}\}</math></li> <li>· List of variables in <math>\vec{b}</math> which constitute <math>\vec{x}</math></li> <li>· List of MATLAB function handles for <math>\vec{n}</math></li> </ul>
Find	<ul style="list-style-type: none"> <li>· <math>\vec{b}(t, \vec{x}, \vec{n})</math></li> <li>· A way to evaluate <math>\vec{n}</math> based on <math>\vec{b}(t, \vec{x}, \vec{n})</math></li> <li>· <math>\vec{x}(t, \vec{x}, \vec{n})</math></li> </ul>

With  $\vec{F}(t, \vec{b}, \vec{x}, \vec{n})$  constructed, the system can be trivially solved for  $\vec{b}(t, \vec{x}, \vec{n})$  using SMT's `solve` command. The outcome is a vector of symbolic expressions of length  $2N_b$ , as expected.

Due to  $\vec{n}$  being present in the bond variable vector, it will generally be impossible to evaluate  $\vec{x}(t, \vec{x}, \vec{n})$  based on time  $t$  and state  $\vec{x}$  alone, because at least some of the numeric variables  $\vec{n}$  will have to be evaluated. This problem was solved using the notion of *numeric layers*. Consider the following algorithm:

- 10 Set layer counter  $l = 1$ .
- 20 Loop through the bond variable vector  $\vec{b}(t, \vec{x}, \vec{n})$ , recording which bond variables can be evaluated using only time  $t$  and state  $\vec{x}$ , without evaluating any numeric variables. Assign these bond variables to the numeric layer 1. Together, these bond variables will be denoted  $\vec{b}^{(l)} = \vec{b}^{(1)}$ .
- 30 Loop through the numeric variable vector  $\vec{n}$ , recording which numeric variables can be evaluated using only  $t$ ,  $\vec{x}$  and  $\vec{b}^{(l)}$ . Assign these numeric variables to the numeric layer 1. Together, these numeric variables will be denoted  $\vec{n}^{(l)} = \vec{n}^{(1)}$ .
- 40 Increment layer counter  $l = l + 1$ .

- 50 Loop through  $\vec{\mathbf{b}}(t, \vec{\mathbf{x}}, \vec{\mathbf{n}})$ , recording which bond variables can be evaluated using only  $t$ ,  $\vec{\mathbf{x}}$  and all  $\vec{\mathbf{n}}^{(k)}$  with  $k < l$ . Assign these bond variables to the numeric layer  $l$ .
- 60 Loop through  $\vec{\mathbf{n}}$ , recording which numeric variables can be evaluated using only  $t$ ,  $\vec{\mathbf{x}}$  and all  $\vec{\mathbf{b}}^{(k)}$  with  $k \leq l$ . Assign these numeric variables to the numeric layer  $l$ .
- 70 If all bond and numeric variables have been assigned to a layer, quit. Otherwise, go to 40.

BGSolver implements this algorithm after  $\vec{\mathbf{F}}(t, \vec{\mathbf{b}}, \vec{\mathbf{x}}, \vec{\mathbf{n}}) = \vec{\mathbf{0}}$  is solved and  $\vec{\mathbf{b}}(t, \vec{\mathbf{x}}, \vec{\mathbf{n}})$  is found. Having all bond and numeric variables assigned to numeric layers is very useful, because it allows the code to execute the expressions in an appropriate order to evaluate  $\vec{\mathbf{b}}(t, \vec{\mathbf{x}}, \vec{\mathbf{n}})$ , and therefore  $\dot{\vec{\mathbf{x}}}(t, \vec{\mathbf{x}})$ . An inspection of the above algorithm shows, that if variables are evaluated in the order in which they are assigned to layers (that is, layer 1 bond variables first, then layer 1 numeric variables, layer 2 bond variables, etc.), the expressions can be executed sequentially with no problem for the code.

Assigning all symbolic expressions in the problem to layers completes the sorting step, and allows the code to construct  $\dot{\vec{\mathbf{x}}}(t, \vec{\mathbf{x}})$ .

The only issue with this approach is speed. The looping described in the algorithm above is time-consuming. More importantly, the substitution of numeric values into symbolic expressions in MATLAB is a slow and memory-intensive process, due to the object-oriented nature of the SME. For these reasons, it is well understood, that while the code demonstrates the theoretical ability to automatically generate and integrate the state equations, it would be inapplicable for large problems.

Section 4.5 contains a more detailed discussion of how the code may be made more robust. In the next section, BGSolver is concisely summarized.

#### 4.4. Final Code Description

When a problem is being modeled with bond graphs, the bond graph processing code BGSolver expects a BGSD text file input. BGSD files are documented in Appendix B.

During processing, BGSolver will request the initial and final time for the time integrator. All other processing is fully automatic.

The code's main product are four arrays:  $\mathbb{T}$ ,  $\mathbb{X}$ ,  $\mathbb{E}$  and  $\mathbb{F}$ . They are summarized below:

$\mathbb{T}$	Vertical vector of time points at which $\mathbb{X}$ , $\mathbb{E}$ and $\mathbb{F}$ were computed
$\mathbb{X}$	Vertical array of horizontal state vectors $\vec{\mathbf{x}}$ , evaluated at time points $\mathbb{T}$
$\mathbb{E}$	Vertical array of horizontal effort vectors $\vec{\mathbf{e}}$ , evaluated at time points $\mathbb{T}$
$\mathbb{F}$	Vertical array of horizontal flow vectors $\vec{\mathbf{f}}$ , evaluated at time points $\mathbb{T}$

These four arrays can be further post-processed; more advanced plots and even videos of the outcomes can be constructed. All this processing is problem-specific and is not done by BGSolver.

The Symbolic Math Engine used for the sorting of AEs in step 3 significantly slows down the problem, and prevents the code from being expandable. In the next section, the possibility of expansion of the code to more realistic problems is discussed.

#### 4.5. Possible Code Acceleration for Large Problems

As discussed in sections 4.2 and 4.3, the use of SME to sort the AEs places an unavoidable limit on BGSolver's productivity. This procedure, along with symbolic

manipulations altogether, has to be replaced to accelerate the code sufficiently for large problems.

Although not discussed in more detail in this text, one possible way of doing so may involve taking the vector field approach proposed by Rosenberg (Ref. [28]), and expanding it to nonlinear systems. Specifically, it may be possible to construct a multi-layer state derivative vector of the following form:

$$\dot{\bar{\mathbf{x}}} = \mathbf{A}\bar{\mathbf{g}}(t, \mathbf{B}\bar{\mathbf{h}}(t, \bar{\mathbf{x}})) \quad (4.6)$$

in which:

$\mathbf{A}, \mathbf{B}$	Constant matrices
$\bar{\mathbf{g}}, \bar{\mathbf{h}}$	Vector functions

Such construction may allow for a fully numeric bond graph processing, as well as fully constructing a Jacobian matrix for faster time integration and uncertainty quantification.

## 5. Benchmark Problems

In chapter 3 (p. 43), 1D one-group coupled neutron and heat diffusion problem was represented with bond graphs. In chapter 4 (p. 67), a bond graph processing code capable of automatically processing such representations was described. The next logical step is to test the resulting approach using a benchmark problem.

Unfortunately, most multiphysics benchmark problems (i.e., Ref. [18]) available in literature are too complicated for the proof-of-concept code package tested here. For this reason, a benchmark problem fit for the preliminary testing has to be developed. The Method of Manufactured Solutions (MMS) is used in this chapter to do so.

Section 5.1 concisely summarizes the theory of MMS, and a benchmark problem is constructed using MMS in section 5.2. A sample run result for this problem is shown and discussed in section 5.3. The conclusions about the potential of the use of bond graphs for coupled nuclear reactor multiphysics simulations are summarized in section 5.4.

### 5.1. Method of Manufactured Solutions Theory

Consider a sample 1D PDE system in Eq. (5.1):

$$\frac{\partial \bar{\mathbf{u}}}{\partial t} = \bar{\mathbf{F}}\left(t, \bar{\mathbf{u}}, \frac{\partial \bar{\mathbf{u}}}{\partial x}, \frac{\partial^2 \bar{\mathbf{u}}}{\partial x^2}\right) + \bar{\mathbf{S}}(t, x) \quad (5.1)$$

in which:

$\bar{\mathbf{u}}(t, x)$	Unknown vector function
$\bar{\mathbf{F}}\left(t, \bar{\mathbf{u}}, \frac{\partial \bar{\mathbf{u}}}{\partial x}, \frac{\partial^2 \bar{\mathbf{u}}}{\partial x^2}\right)$	Known function of time and $\bar{\mathbf{u}}$ which defines the PDE
$\bar{\mathbf{S}}(t, x)$	External source vector

Equations like Eq. (5.1) are generally not solvable analytically. However, assuming sufficiently smooth material properties and  $\bar{\mathbf{u}}$ ,  $\bar{\mathbf{F}}$  can be evaluated analytically. Therefore, while no exact solution will generally exist for Eq. (5.1), an exact solution can be intentionally constructed using the following procedure:

1. Pick an arbitrary, but realistic and smooth set of material properties.
2. Using these material properties, construct  $\bar{\mathbf{F}}$ .
3. Pick an arbitrary, but realistic (order-of-magnitude) for the material properties chosen, exact solution function vector  $\bar{\mathbf{u}}(t, x)$ .  $\bar{\mathbf{u}}(t, x)$  has to be sufficiently smooth to be analytically evaluated by  $\bar{\mathbf{F}}$ .
4. Analytically differentiate  $\bar{\mathbf{u}}(t, x)$  to construct  $\frac{\partial \bar{\mathbf{u}}}{\partial t}$  and  $\bar{\mathbf{F}}\left(t, \bar{\mathbf{u}}, \frac{\partial \bar{\mathbf{u}}}{\partial x}, \frac{\partial^2 \bar{\mathbf{u}}}{\partial x^2}\right)$ .
5. Subtract  $\bar{\mathbf{F}}\left(t, \bar{\mathbf{u}}, \frac{\partial \bar{\mathbf{u}}}{\partial x}, \frac{\partial^2 \bar{\mathbf{u}}}{\partial x^2}\right)$  from  $\frac{\partial \bar{\mathbf{u}}}{\partial t}$  to obtain the corrective source function:

$$\bar{\mathbf{S}}(t, x) = \frac{\partial \bar{\mathbf{u}}}{\partial t} - \bar{\mathbf{F}}\left(t, \bar{\mathbf{u}}, \frac{\partial \bar{\mathbf{u}}}{\partial x}, \frac{\partial^2 \bar{\mathbf{u}}}{\partial x^2}\right) \quad (5.2)$$

6. The material properties chosen above, together with the corrective source from Eq. (5.2), now constitute a problem with a known exact solution  $\bar{\mathbf{u}}(t, x)$ . Numerical methods can be tested using this problem; a good numerical method is expected to converge to the exact solution  $\bar{\mathbf{u}}(t, x)$ .

It must be recognized, that MMS works best when  $\bar{\mathbf{S}}$  sets the shape of the solution, but not its general scale. The reason for this, is that if the solution is dominated by  $\bar{\mathbf{S}}$ , and not by  $\bar{\mathbf{F}}$ , any error made in discretizing  $\bar{\mathbf{S}}$  will overshadow the potential errors made from the discretization of  $\bar{\mathbf{F}}$ . This is undesirable for testing numerical methods, so care must be taken to keep  $\bar{\mathbf{S}}$  on a smaller scale than  $\bar{\mathbf{F}}$ .

MMS is clearly applicable to the coupled diffusion problem discussed in chapter 3 (p. 43). A benchmark problem for the coupled problem is constructed using MMS in section 5.2.

## 5.2. Benchmark Problem Construction

The coupled diffusion problem's PDEs are repeated below, for convenience:

$$\frac{\partial}{\partial t} n(t, x) = -\frac{\partial}{\partial x} J(t, x) + \nu \Sigma_f(x, T) \phi(t, x) - \Sigma_a(x, T) \phi(t, x) + s_{ex}(t, x) \quad (5.3)$$

$$J(t, x) = -D(x, T) \frac{\partial}{\partial x} \phi(t, x) \quad (5.4)$$

$$\phi(t, x) = V_n n(t, x) \quad (5.5)$$

$$\frac{\partial}{\partial t} u_v(t, x) = -\frac{\partial}{\partial x} u_v''(t, x) + w \Sigma_f(x, T) \phi(t, x) + \dot{u}_{v,ex}(t, x) \quad (5.6)$$

$$u_v''(t, x) = -k(x) \frac{\partial}{\partial x} T(t, x) \quad (5.7)$$

$$T(u_v) = \frac{1}{\rho c_p} u_v \quad (5.8)$$

The diffusion coefficient is constructed using the cross-sections and the mean scattering angle cosine (assumed constant):

$$D(x, T) = \frac{1}{3(\Sigma_t(x, T) - \bar{\mu}_0 \Sigma_s(x, T))} \quad (5.9)$$

The spatial domain considered below is  $0 \leq x \leq 1$ . The other geometric and material properties used are listed in Table 17. Unless explicitly specified otherwise, the material properties are all constants.

**Table 17. Benchmark Problem's Material and Geometric Properties**

Property	Value/Expression	Property	Value/Expression
$\Sigma_s(x, T)$	6	$k$	1
$\Sigma_a(x, T)$	$2.5 + 0.5T$	$c_p$	1
$\Sigma_f(x, T)$	$2 - 0.3T$	$\rho$	1
$\bar{\mu}_0$	$2/3$	$w$	1
$V_n$	1	$A$	1
$\nu$	2.5		

With the material properties chosen, the exact solution can be chosen to construct the corresponding corrective sources. A convenient choice of the exact solutions are polynomial functions, scaled in time. Even if initially a rise in power is observed (if the starting temperature is sufficiently low), the properties in Table 17 will clearly result in the power eventually stabilizing, because of the growing absorption and reducing fission cross-sections. The exact solution should reflect this behavior, and simulate a rise to power stabilized by a negative thermal feedback.

The following shape functions were used for the exact solutions:

$$T_{\infty}(x) = 5.56x^4 - 11.11x^3 + 4.94x^2 + 0.61x + 2.5 \quad (5.10)$$

$$\phi_{\infty}(x) = 33.33x^4 - 66.67x^3 + 33.67x^2 - 0.33x \quad (5.11)$$

in which:

$T_{\infty}(x)$	The solution that the temperature approaches as $t \rightarrow \infty$
$\phi_{\infty}(x)$	The solution that scalar flux approaches as $t \rightarrow \infty$

As described above, these shape functions were scaled in time to yield the exact solution functions:

$$T(t, x) = (1 - e^{-t})T_{\infty}(x) + e^{-t} \quad (5.12)$$

$$\phi(t, x) = (1 - e^{-t})\phi_{\infty}(x) \quad (5.13)$$

From Eqs. (5.12) and (5.13), the initial condition functions can be constructed:

$$T^0(x) = 1 \quad (5.14)$$

$$\phi^0(x) = 0 \quad (5.15)$$

in which:

$T^0(x)$	The initial condition function for temperature
$\phi^0(x)$	The initial condition function for scalar flux

The Dirichlet boundary condition functions can also be constructed:

$$T_{left}(t) = 2.5(1 - e^{-t}) + e^{-t} \quad (5.16)$$

$$T_{right}(t) = 2.5(1 - e^{-t}) + e^{-t} \quad (5.17)$$

$$\phi_{left}(t) = 0 \quad (5.18)$$

$$\phi_{right}(t) = 0 \quad (5.19)$$

The material properties, initial and boundary conditions, and the exact solution function are known. It remains to construct the corrective source functions. First, defining the shape derivative functions:

$$T'_{\infty}(x) = \frac{\partial}{\partial x} T_{\infty}(x) \quad (5.20)$$

$$T''_{\infty}(x) = \frac{\partial^2}{\partial x^2} T_{\infty}(x) \quad (5.21)$$

$$\phi'_{\infty}(x) = \frac{\partial}{\partial x} \phi_{\infty}(x) \quad (5.22)$$

$$\phi''_{\infty}(x) = \frac{\partial^2}{\partial x^2} \phi_{\infty}(x) \quad (5.23)$$

With these definitions, the corrective source functions can be found by plugging the material properties in Table 17 and Eqs. (5.12) and (5.13) into Eqs. (5.3)-(5.8). The following corrective sources result:

$$\dot{u}_{v,ex}(t, x) = e^{-t}(T_{\infty}(x) - 1) - w\Sigma_f(x, T(t, x))\phi(t, x) - (1 - e^{-t})T''_{\infty}(x) \quad (5.24)$$

$$s_{ex}(t, x) = e^{-t}\phi_{\infty}(x) + \Sigma_a(x, T(t, x))\phi(t, x) - \nu\Sigma_f(x, T(t, x))\phi(t, x) -$$

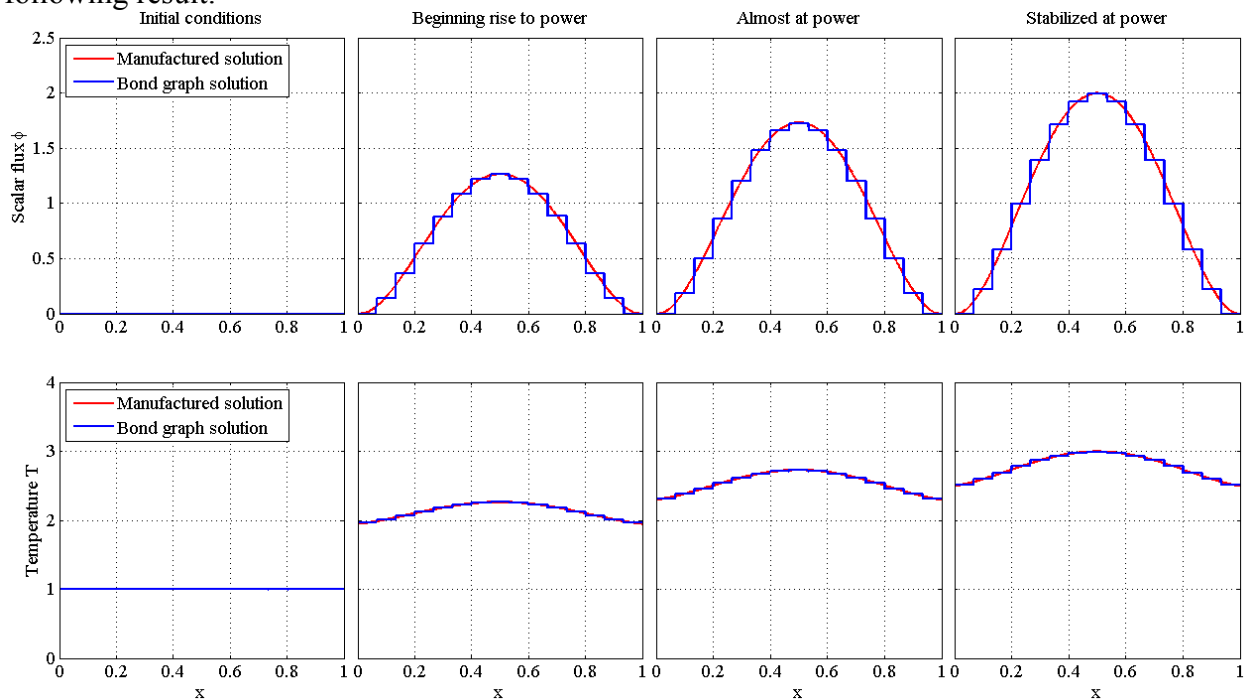
$$-D(x, T(t, x))(1 - e^{-t})\phi''_{\infty}(x) + \frac{(1 - e^{-t})^2 \phi'_{\infty}(x) T'_{\infty}(x)}{6[\Sigma_t(x, T(t, x)) - \bar{\mu}_0 \Sigma_s(x, T(t, x))]^2} \quad (5.25)$$

With these corrective source functions, the benchmark problem is constructed. In section 5.3, the simulation results are provided.

### 5.3. Benchmark Simulation Results

A MATLAB script was developed to construct a BGSD file using the discretization algorithm in section 3.3 (p. 57). The script accepts the geometric and material properties and external source functions as inputs. For the benchmark problem, these parameters were outlined in section 5.2 above.

The domain was discretized into 15 finite slabs. Applying BGSolver to the resultant BGSD file and using MATLAB ode45 Runge-Kutta-type ODE integrator produced the following result:



**Figure 16. Benchmark Simulation Results**

Figure 16 clearly shows that the code traced the solution in time, and did not diverge from it even during the transient. The temperature and flux peaks of the numerical solution also did not appear to desynchronize in time, which is a common problem in operator splitting. This verification clearly demonstrates, that the coupled multiphysics bond graph-based approach can be effectively used to simulate 1D one-group coupled diffusion equations, and is very accurate in doing so even with a relatively coarse spatial mesh.

### 5.4. Conclusions

A benchmark coupled neutron and thermal diffusion problem was created using the method of manufactured solutions. The benchmark was simulated using BGSolver, and demonstrated that the method successfully modeled the transient. The scalar flux and temperature peaks did not desynchronize in time at all, which is the main desired advantage for fully coupled simulations.

For these reasons, the test can be considered successful. The bond graph-based approach to coupled nuclear reactor simulation was shown to be accurate and stable, and to exceed operator splitting in accuracy. Larger tests are required to fully understand the approach's limitations.



## 6. Summary and Recommendations for Future Work

Section 6.1 summarizes the main results of this dissertation. Section 6.2 outlines several possible directions to continue this work in.

### 6.1. Summary

The objectives of the project were:

1. To identify a method to represent neutron diffusion via bond graph formalism.
2. To identify a method to couple neutron diffusion and thermal diffusion via bond graph formalism.
3. To develop a code fit for automatic processing of systems modeled via bond graph formalism.
4. To construct and run a benchmark and draw conclusions about the method's feasibility.

All these objectives were successfully achieved. Neutron diffusion, and thermal diffusion of heterogeneous materials, were discretized using rigorous finite volume formulations. The resulting discretization was then used as the basis for a bond graph representation of these physics. A MATLAB-based bond graph processing code was developed, and utilized to simulate a benchmark coupled neutron and thermal diffusion problem. The simulated solution accurately traced the exact solution in time, and did not desynchronize even during the fastest part of the transient. This fact illustrated the desired advantage of the coupled modeling approach over the conventional operator splitting.

Currently, the symbolic engine in the bond graph processing code is the limiting factor in the developed code package. However, for the above reasons, the bond graph-based approach to coupled nuclear reactor simulation was deemed feasible, and appropriate for high-fidelity simulation. Suggestions were made for accelerating the bond graph processing code; when those are implemented, the code can be used for running large simulations.

### 6.2. Recommendations for Future Work

Bond graph formalism was deemed appropriate for high fidelity coupled simulations of nuclear reactors. However, the proof-of-concept code BGSolver written as part of this project is clearly inappropriate for large problems, because of the symbolic engine used in the code. Another issue with the code is its use of MATLAB ODE integrators to integrate the state derivative vectors; these integrators are efficient on single machines, but are not parallelizable over clusters. This is a strong limitation against the use of the code for large problems.

For these reasons, it is recommended to continue this work as follows:

1. To identify a fully numeric method to replace the sorting step in the bond graph processing code. One direction to look in is to expand Rosenberg's vector field-based approach.
2. To implement this numeric method as the sorting procedure, thus phasing the MATLAB symbolic engine out of the code.
3. To utilize a higher performance, parallelizable time integration package for integrating the large ODE systems. One such package which should be explored is TRILINOS (Ref. [39]).
4. To develop and implement bond graph representation techniques for more complicated physics, such as advection, two phase flow, multigroup  $P_N$  and  $S_N$  neutron transport.
5. To continue expanding the code's capabilities, increasing both its speed and range of representable physics.

The most important recommendation of these is the first one – it is critical to turn the sorting procedure in the code to a fully numeric one. Doing so will remove a major bottleneck in the code, which currently serves as the main limiting factor.

## Appendix A. Bond Graph Processing Code Documentation

The bond graph processing code developed for this project was named BGSolver. The version associated with this text is BGSolver 1.01. The BGSD file format (summarized in section Appendix B, p. 89), the BGSD\_Creator and the BGSD\_Generator tools are all versioned similarly, and are all in versions 1.01.

BGSolver 1.01 utilizes MATLAB Symbolic Math Toolbox. SMT's syntax conventions changed between versions; the code is known to work in MATLAB R2009a x86 version. It may be incompatible with other versions of MATLAB and/or SMT.

The BGSD\_Creator tool for creating simple BGSD files is described in Appendix C (p. 97).

BGSolver 1.01 supports 18 element types and 7 expression types. Section 4.2 (p. 72) summarizes the differences between the expression types and classes, and the consequences for the code of using each expression type. Only fully causal systems are supported. The constituent equations for the source, storage and junction elements for all their possible expression types are listed in Table 18 below. The constituent equations for the resistive elements for all their possible expression types and causalities are listed in Table 19.

The following notation is used in these tables:

$e$	Effort on the single bond connected to the element
$f$	Flow on the single bond connected to the element
$A$	Constant
$A(\vec{m})$	Function of modulating variables
$\vec{m}$	Vector of modulating variables (may include time)
$q$	Capacitive element's displacement
$p$	Inertial element's momentum
$e_{to}$	Effort on the "to bond" – the bond pointing toward the 2-port element
$e_{from}$	Effort on the "from bond" – the bond pointing away from the 2-port element
$f_{to}$	Flow on the "to bond" – the bond pointing toward the 2-port element
$f_{from}$	Flow on the "from bond" – the bond pointing away from the 2-port element
$e_i$	Effort on bond $i$ connected to the junction element
$f_i$	Flow on bond $i$ connected to the junction element
$i$	Bond index of the first bond connected to the junction element
$i+k$	Bond index of the last bond connected to the junction element
$d_i$	$d_i \equiv \begin{cases} +1 & \text{for bonds directed toward the junction} \\ -1 & \text{for bonds directed away from the junction} \end{cases}$
$T$	Transformer's coefficient
$T(\vec{m})$	Modulated transformer's coefficient
$G$	Gyrator's coefficient
$G(\vec{m})$	Modulated gyrator's coefficient
$G(e)$	Resistive element's constituent expression for $f$

The differences between numeric and closed-form expressions were outlined in section 4.2. They are different in implementation, but mathematically, their constituent equations are identical.

**Table 18. Source, Storage and Junction Element and Expression Type Compatibility**

Element type		Expression type	Corresponding constituent equation(s)
SE	Source of Effort	CC	$e = A$
SF	Source of Flow	CC	$f = A$
MSE	Modulated Source of Effort	CMC, NMC	$e = A(\vec{m})$
MSF	Modulated Source of Flow	CMC, NMC	$f = A(\vec{m})$
C	Capacitive element	CC	$e = Cq$
		CE, NE	$e = C(q)$
I	Inertial element	CC	$f = Ip$
		CE, NE	$f = I(p)$
MC	Modulated Capacitive element	CMC, NMC	$e = C(\vec{m})q$
		CME, NME	$e = C(q, \vec{m})$
MI	Modulated Inertial element	CMC, NMC	$f = I(\vec{m})p$
		CME, NME	$f = I(p, \vec{m})$
1	1-junction		$f_i = f_{i+1} = \dots = f_{i+k-1} = f_{i+k}$ $\sum_{l=i}^{i+k} d_l e_l = 0$
0	0-junction		$e_i = e_{i+1} = \dots = e_{i+k-1} = e_{i+k}$ $\sum_{l=i}^{i+k} d_l f_l = 0$
TF	Transformer	CC	$e_{to} = T e_{from}, f_{out} = T f_{to}$
GY	Gyrator	CC	$e_{to} = G f_{out}, e_{from} = G f_{to}$
MTF	Modulated Transformer	CMC, NMC	$e_{to} = T(\vec{m}) e_{from}, f_{from} = T(\vec{m}) f_{to}$
MGY	Modulated Gyrator	CMC, NMC	$e_{to} = G(\vec{m}) f_{from}, e_{from} = G(\vec{m}) f_{to}$

None of the elements in Table 18 are affected by causality, because fully causal system is assumed. However, resistive elements are still affected by causality. Except for the coefficient forms of the 1-port resistive element, the elements' constituent expressions vary depending on the causality they are in. Table 19 below lists all possible combinations of expression and causality types for pure resistive elements.

**Table 19. Resistive Element Expression and Causality Type Compatibility**

Element type		Expression type	Causality		Corresponding constituent equation(s)
			Input	Output	
R	Resistive element	CC	$f$	$e$	$e = Rf$
			$e$	$f$	
		CE, NE	$f$	$e$	$e = R(f)$
			$e$	$f$	$f = G(e)$
R2	2-port Resistive element	CC	$f_{to}$	$e_{to}$	$e_{to} = \begin{bmatrix} R2_{11} & R2_{12} \\ R2_{21} & R2_{22} \end{bmatrix} \begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix}$
			$f_{from}$	$e_{from}$	$e_{from} = \begin{bmatrix} R2_{11} & R2_{12} \\ R2_{21} & R2_{22} \end{bmatrix} \begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix}$
			$e_{to}$	$f_{to}$	$f_{to} = \begin{bmatrix} R2_{11} & R2_{12} \\ R2_{21} & R2_{22} \end{bmatrix} \begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix}$
			$e_{from}$	$f_{from}$	$f_{from} = \begin{bmatrix} R2_{11} & R2_{12} \\ R2_{21} & R2_{22} \end{bmatrix} \begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix}$
			$f_{to}$	$e_{to}$	$e_{to} = \begin{bmatrix} R2_{11} & R2_{12} \\ R2_{21} & R2_{22} \end{bmatrix} \begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix}$
			$e_{from}$	$f_{from}$	$f_{from} = \begin{bmatrix} R2_{11} & R2_{12} \\ R2_{21} & R2_{22} \end{bmatrix} \begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix}$
			$e_{to}$	$f_{to}$	$f_{to} = \begin{bmatrix} R2_{11} & R2_{12} \\ R2_{21} & R2_{22} \end{bmatrix} \begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix}$
			$f_{from}$	$e_{from}$	$e_{from} = \begin{bmatrix} R2_{11} & R2_{12} \\ R2_{21} & R2_{22} \end{bmatrix} \begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix}$
		CE, NE	$\begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix}$	$\begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix}$	$\begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix} = \begin{bmatrix} R2_1(f_{to}, f_{from}) \\ R2_2(f_{to}, f_{from}) \end{bmatrix}$
			$\begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix}$	$\begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix}$	$\begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix} = \begin{bmatrix} R2_1(e_{to}, e_{from}) \\ R2_2(e_{to}, e_{from}) \end{bmatrix}$
			$\begin{bmatrix} f_{to} \\ e_{from} \end{bmatrix}$	$\begin{bmatrix} e_{to} \\ f_{from} \end{bmatrix}$	$\begin{bmatrix} e_{to} \\ f_{from} \end{bmatrix} = \begin{bmatrix} R2_1(f_{to}, e_{from}) \\ R2_2(f_{to}, e_{from}) \end{bmatrix}$
			$\begin{bmatrix} e_{to} \\ f_{from} \end{bmatrix}$	$\begin{bmatrix} f_{to} \\ e_{from} \end{bmatrix}$	$\begin{bmatrix} f_{to} \\ e_{from} \end{bmatrix} = \begin{bmatrix} R2_1(e_{to}, f_{from}) \\ R2_2(e_{to}, f_{from}) \end{bmatrix}$

Modulated resistive elements have similar constituent expressions. However, their coefficients and expressions also accept modulating variables (time and bond variables) as inputs. The possible combinations of expression and causality types for modulated resistive elements are listed in Table 20 below.

**Table 20. Modulated Resistive Element Expression and Causality Type Compatibility**

Element		Expression type	Causality		Corresponding constituent equation(s)
			Input	Output	
MR	Modulated Resistive element	CMC, NMC	$f, \vec{m}$	$e$	$e = R(\vec{m})f$
			$e, \vec{m}$	$f$	
		CE, NE	$f, \vec{m}$	$e$	$e = R(f, \vec{m})$
			$e, \vec{m}$	$f$	$f = G(e, \vec{m})$
MR2	Modulated 2-port Resistive element	CMC, NMC	$\begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix}, \vec{m}$	$\begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix}$	$\begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix} = \begin{bmatrix} R2_{11}(\vec{m}) & R2_{12}(\vec{m}) \\ R2_{21}(\vec{m}) & R2_{22}(\vec{m}) \end{bmatrix} \begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix}$
			$\begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix}, \vec{m}$	$\begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix}$	$\begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix} = \begin{bmatrix} R2_{11}(\vec{m}) & R2_{12}(\vec{m}) \\ R2_{21}(\vec{m}) & R2_{22}(\vec{m}) \end{bmatrix} \begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix}$
			$\begin{bmatrix} f_{to} \\ e_{from} \end{bmatrix}, \vec{m}$	$\begin{bmatrix} e_{to} \\ f_{from} \end{bmatrix}$	$\begin{bmatrix} e_{to} \\ f_{from} \end{bmatrix} = \begin{bmatrix} R2_{11}(\vec{m}) & R2_{12}(\vec{m}) \\ R2_{21}(\vec{m}) & R2_{22}(\vec{m}) \end{bmatrix} \begin{bmatrix} f_{to} \\ e_{from} \end{bmatrix}$
			$\begin{bmatrix} e_{to} \\ f_{from} \end{bmatrix}, \vec{m}$	$\begin{bmatrix} f_{to} \\ e_{from} \end{bmatrix}$	$\begin{bmatrix} f_{to} \\ e_{from} \end{bmatrix} = \begin{bmatrix} R2_{11}(\vec{m}) & R2_{12}(\vec{m}) \\ R2_{21}(\vec{m}) & R2_{22}(\vec{m}) \end{bmatrix} \begin{bmatrix} e_{to} \\ f_{from} \end{bmatrix}$
		CE, NE	$\begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix}, \vec{m}$	$\begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix}$	$\begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix} = \begin{bmatrix} R2_1(f_{to}, f_{from}, \vec{m}) \\ R2_2(f_{to}, f_{from}, \vec{m}) \end{bmatrix}$
			$\begin{bmatrix} e_{to} \\ e_{from} \end{bmatrix}, \vec{m}$	$\begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix}$	$\begin{bmatrix} f_{to} \\ f_{from} \end{bmatrix} = \begin{bmatrix} R2_1(e_{to}, e_{from}, \vec{m}) \\ R2_2(e_{to}, e_{from}, \vec{m}) \end{bmatrix}$
			$\begin{bmatrix} f_{to} \\ e_{from} \end{bmatrix}, \vec{m}$	$\begin{bmatrix} e_{to} \\ f_{from} \end{bmatrix}$	$\begin{bmatrix} e_{to} \\ f_{from} \end{bmatrix} = \begin{bmatrix} R2_1(f_{to}, e_{from}, \vec{m}) \\ R2_2(f_{to}, e_{from}, \vec{m}) \end{bmatrix}$
			$\begin{bmatrix} e_{to} \\ f_{from} \end{bmatrix}, \vec{m}$	$\begin{bmatrix} f_{to} \\ e_{from} \end{bmatrix}$	$\begin{bmatrix} f_{to} \\ e_{from} \end{bmatrix} = \begin{bmatrix} R2_1(e_{to}, f_{from}, \vec{m}) \\ R2_2(e_{to}, f_{from}, \vec{m}) \end{bmatrix}$

BGSolver 1.01 is a package of MATLAB .m files, consisting of:

BGSolve	The main script file which executes the code. All other files are referenced by this file.
readBGSD	A function file that reads a BGSD file and returns a data structure with all the data from the BGS.
getbvids	A function file that takes a vector of symbolic bond variables as an argument, and returns their bond variable IDs. This operation is used when numeric layers are being assigned.
getnvids	A function file that takes a vector of symbolic numeric variables as an argument, and returns their numeric variable IDs. This operation is used when numeric layers are being assigned.

<code>evalxdot</code>	A function file that takes a time value, a state vector and an appropriate data structure as arguments, and evaluates the state derivative vector with these inputs. This operation is used to construct a MATLAB function handle for the state derivative vector to then integrate the ODE system.
<code>evalbvars</code>	A function file that takes a vector of time values, an array of state vectors and an appropriate data structure as arguments, and evaluates the bond variables at these time values. This operation is used in post-processing.
BGSD Generator 1.01 consists of a single file:	
<code>writeBGSD</code>	A function file that takes an appropriate data structure, describing a BGS, as an argument, and writes a BGSD file based on this data structure. This operation is used by problem-discretizing scripts which are set up to construct the BGSD files.
BGSD Creator 1.01 also consists of a single file:	
<code>BGSD_Creator</code>	A function file that starts up a MATLAB Java GUI that asks the user a series of questions and constructs and writes a BGSD file based on the answers. This operation is useful for debugging the BGSolver code, and for trying new types of bond graph system models.
The code's main product are four arrays: $\mathbb{T}$ , $\mathbb{X}$ , $\mathbb{E}$ and $\mathbb{F}$ . They are summarized below:	
$\mathbb{T}$	Vertical vector of time points at which $\mathbb{X}$ , $\mathbb{E}$ and $\mathbb{F}$ were computed
$\mathbb{X}$	Vertical array of horizontal state vectors $\bar{\mathbf{x}}$ , evaluated at time points $\mathbb{T}$
$\mathbb{E}$	Vertical array of horizontal effort vectors $\bar{\mathbf{e}}$ , evaluated at time points $\mathbb{T}$
$\mathbb{F}$	Vertical array of horizontal flow vectors $\bar{\mathbf{f}}$ , evaluated at time points $\mathbb{T}$





## Appendix B. BGSF File Format Documentation

The purpose of the BGSF file is to contain all information about a specific fully causal bond graph system, including all of its elements and their constituent equations or values, initial conditions, and bonds which connect the elements. The file can also contain additional information about elements, such as their names, in the form of character strings.

BGSF stands for Bond Graph System Descriptor. The BGSF file is referred to as “the BGSF” in this documentation.

BGSF format differs between versions of code, because different versions of BGSolver support different types of elements and expressions. The format provided here is for code version 1.01.

The format presented in this appendix is not problem-specific, and can be used for describing any BGS, modeling any physical system. Any fully causal BGS described by this format can be processed by BGSolver 1.01.

### General notes

A BGSF file consists of 7 sections:

- File Header Section
- Specified System Information Section
- Element List Section
- Bond Directionality Matrix Section
- Expression List Section
- Initial Value List Section
- File End Section

They are presented separately below. Notice that only text in Courier font appears in the BGSF, the rest is just comments.

### Line format

A BGSF file consists of lines. Almost any line has the following general syntax:

```
keyword string
```

Here, the keyword indicates what type of information (if any) is contained in the string. The keyword cannot be empty, but the string can be. They are separated by exactly one space.

Notice that the keyword appears exactly as shown in the text below, while the string is different based on whatever information that string is intended to contain. For example, if the BGSF file is intended for software version 1.01, the following will be the SVF line in the File Header:

```
SVF 1.01
```

The 7 sections of the BGSF file are presented below. In every section description the syntax of the section is given first, followed by an explanation of the meaning of each line.

### File Header

```
BGSF
SVF Version
SVC Version
NOTES If_Notes
Notes_Entry
NOTESEND
```

## Comments

- I. BGSD – “Bond Graph System Descriptor” keyword. It is the first line of the file header.
- II. SVF – “Software Version For” keyword.  
Version – The version number of the processing software that the BGSD was created for.
- III. SVC – “Software Version Created” keyword.  
Version – The version of the software that the BGSD was created by. If SVC and SVF mismatch, the processing software throws a warning.
- IV. NOTES – “Notes” keyword. It indicates the beginning of the optional Notes section.  
If\_Notes – Whether or not there are any optional notes entered. It can be one of the following keywords:
  - 0 – There are no optional notes entered. The NOTESEND keyword follows.
  - 1 – There are optional notes entered. The Notes\_Entry follows.
- V. Notes\_Entry – The optional notes entry. This is a line of any text, in double quotation marks.
- VI. NOTESEND – “Notes End” keyword. It’s present whether or not any optional notes were entered.

## Specified System Information

SCI SC\_Info  
SMI SM\_Info  
SLI SL\_Info  
SRI SR\_Info  
SEI SE\_Info  
SSI SS\_Info  
SNI SN\_Info

## Comments

- I. SCI – “Specified Causality Information” keyword.  
SC\_Info – Specified information about causality of the BGS. This can be one of the following keywords:
  - FC – “Full Causality”
  - UC – “Unknown Causality”Software v1.01 only works with BGSDs that are known to be fully causal.
- II. SMI – “Specified Modulation Information” keyword.  
SM\_Info – Specified information about the modulated elements of the BGS. This can be one of the following keywords:
  - NMEP – “No Modulated Elements Present”
  - TMEP – “Time-Modulated Elements Present”
  - SMEP – “Signal-Modulated Elements Present”
  - UMP – “Unspecified if Modulation Present”Software v1.01 does not use this information, but it will be used in future versions.
- III. SLI – “Specified Linearity Information” keyword.  
SL\_Info – Specified information about linearity of the BGS. This can be one of the following keywords:

- ACC – “All Constant Coefficients” – All elements’ expressions are constant coefficients
- ABSCON – “All But Source Constant” – Source elements’ expressions may be time-modulated, the other elements’ expressions are constant coefficients
- NCC – “Non-Constant Coefficients” – Expressions other than constant coefficients may be present
- UL – “Unspecified Linearity”

Software v1.01 does not use this information, but it will be used in future versions.

#### IV. SRI – “Specified Reduction Information” keyword.

SR\_Info – Specified information about whether or not the BGS should be reduced. A reduction of a BGS refers to replacing it with an alternative BGS with identical state equations, but fewer bonds, elements, or both. Typically this is achieved by eliminating bypassed 1-port elements, or replacing a junction structure with a simpler equivalent junction structure. SR\_Info can be one of the following keywords:

- FR – “Fully Reduced”
- UR – “Unknown if Reduced”

#### V. SEI – “Specified Error Information” keyword.

SE\_Info – Specified information about whether or not to check the BGS for errors. Examples of errors include 1-port elements with multiple bonds connected to them, 2-port elements with any number of bonds other than 2 connected to them, and similar errors that prevent the BGS from making physical sense. SE\_Info can be one of the following keywords:

- CE – “Check for Errors”
- DNCE – “Do Not Check for Errors”

#### VI. SSI – “Specified Solver Information” keyword.

SS\_Info – Specified information about the solver to use to solve the ODEs formulated by the sorter. SS\_Info can be one of the following keywords:

- MSSNS – “MuPAD Symbolic Solver then Numeric Solver”
- NSO – “Numeric Solver Only”

#### VII. SNI – “Specified Numeric Information” keyword.

SN\_Info – Specified information about the numeric solver to use if a numeric solver is used to solve the ODEs formulated by the sorter. SN\_Info can be one of the following keywords:

- ode15s – Use MATLAB’s ode15s solver (stiff systems, low accuracy)
- ode45 – Use MATLAB’s ode45 solver (non-stiff systems, medium accuracy)
- ode113 – Use MATLAB’s ode113 solver (non-stiff systems, high accuracy)

## Element List

```
EL Number
ENAMES Name_IDs
Name_List
EINFOS Info_IDs
Info_List
ETYPES Element_Types
ELEND
```

## Comments

- I. EL – “Element List” keyword.  
Number – Number of bond graph elements in the Bond Graph System being described in the BGSD.
- II. ENAMES – “Element Names” keyword.  
Name\_IDs – List of 1s and 0s, one value for every element in the BGSD, listed in the order of increasing element IDs. 1 indicates that the element’s name is specified, 0 indicates that the element’s name is not specified. The specified names themselves are listed in the Name\_List.
- III. Name\_List – List of the specified names of the elements, one name per line, in double quotation marks. Only the elements whose corresponding values in the Name\_IDs list are 1s have specified names.
- IV. EINFOS – “Element Infos” keyword.  
Info\_IDs – List of 1s and 0s, one value for every element in the BGSD, listed in the order of increasing element IDs. 1 indicates that the element’s info is specified, 0 indicates that the element’s info is not specified. The specified infos themselves are listed in the Info\_List.
- V. Info\_List – List of the specified infos of the elements, one info per line, in double quotation marks. Only the elements whose corresponding values in the Info\_List are 0s have specified infos.
- VI. ETYPES – “Element Types” keyword.  
Element\_Types – List of element types, one element type for every element in the list, listed in the order of increasing element IDs. Each type can be one of the following 18 keywords:  
SE, SF, I, C, R, TF, GY, 1, 0, MSE, MSF, MI, MC, MR, MTF, MGY, R2, MR2
- VII. ELEND – “Element List End” keyword. This keyword concludes the Element List.

## Bond Directionality Matrix

BDM Number  
Table  
BDMEND

## Comments

- I. BDM – “Bond Directionality Matrix” keyword.  
Number – Number of bonds in the BGS.
- II. Table – Table which stores the bond directionality information according to the following logic:  
For every nonzero cell value:  
Row Number of that Cell – EID of the element the bond points From  
Column Number of that Cell – EID of the element the bond points To  
Value of that Cell – the bond’s ID Number. Bond IDs are similar to element IDs described above.  
The resulting table is an  $N_e \times N_e$  matrix which is mostly 0s.  $N_e$  is the number of elements in the BGS.
- III. BDMEND – “Bond Directionality Matrix End” keyword.

## Expression List

Notice that the blank lines are not in the BGSD, and are given here just for simplifying reading. The indented lines are also only indented to simplify reading and are not indented in the BGSD.

```
EXPRL Number
EXPRIDS Expression_IDs
EXPRTYPES Expression_Types
```

```
Expression_Blocks
```

```
EXPREND
```

### Comments

I. EXPRL – “Expression List” keyword.

Number – Number of bond graph elements with constituent expressions in the BGSD.

II. EXPRIDS – “Expression IDs” keyword.

Expression\_IDs – List of element IDs of elements with constituent expressions, in order of increasing element IDs.

III. EXPRTYPES – “Expression Types” keyword.

Expression\_Types – List of expression types, one expression type for every element in the expression list, listed in the order of increasing element IDs. Each type can be one of the following 7 keywords:

CC, CMC, NMC, CE, NE, CME, NME

Expression\_Blocks consists of an expression block for every element with a constituent expression in the BGSD. Every such block has the following form:

```
EID ID_Number
EC Causality
Expression
```

IV. EID – “Element ID” keyword.

ID\_Number – Unique identification number of the element whose expression follows.

V. EC – “Element Causality” keyword.

Causality – Causality vector of the element whose expression follows. For 2-port elements the vector is in the order of input, output. Causality vector consists of a single entry for every port of the element, with the following possible values:

- 0 – Causality for this element is irrelevant/unspecified.
- 1 – Effort is input on the port.
- 2 – Flow is input on the port.

VI. Expression – Element’s expression. The expression format depends on the expression type and also on the element’s type. The formats are presented below:

- For all elements other than R2 and MR2, each element has one constituent expression.
  - For CC expression type:

```
Coefficient
```

- For CMC expression type:

MPARAM List  
 Mod\_CF\_Expression  
   ○ For NMC expression type:  
 MPARAM List  
 Mod\_Function\_Handle  
   ○ For CE expression type:  
 CF\_Expression  
   ○ For NE expression type:  
 Function\_Handle  
   ○ For CME expression type:  
 MPARAM List  
 Mod\_CF\_Expression  
   ○ For NME expression type:  
 MPARAM List  
 Mod\_Function\_Handle  
   • For R2 and MR2 elements, each element has two or four constituent expressions.  
     ○ For CC expression type, matrix of the following form:  
 Coefficient1   Coefficient2  
 Coefficient3   Coefficient4  
     ○ For CMC expression type:  
 MPARAM List  
 Mod\_CF\_Expression1  
 Mod\_CF\_Expression2  
 Mod\_CF\_Expression3  
 Mod\_CF\_Expression4  
     ○ For NMC expression type:  
 MPARAM List  
 Mod\_Function\_Handle1  
 Mod\_Function\_Handle2  
 Mod\_Function\_Handle3  
 Mod\_Function\_Handle4  
     ○ For CE expression type:  
 CF\_Expression1  
 CF\_Expression2  
     ○ For NE expression type:  
 Function\_Handle1  
 Function\_Handle2  
     ○ For CME expression type:  
 MPARAM List  
 Mod\_CF\_Expression1  
 Mod\_CF\_Expression2  
     ○ For NME expression type:  
 MPARAM List  
 Mod\_Function\_Handle1  
 Mod\_Function\_Handle2

The keywords are described below:

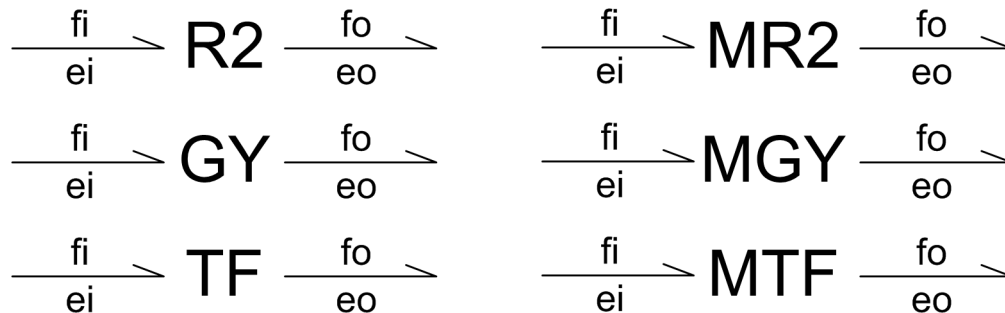
- a. Coefficient – A single number.
- b. MPARAM – “Modulating Parameters” keyword.
- c. List – Double-quoted, comma-separated list of parameters modulating the expression. They are in the following format:
  - Effort on bond #  $N$  – ‘ $eN$ ’
  - Flow on bond #  $N$  – ‘ $fN$ ’
  - Time – ‘ $t$ ’
- d. Mod\_CF\_Expression – A MuPAD expression in double quotes for the expression being defined. It is a one-line expression which involves some of the modulating parameters listed in the MPARAM line, as well as the independent variable. It does not involve the quantity actually being defined (the output), as that is assumed, based on the causality. So, if trying to define an expression of the format  $f(x) = x^2$ , the Expression will be:

“ $x^2$ ”

The independent variable(s) are specified according to the following:

- $q$  – Displacement associated with the port (Capacitive element).
- $p$  – Momentum associated with the port (Inertial element).
- $e$  – Effort input associated with the port (Resistive element in Conductance causality).
- $f$  – Flow input associated with the port (Resistive element in Resistance causality).

For 2-port elements, the  $i$  and  $o$  (“input” and “output,” or “to” and “from”) bonds are specified according to the convention in Figure 17 below. Notice that “input” bonds still have an output quantity, and “output” bonds still have an input quantity, according to the bonds’ causalities. Here, “input” and “output” is just a naming convention for the bonds’ directions.



**Figure 17. Bond Directionality Convention for 2-Port Elements**

- $e_i$  – Effort input on the input bond of a 2-port element.
- $f_i$  – Flow input on the input bond of a 2-port element.
- $e_o$  – Effort input on the output bond of a 2-port element.
- $f_o$  – Flow input on the output bond of a 2-port element.

For modulated elements, the following modulating independent variables are also used (all modulating variables have to be listed in the List of Modulating Parameters):

- $t$  – Time.
- $eN$  – Effort on bond #  $N$ .
- $fN$  – Flow on bond #  $N$ .

- e. Mod\_Function\_Handle – A MATLAB function handle. The function handle’s arguments are in the following order:

- Independent variable(s), for a 2-port element – as a  $1 \times 2$  vector. For a 2-port element, the vector's elements are first that from the input bond, then that from the output bond.
  - Modulating parameters, in a vector. The order of elements in that vector is the same as the order of variables in the List of Modulating Parameters.
- f. `CF_Expression` – A Maple expression in double quotes for the expression being defined. It is a one-line expression, similar to the `Mod_CF_Expression`, defined above, but without the modulating variables. Everything else is the same.
- g. `Function_Handle` – A MATLAB function handle, similar to the `Mod_Function_Handle`, defined above, but without the modulating variables. Everything else is the same. After the functions are listed, is the `EXPRLEND` keyword.
- VII. `EXPRLEND` – “Expression List End” keyword.

### Initial Value List

```
EV0L Number
EV0IDS Initial_Value_IDs
EV0 Initial_Values
EVOLEND
```

#### Comments

- I. `EV0L` – “Element Values at 0 List” keyword.  
 Number – Number of storage elements with initial values in the BGSD.
- II. `EV0IDS` – “Element Values at 0 IDs” keyword.  
`Initial_Value_IDs` – List of element IDs of elements with initial values, in order of increasing element IDs.
- III. `EV0` – “Element Values at 0” keyword.  
`Initial_Values` – List of initial values, separated by spaces. The initial values are given in the order of increasing element IDs, with the corresponding element IDs given in the `EV0IDS` list above.
- IV. `EVOLEND` – “Element Values at 0 List End” keyword.

### File End

```
BGSDEND
```

#### Comments

`BGSDEND` – “Bond Graph System Descriptor End” keyword.



## Appendix C. BGSD File Creator Documentation

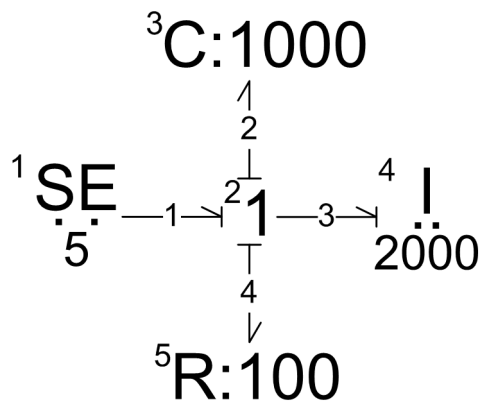
BGSD\_Creator 1.01 is a Graphical User Interface (GUI) for creating BGSD files for simple bond graph systems. The BGSD file format is described in Appendix B (p. 89). In this appendix, BGSD\_Creator is used to construct the bond graph system for the RLC circuit from section 2.5 (p. 39).

Like BGSolver 1.01, BGSD\_Creator 1.01 has only been tested in MATLAB R2009a x86.

To create a BGSD file using BGSD\_Creator, the user is expected to have the following:

- A complete diagram of the bond graph system
- All bonds must be numbered
- All elements must be numbered
- All constituent expressions of the elements must be known
- All initial values of the storage variables must be known

Consider the following bond graph system:



**Figure 18. Annotated Bond Graph System Diagram**

This BGS is clearly identical to the one in section 2.5, except in this one, specific numerical values are assigned to each element, and the elements are numbered. All zero initial conditions are assumed.

After starting the BGSD\_Creator by typing “BGSD\_Creator” in the MATLAB prompt, and clicking “Create a new BGSD file” button, the user is asked for some text information about the system being modeled. This part does not affect the BGS itself and can be clicked through. After that, the GUI requests additional information about the BGS; this screen is shown in Figure 19 below.

Most questions asked at this stage are not utilized by BGSolver 1.01; however, the choice of the numeric ODE solver chosen at this stage may make a difference for some problems. Leaving the default stiff solver `ode15s` on the user can proceed to the next stage, in which the numbers of elements and bonds are requested. The bond graph system above has 5 elements and 4 bonds, which can be entered. After these numbers are entered, the element types, and optional names and infos (not used by the solver, but can be used in post-processing) can be specified. The element type selection screen, with the elements from Figure 18 already entered, is shown below in Figure 20.

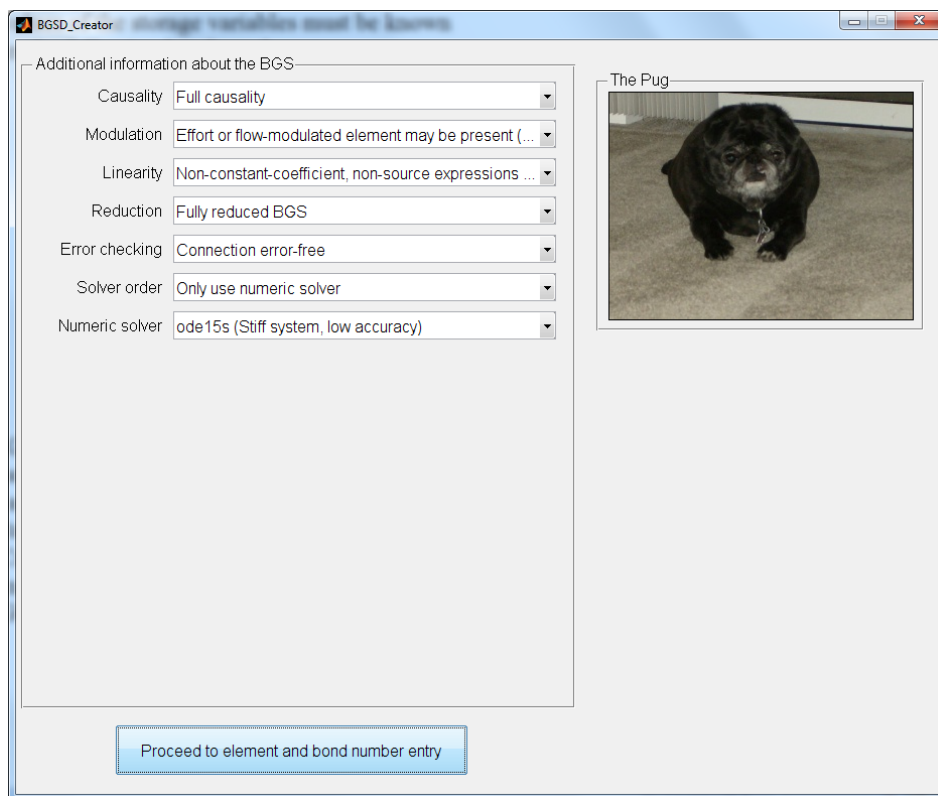
After the element types are chosen, the constituent expressions for the elements can be entered. At this point, the GUI goes through every element chosen previously, and asks the user for the expression type, causality (if applicable), and expression itself. In this case, all elements

have constant coefficient expressions, so the expression entry screens are very simple. A sample expression entry screen for the resistive element is shown in Figure 21 below.

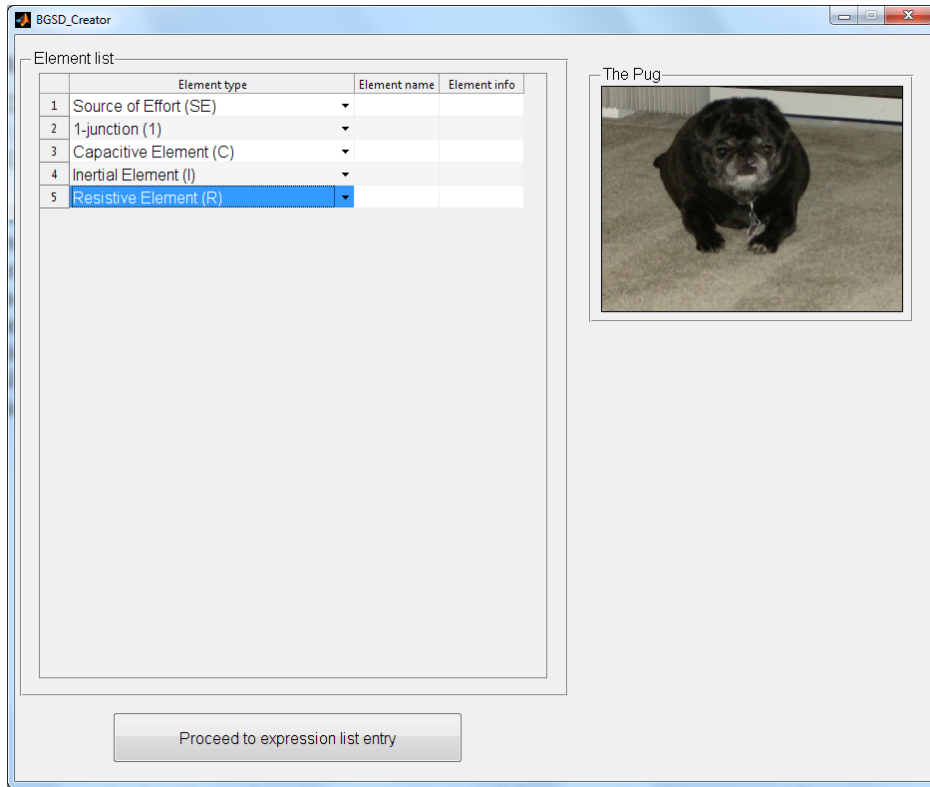
After the expressions are entered, the bond directionality screen comes up. In this screen, the user can enter the connections that the bonds in the BGS make. The filled-out bond directionality entry screen for the BGS in Figure 18 is shown in Figure 22 below.

After the bond directionality is specified, the initial condition table is provided. After entering the initial conditions, the user can save the BGSD file.

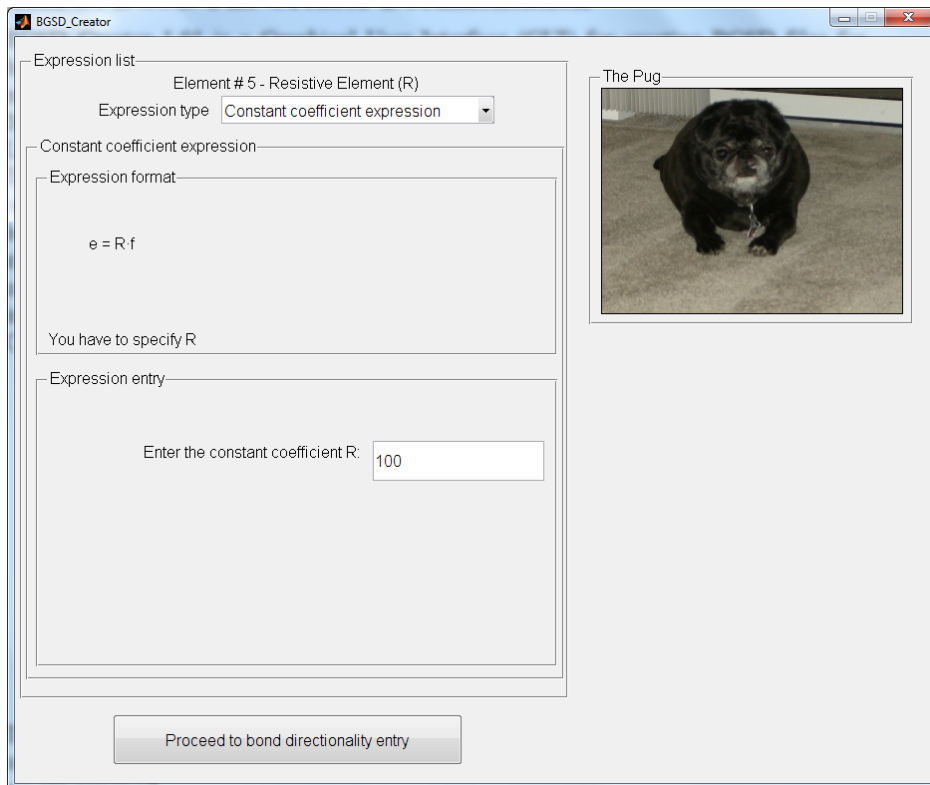
Clearly, large BGSD files cannot be created using the BGSD\_Creator GUI. However, it was used successfully for testing the BGSolver code on small problems, which is its primary purpose. The code may also prove to be convenient for modeling discrete dynamic systems using the bond graph processing package developed in this project, because discrete dynamic systems generally have much less elements and bonds than discretized coupled systems do.



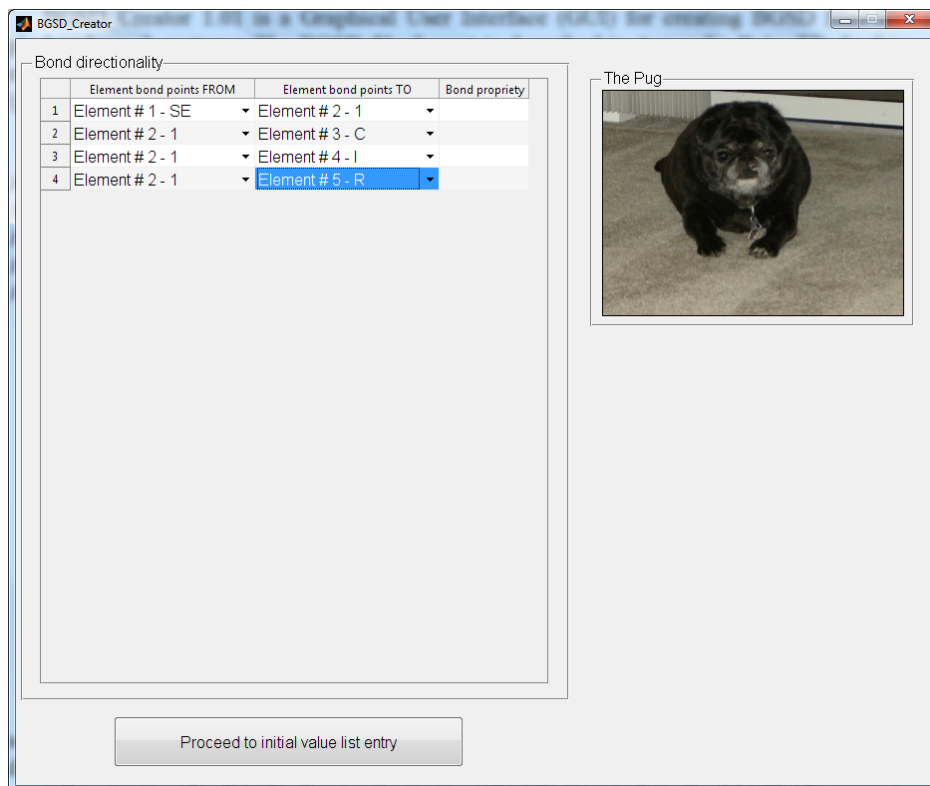
**Figure 19. BGSD\_Creator Additional Information Specification Screen**



**Figure 20. BGSD\_Creator Element Type Selection Screen**



**Figure 21. BGSD\_Creator Expression Entry Screen**



**Figure 22. BGSD\_Creator Bond Directionality Entry Screen**

## Glossary

This chapter includes a list of definitions of various terms used in this text, which may be helpful to the reader.

### Acausal bond graph system

Adapted from Ref. [19]:

An acausal bond graph system is a bond graph system in which no bonds have been assigned causality.

### Augmented bond graph system

Adapted from Ref. [19]:

An augmented bond graph system is a bond graph system in which every bond has been assigned causality.

### BGSD\_Creator

BGSD\_Creator stands for Bond Graph System Descriptor Creator. It is a simple MATLAB graphical user interface, developed to create simple BGSD files for testing the *BGSolver* bond graph processing code and new physical models.

### BGSolver

BGSolver stands for Bond Graph Solver. It is a MATLAB bond graph processing code package which utilizes the MATLAB symbolic toolbox as the equation sorting engine, formulates the ODEs using the sorted equations and integrates the ODEs using a specified numerical method. BGSolver was developed as part of this project, and used as a proof-of-concept test of the applicability of the bond graph method to coupled nuclear reactor modeling.

### Bond

In this text: a scalar bond connects two bond graph elements, pointing from one element to the other. Bonds are typically numbered, for convenience. A full bond  $i$  has two variables associated with it: effort variable  $e_i$  and flow variable  $f_i$ . A signal (also known as “activated” or “active”) bond only has one of the associated variables – the other is automatically zero. Depending on the bond’s causality, the flow variable is carried one way across the bond, and the effort variable – the other way.

In most bond graph modeling texts (for example, Refs. [19,20]) bonds carry power, with the direction the bond points in being the direction of positive power flow, and the power carried across the bond  $i$  being given by:

$$P_i = e_i f_i \tag{G.1}$$

in which:

$P_i$	Power carried by bond $i$
$e_i$	Effort on bond $i$
$f_i$	Flow on bond $i$

In this text, bonds are not restricted to carrying power, and thus the Eq. (G.1) does not necessarily hold.

A vector bond is a collection of scalar bonds pointing from one field (“tensor”) element to another field element in the same physical domain. Examples of vector bonds include

convection bonds (Ref. [26]) and 3D mechanical bonds (Ref. [40]). Vector bonds are not used in this text.

### **Bond graph element**

Adapted from Ref. [19]:

In this text: a bond graph element is a constituent object of a bond graph system which has ports to which bonds connect, and exactly one constituent equation per port. The number of ports and the forms the constituent equations take depend on the element type. The constituent equations take as inputs the causal inputs to the element (variables delivered by the bonds connected to the element's ports), and set the outputs delivered by the same bonds to the elements on the other sides of the bonds.

### **Bond graph system**

In this text: a mathematical structure consisting of bond graph elements connected with bonds. A bond graph system typically models an engineering system, with the mathematical model (equations describing the system) derivable from the bond graph system.

In many texts (for example, Refs. [19,20,25]), the term “bond graph” is used instead of “bond graph system.”

### **Bond variables**

In this text: efforts and flows in a bond graph system are referred to as “bond variables.”

### **Causality**

Adapted from Ref. [19]:

In this text: the causality is a property of a bond, determining which way the bond delivers effort. By extension, the bond delivers flow the opposite way. Causality is assigned to bonds during the system augmentation.

### **Confusion coefficient**

The name proposed in this text for the reciprocal of the neutron diffusion coefficient. Symbol  $Co$  is used for the confusion coefficient:

$$Co(\vec{x}, T) \equiv \frac{1}{D(\vec{x}, T)} \quad (G.2)$$

### **Convection bond**

Adapted from Ref. [26]:

A *vector bond* with two efforts and one flow, representing a thermodynamic energy and mass transfer by a fluid. Convection bonds are primarily used in lumped thermodynamic analysis. In this text, simpler pseudo-bonds are used instead.

### **Convergence plot**

Adapted from Ref. [35]:

A convergence plot is a log-log plot of the error of a numeric scheme versus the time, or spatial step size used by the scheme. If the scheme exhibits asymptotic convergence, the plot should look like a straight line, whose slope is the observed order of accuracy (also known as order of convergence) of the scheme.

## Coupled phenomena

In this text, the mathematical definition of coupled phenomena is used.

Adapted from Ref. [36]:

Physical phenomena  $x$  and  $y$  are considered uncoupled if changing the properties or the dynamics of phenomenon  $x$  does not affect the properties and the dynamics of phenomenon  $y$  and vice versa.

Physical phenomena  $x$  and  $y$  are considered triangularly coupled if changing the properties or the dynamics of phenomenon  $x$  affects the properties or the dynamics of phenomenon  $y$ , but not vice versa.

Physical phenomena  $x$  and  $y$  are considered fully coupled if changing the properties or the dynamics of either of the phenomena affects the properties or the dynamics of the other phenomenon.

Simple linear uncoupled, triangularly coupled and fully coupled dynamic models are illustrated in Eq. (G.3):

$$\underbrace{\frac{\partial}{\partial t} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}}_{\text{Uncoupled (diagonalized) phenomena}} \quad \underbrace{\frac{\partial}{\partial t} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}}_{\text{Triangularly coupled phenomena}} \quad \underbrace{\frac{\partial}{\partial t} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}}_{\text{Fully coupled phenomena}} \quad (\text{G.3})$$

## Internal energy

From Ref. [13]:

Strictly, the internal energy of a system is the total energy due to the kinetic energy of its particles and the potential energy associated with the crystalline structure of the materials, as well as the chemical and nuclear bonds of the particles in the system. This makes the internal energy of a system the combination of sensible, latent, chemical and nuclear energies of the system.

Generally in thermodynamic and heat transfer analysis of a system, only the sensible and latent energies, known together as the *thermal energy*, are considered. For this reason, in this text, internal energy refers only to the thermal energy.

## Mathematical model

Adapted from Ref. [19]:

In this text: a mathematical model of a given engineering system is a system of partial differential and integro-differential, ordinary differential and algebraic equations, the solution for which describes the engineering system's dynamic (for dynamic models) or static (for steady-state models) behavior. The accuracy of the description is dependent upon the assumptions made when constructing the models, the numerical accuracy of the solution, the quality of physical data available for the system's properties, and other factors.

## Mechatronic system

In this text: a mechatronic system is any engineering system that includes, at least, mechanical (translational and/or rotational) and electrical components.

## Method of manufactured solutions

The method of manufactured solutions is a technique for benchmarking proof-of-concept codes, which consists of several steps:

1. Write out the system of the fundamental partial differential equations that the code is trying to solve, along with some set of material and geometric properties describing the system.
2. Pick an arbitrary but physically reasonable smooth solution function for each PDE in the system.
3. Plug the smooth “manufactured” solutions from step 2 into each PDE.
4. Solve for the “corrective” external source functions in each PDE.

The new problem now consists of the material and geometric properties from step 1 with the corrective external source functions from step 4. The exact solutions to this new problem are the manufactured solutions from step 3. Having exact solutions to the problem, obtained without the use of codes, allows for a much more in-depth analysis of the code and convergence studies than using fine-mesh solutions and/or other higher performance codes.

### **Modulated element**

Adapted from Ref. [19]:

In this text: an element whose constituent equation(s) include not only the variables delivered to it by its full bonds, but also time (for time-modulated variables), and bond variables on bonds other than the bonds connected to the element. These additional (modulating) variables are delivered to the modulated element via signal bonds.

Modulated elements have “M” in front of their names: for example, the symbol of a Modulated Source of Effort is MSE.

### **Numeric layer**

During the sorting procedure (step 3 of the bond graph method), the algebraic equations (AEs) are solved and the bond variable vector is found. The bond variable vector is a function of time, state vector and, if they are present, one or more numeric variables. The numeric variables are the symbolic variables which serve as the symbolic placeholders for the MATLAB functions in the constituent expressions of the elements in the *bond graph system* being processed. The bond variables and the numeric variables are then separated between several numeric layers, with the following structure:

- To evaluate bond variables in layer 1, no numeric variables need to be known.
- To evaluate numeric variables in layer 1, only the bond variables in layer 1 need to be known.
- To evaluate bond variables in layer  $i > 1$ , only numeric variables in layers  $1 \leq j < i$  need to be known.
- To evaluate numeric variables in layer  $i > 1$ , only bond variables in layers  $1 \leq j \leq i$  need to be known.

MATLAB functions cannot be directly manipulated by the symbolic math toolbox. However, this structure allows using the symbolic sorter to solve the AEs formed in step 2, even if some of the AEs may involve numeric expressions with underlying MATLAB functions.

### **Physical domain**

In this text: part of a system governed by a certain unique set of physical laws, which govern specific behavior. Geometrically, several physical domains may be present in a region of a system. Examples of physical domains include:

- Translational mechanics of a body
- Rotational mechanics of a body
- Electrical circuits and networks



- Hydraulic and acoustic networks
- Magnetic fields
- Neutron transport

In Ref. [19], the term “*energy domain*” is used in place of “physical domain,” because in most physical domains, energy is a conserved quantity. Therefore, the way the energy is stored and power transferred through the system is dependent on the physical domain the energy is in. In this text, several domains are considered where conservation of energy is not part of the governing laws (i.e., neutron transport), and therefore the term “physical domain” is suggested.

### **Reticulation**

Adapted from Ref. [20]:

Reticulation of a system is the process of decomposing the system into interacting component parts. The result of this decomposition can also be referred to as the reticulation. Reticulation involves making simplifying assumptions about both the nature of the component parts and about their interactions. Generally, a good reticulation of an engineering system is not unique, nor are dynamically equivalent reticulations necessarily identical in their structure and complexity.

### **Signal bond**

See *bond*.

### **System**

Quoted verbatim from Ref. [19]:

1. A system is assumed to be an entity separable from the rest of the universe (the environment of the system) by means of a physical or conceptual boundary. An animal, for example, can be thought of as a system that reacts to its environment (the temperature of the air, for example) and that interchanges energy and information with its environment. In this case the boundary is physical or spatial. An air traffic control system, however, is a complex, man-made system, the environment of which is not only the physical surroundings but also the fluctuating demands for air traffic, which ultimately come from human decisions about travel and the shipping of goods. The unifying element in these two disparate systems is the ability to decide what belongs in the system and what represents an external disturbance or command originating from outside the system.
2. A system is composed of interacting parts. In an animal we recognize organs with specific functions, nerves that transmit information, and so on. The air traffic control system is composed of people and machines with communication links between them. Clearly, the *reticulation* of a system into its component parts is something that requires skill and art, since most systems could be broken up into so many parts that any analysis would be swamped with largely irrelevant detail.

### **Thermal energy**

From Ref. [13]:

The combination of sensible and latent energies of the system. If no chemical or nuclear reactions occur, all changes in the internal energy of the systems are due to changes in the thermal energy of the system. Only the sensible energy of the system is dependent on the system’s thermodynamic state (temperature and pressure). For this reason, in this text, the terms *internal energy* and thermal energy are used interchangeably.

**Vector bond**

See *bond*.

## References

1. D. A. Knoll and D. E. Keyes, "Jacobian-free Newton-Krylov methods: a survey of approaches and applications," *Journal of Computational Physics* **193** (2), 357-397 (2004).
2. H. M. Paynter, *Analysis and Design of Engineering Systems*. (The MIT Press, Cambridge, MA, 1961).
3. J. J. Duderstadt and L. J. Hamilton, *Nuclear Reactor Analysis*. (John Wiley & Sons, Inc., Hoboken, NJ, 1976).
4. A. F. Henry, *Nuclear-Reactor Analysis*. (The MIT Press, Cambridge, MA, 1975).
5. W. M. Stacey, *Nuclear Reactor Physics*, 2<sup>nd</sup> ed. (Wiley-VCH Verlag GmbH & Co. KGaA, Weinheim, Germany, 2007).
6. S. M. Ghiaasiaan, *Two-Phase Flow, Boiling and Condensation in Conventional and Miniature Systems*. (Cambridge University Press, Cambridge, UK, 2008).
7. P. K. Kundu and I. M. Cohen, *Fluid Mechanics*, 4<sup>th</sup> ed. (Academic Press, Burlington, MA, 2008).
8. N. E. Todreas and M. S. Kazimi, *Nuclear Systems*, Vol. 1. (Taylor & Francis Group, LLC, New York, NY, 1990).
9. R. L. Norton, *Machine Design – An Integrated Approach*, 3<sup>rd</sup> ed. (Prentice Hall, Upper Saddle River, NJ, 2006).
10. ASM International Handbook Committee, *ASM Handbook*, Vol. 13C, 10<sup>th</sup> ed. (ASM International, Materials Park, OH, 2007).
11. I. Gouja, M. Avramova and A. Rubin, "Development and Optimization of Coupling Interfaces Between Reactor Core Neutronics and Thermal-Hydraulic Codes," presented at the PHYSOR 2010 Conference, Pittsburgh, PA, 2010.
12. M. J. Moran and H. N. Shapiro, *Fundamentals of Engineering Thermodynamics*, 5<sup>th</sup> ed. (John Wiley & Sons, Inc., Hoboken, NJ, 2004).
13. F. P. Incropera, D. P. DeWitt, et al., *Fundamentals of Heat and Mass Transfer*, 6<sup>th</sup> ed. (John Wiley & Sons, Inc., Hoboken, NJ, 2007).
14. K. S. Krane, *Introductory Nuclear Physics*. (John Wiley & Sons, Inc., Hoboken, NJ, 1988).
15. K. Shibata, T. Kawano, et al., "Japanese Evaluated Nuclear Data Library Version 3 Revision-3: JENDL-3.3," *Journal of Nuclear Science and Technology* **39** (11), 1125-1136 (2002).
16. R. L. Liboff, *Introductory Quantum Mechanics*, 4<sup>th</sup> ed. (Addison Wesley, San Francisco, CA, 2003).
17. M. Clergeau, F. Dubois, et al., "HEMERA: A 3D Computational Tool for Analysis of Accidental Transients," presented at the PHYSOR 2010 Conference, Pittsburgh, PA, 2010.
18. "NEACRP 3-D LWR Core Transient Benchmark." H. Finnemann and A. Galati, OECD Nuclear Energy Agency, (1992).
19. D. C. Karnopp, D. L. Margolis and R. C. Rosenberg, *System Dynamics: Modeling and Simulation of Mechatronic Systems*, 4<sup>th</sup> ed. (John Wiley & Sons, Inc., Hoboken, NJ, 2006).
20. F. T. Brown, *Engineering System Dynamics: A Unified Graph-Centered Approach*, 2<sup>nd</sup> ed. (CRC Press, Boca Raton, FL, 2007).
21. A. R. Hambley, *Electrical Engineering: Principles and Applications*, 3<sup>rd</sup> ed. (Prentice Hall, Upper Saddle River, NJ, 2005).
22. D. C. Karnopp, "Pseudo Bond Graphs for Thermal Energy Transport," *Journal of Dynamic Systems, Measurement and Control*, Transactions of the ASME, Series G **100** (3), 165-169 (1978).

23. D. C. Karnopp, "A Bond Graph Modeling Philosophy for Thermofluid Systems," *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME, Series G* **100** (1), 70-75 (1978).
24. D. C. Karnopp, "State Variables and Pseudo Bond Graphs for Compressible Thermofluid Systems," *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME, Series G* **101** (3), 201-204 (1979).
25. F. E. Cellier, *Continuous System Modeling*. (Springer-Verlag, New York, NY, 1991).
26. F. T. Brown, "Convection Bonds and Bond Graphs," *Journal of the Franklin Institute* **328** (5-6), 871-886 (1991).
27. P. J. Gawthrop and L. Smith, "Causal Augmentation of Bond Graphs with Algebraic Loops," *Journal of the Franklin Institute* **329** (2), 291-303 (1992).
28. R. C. Rosenberg, "State-Space Formulation for Bond Graph Models of Multiport Systems," *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME, Series G* **93** (1), 35-40 (1971).
29. J. D. Lamb, D. R. Woodall and G. M. Asher, "Bond Graphs I: Acausal Equivalence," *Discrete Applied Mathematics* **72**, 261-293 (1997).
30. J. D. Lamb, D. R. Woodall and G. M. Asher, "Bond Graphs II: Causality and Singularity," *Discrete Applied Mathematics* **73**, 143-173 (1997).
31. J. D. Lamb, G. M. Asher and D. R. Woodall, "Bond Graphs III: Bond Graphs and Electrical Networks," *Discrete Applied Mathematics* **73**, 211-250 (1997).
32. E. O. Kreyszig, *Advanced Engineering Mathematics*, 9<sup>th</sup> ed. (John Wiley & Sons, Inc., Hoboken, NJ, 2006).
33. R. J. LeVeque, *Finite Volume Methods for Hyperbolic Problems*. (Cambridge University Press, Cambridge, UK, 2002).
34. D. Varberg, E. J. Purcell and S. E. Rigdon, *Calculus*, 8<sup>th</sup> ed. (Prentice Hall, Upper Saddle River, NJ, 2000).
35. R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time Dependent Problems*. (Society for Industrial and Applied Mathematics, Philadelphia, PA, 2007).
36. G. Strang, *Computational Science and Engineering*. (Wellesley-Cambridge Press, Wellesley, MA, 2007).
37. The MathWorks, "MATLAB – Symbolic Math Toolbox." (2010). Available at <http://www.mathworks.com/products/symbolic/>. (Accessed on 4/25/2010).
38. Maplesoft, "Maple 14 – Math & Engineering Software." (2010). Available at <http://www.maplesoft.com/products/Maple/index.aspx>. (Accessed on 4/25/2010).
39. Sandia National Laboratories, "The Trilinos Project." (2009). Available at <http://trilinos.sandia.gov/>. (Accessed on 4/23/2009).
40. E. P. Fahrenthold and J. D. Wargo, "Vector and Tensor Based Bond Graphs for Physical Systems Modeling," *Journal of the Franklin Institute* **328** (5-6), 833-853 (1991).