THE MOVING OBJECT SIMULATION ENVIRONMENT: A GENERIC FRAMEWORK FOR BUILDING TRANSIENT MODELS

By

Dave Gilbert

B.Sc., 2nd B.Sc.H., M.Sc.

A Thesis Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements for the Degree

Doctor of Philosophy

McMaster University

Your file Votre référence
ISBN: 978-0-494-36027-9
Our file Notre référence
ISBN: 978-0-494-36027-9

THE MOVING OBJECT SIMULATION ENVIRONMENT: A GENERIC FRAMEWORK FOR BUILDING TRANSIENT MODELS

DOCTOR OF PHILOSOPHY (2007) McMaster University

(Electrical and Computer Engineering) Hamilton, Ontario

TITLE: The Moving Object Simulation Environment: A Generic Framework for Building Transient Models

Author: Dave Gilbert

B.Sc. (Trent University)

2nd B.Sc.H. (McMaster University)

M.Sc. (McMaster University)

ADVISORS Dr. W.F.S. Poehlman, Dr. Wm. J. Garland

NUMBER OF PAGES xiii, 303

Abstract

A framework called the MOOSE (MOving Object Simulation Environment) has been developed for modeling moving components in the presence of diffusion phenomena. The framework focuses on general elliptic and parabolic problems which can be represented on a two dimensional patched Cartesian grid. The idea of a problem solving environment is presented and the MOOSE is developed within this conceptual paradigm using several novel implementation techniques. Code generation, symbolic computation, and high performance spectral solvers are joined within a flexible and unified tool that implements a mesh linking algorithm capable of minimizing errors induced by moving components in close proximity to material discontinuities.

The MOOSE constructs linear finite difference models based on symbolic mathematical descriptions supplied by the model designer. Solutions are computed by transforming abstract descriptions into matrix notation compatible with a collection of high performance parallel linear and eigenvalue solvers. Design techniques are presented for the implementation of a patched non-conformal mesh that links groups of sub-meshes, which can move relative to one another. The generation of a sequence of matrices which model dynamic components using moving meshes that conserve flow at their boundaries, and the performance of the framework when applied to a variety of test cases is discussed.

A major case study based on the 1994 reactivity insertion incident which occurred at the McMaster Nuclear Reactor is undertaken. The flexibility, precision, and robustness of the MOOSE framework and algorithms are exercised by this study. The results from the original tech report are verified for higher dimensional cases.

The MOOSE uses techniques that are mathematically simpler than previously accepted non-linear nodal methods used in nuclear engineering, but still capable of

easily representing moving components. A concise ruleset for linking moving meshes is presented which is demonstrated by the framework. Error reductions of several orders of magnitude are demonstrated by the MOOSE's multi-resolution moving mesh algorithm over more costly brute force strategies.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Problem Solving Environments (PSEs)

Engineering design as an activity was originally the domain of scale models and mathematical paper approximations that gave good intuition to system designers as to how a new machine would perform. In the last 50 years computers have taken an increasing role in the engineering design process, to the point where today the subject of computational science and engineering merits treatment on its own. Computational science and engineering is often thought of as a discipline positioned between theoretical and experimental areas of science and engineering [134, 135, 136]. Relevant areas of engineering [52, 89] and science include, but are not limited to, fluid mechanics, thermodynamics, electromagnetic phenomena, nuclear reactor simulation, weather forecasting, and aircraft design.

As simulation costs decrease and physical prototype costs increase there is more pressure to accept simulation results. As a consequence both model and software validation issues are becoming more important. Validation of most simulation systems is often inadequate. The three principle sources of simulation error are: incorrect math-

ematical models, inaccurate numerical approximations, and incorrectly constructed software systems. Comparison with known results is the best form of validation. However, for problems addressed by some software systems there are no known results, for example the next earthquake to hit California, or the next great fire in Chicago.

The state of the art in computer hardware provides the engineer who is interested in simulation with powerful tools at low cost. The current difficulty lies in taking advantage of these tools. In the last 20 years mathematical libraries have provided an excellent model for software reuse. Many people use commercial and government sponsored libraries such as NAG, IMSL, and LAPACK [52]. Libraries in themselves do not entirely solve the problem of model construction since any given library assumes a certain amount of expertise on the part of the user. The user must still convert the problem into the generic mathematical language of the numerical solver, and the user must understand that language so the correct algorithms can be selected from the library.

The purpose of a Problem Solving Environment (PSE) is to automate the process of model construction by creating a reusable tool for a domain of problems. The classical simulation design and construction process is unified in a single tool that encapsulates expertise from a variety of domains. A PSE collects together several solution methods and models, addressing issues such as appropriate software reuse, intrinsic model validation, and intelligent algorithm selection.

The ideal PSE is an abstract concept which researchers in the area are still striving to achieve. PSEs are often limited in their generality, their performance, the solution algorithms that they implement, the resolution of the models that they compute and the degree to which the problem solving process is automated.

- PSE should be able to handle:
  - Problem specification- typically includes physical model and geometric model
  - Solution specification- facilitates the choice of algorithms and solution strategies
  - Model compilation and execution
    * Sometimes also includes solution steering, and progress monitoring
    * Data check-pointing- calculation may require days or weeks,
  - Output rendering and analysis, typically graphical

PSE research must somehow address the shortcomings in the current state of the art. In the last 15 years several workshops and discussion sessions have focused on PSEs. To supplement the recent conferences a book edited by Houstis et. al [89] was published which collects 28 articles discussing several important aspects of PSEs along with a comprehensive bibliography with over 400 entries.

Some PSE research horizons are discussed by John Rice in two separate articles [135, 136], where he describes *multi-physics phenomena* and *multi-scale phenomena.* According to Rice, multi-physics phenomena involve two or more separate physical regimes. An example might be heating a pot of water, the heat source being one system, the fluid dynamics of the water would represent the other. Multi-physics phenomena might be spatially and temporally superimposed, or might be separate. The interfaces between the phenomena present a variety of challenges. It might be difficult to obtain information about the interfaces, or in some cases there may be no known valid models which describe the interfaces. Friction, for example, affects all sorts of applications and is especially relevant for describing losses that occur at an

interface. Despite this, there is no reliable consistent model for friction effects which is universally applicable.

Multi-scale phenomena involve vastly different time and space scales. For example a jet engine is several meters long, yet its fuel-spray droplets are 5 to 10 microns. Its blades are tens of centimeters long, but cracks form in areas of tens of angstroms in size. Since it is impossible to model the entire system at the resolution of microns the key issue becomes defining a consistent way to simulate all of these phenomena simultaneously. The current approach is to use models of different scales and use a special approximation that links the fine scale model to the coarse scale model. This method is problem specific and prone to a variety of errors.

The predicted growth of computational power and network bandwidth suggests that computational modeling will shift from focusing on a single component design to the design of an entire system. The analysis of an engine involves the domains of thermodynamics (gives the behavior of the gases in the piston-cylinder assemblies), mechanics (gives the kinematic and dynamic behavior of pistons, links, cranks, and so forth), structures (gives the stresses and strains on the parts) and geometry (gives the shape of the components and the structural constraints). The design of an engine requires that these different domain specific analyses interact to find the final solution [18, 88].

The goal of the PSE is to solve all of these problems in an elegant efficient package. Bringing together the expertise of researchers from a variety of areas and concentrating it into a single tool represents the task faced by the PSE designer. The state of the art in simulation design and implementation today is very much like the state of the art in type setting and publishing 100 years ago. In an analogous way, simulation researchers are assembling thoughts from movable foundry type, transferred from a distribution box to a composition stick, and laboriously mounted into a press to produce several

hundred copies of a single page of a large volume. The PSE designer foresees the future of simulation in the form that desktop publishing software may have appeared centuries ago to an early printing press operator.

1.2 Addressing Cross Disciplinary Issues

Solving large engineering problems requires the collaboration of experts from a wide variety of fields. A typical engineering problem may require:

- Domain engineer- provides special understanding of problem area
- Numerical specialist- provides expertise in numerical algorithms and methods, parallel computing, and hardware considerations
- Scientific computing specialist- provides expertise in optimization, integration, and linear algebra methods
- Software implementation group- supplies the software implementation which links the various components together and the implementation of algorithms designed by other specialists

It is generally acknowledged [52, 123] that collaboration between these diverse groups is difficult to establish. While some persons may have skills that cover several areas, usually a single person cannot acquire all the knowledge required for a high performance engineering problem. Stanzione writes

> "To summarize, the heart of the problem is not that inadequate computing facilities exist to run simulations, but rather that the expertise required to create the simulation codes for the target computing resource

> is possessed by two distinct groups of people. Researchers in the application domain on one side and [High Performance Computing] experts on the other, each with too many demands from their own fields to adequately learn the others."[151]

Scientific research is inherently an act of collaborative problem solving. Merely providing access to computational resources and domain tools is not enough to facilitate or enhance scientific problem solving. PSE design is about bridging a gap between computational resources, appropriate algorithms and the people who need to use those resources.

Science has developed a standard language with many sub-dialects for different sub-fields. A PSE should use this language. Some parts of scientific language are well standardized, for example symbolic mathematics, and numerical algorithms, other parts are not. In particular computer languages for geometry tend to be primitive and inconsistent. Mesh and grid generators are used to discretize geometry but the software is complex and less than completely robust or reliable [88].

The experiment is the vehicle through which scientists and engineers attack their research. A study done by [97] identified certain practices in the design and execution of models.

- Experiments are built through defining sequential steps that utilize the model, observational data, application tools, computers and other miscellaneous resources.
- The experimentation process is highly repetitive. A cycle of steps is repeated that includes modifying the configuration and initial conditions of the computational experiment, executing the experiment, and evaluating the generated output and its convergence to observed or theorized results.

- Computational experiments require long sequences of computer operations such as logging into the system, querying for and collecting data from on-line databases and repositories, running applications on distributed computers, capturing experiment output to files, transferring data files between computers, applying translators to convert data formats, and executing analysis and visualization packages on specific data sets.

- In designing and executing experiments, practitioners typically maintain the design and execution processes in notes, maintaining a log of their activities

A PSE that captures not only the language of science and engineering but also the procedures that the scientist or engineer would follow contributes better to the problem solving process [150].

The use of appropriate abstractions is the key to mastering the language of science and engineering, and is the foundation upon which a well developed PSE must be built. The designer of a PSE is charged with the task of not only solving a difficult computational problem, but also of solving that problem in a contextually specific way that is meaningful to the practitioners of the target domain. A tool that requires the practitioner to learn many new task specific procedures for its use, or delve deeply into new areas of science and gain a deep understanding of methodologies solely for the purpose of simulation and problem solving, will be less of an aid than a tool which hides its implementation behind the terminology and procedures of the target user.

1.3 Simulating Motion

Simulation environments designed to study steady state or transient problems already have many representatives in both commercial forms, and as research projects.

FEMLAB is an excellent example of a successful commercial PSE that focuses on the application of unstructured finite element meshes to fluid mechanics, electrodynamics and a wide variety of other problem domains. An important sub-domain of transient simulations are those problems that examine phenomena in the presence of moving components. PSEs for the study of motion are not well represented. Overture [32, 33] is one example of an academic project which focuses on linking deformed overlapping structured meshes for the study of fluid flow problems and moving components, however there are very few other general projects in this area.

This thesis uses as its target problem one inspired by reactor safety analysis [4, 57, 61, 69, 107]. Historically, nuclear engineering papers have used Cartesian meshes with nonlinear approximations in very large mesh cells to model transient problems. The most popular methods are referred to as nodal methods. Nodal methods [103, 110, 124] are able to use very large geometric cells because each cell uses a complex set of nonlinear equations to estimate the neutron density distribution within that cell. Nodal methods have been reported to be an order of magnitude faster than some linear implementations of equal precision. Control rod motion is modeled with nodal methods by using special approximations [96] to estimate the behavior of the leading and trailing cell of moving assemblies. This style of solution is incompatible with existing generic tools like FEMLAB or Overture, precluding their application. Existing multi-physics PSEs do not provide a drop in replacement solution for moving assembly simulation scenarios.

Nodal methods have a variety of difficulties. The mathematics behind nodal methods tends to be dense. Simulations based on nodal methods have limited generality. A nodal solution is often only valid for a narrow range of problem parameters, and nodal methods are also often limited to a very few number of energy groups, typically two. The original nodal methods required complex calibration of linkage constants.

While modern nodal methods have resolved the problem of calibrating cell by cell linkage, it has been at the price of further mathematical complexity. Given some of the problems with nodal methods, and the absence of a generic tool for modeling moving components, provides the special focus for the development of the MOOSE framework.

In the last 15 years there have been remarkable innovations in solution techniques for linear [21] and eigenvalue [17] problems and some practical implementations of those solution techniques have been produced. Very recently, a new public domain eigenvalue solver, SLEPc [82], based on the already well established high performance parallel linear solver PETSc, has reached a state of noteworthy maturity. SLEPc's most recent release (as of Oct 2006) incorporates the Krylov-Schur method which provides reliable and fast calculations of extremal eigenvalues. SLEPc is able to solve very hard eigenvalue problems of the general form $Ax = \lambda Bx$. Although not limited to the followings cases, but of interest to practitioners of nuclear engineering, SLEPc can handle problems where A is non-symmetric, and where B is singular, and SLEPc can be configured to solve for the single smallest eigenvalue near unity. This class of problems corresponds to a general interpretation of the steady state multi-group neutron diffusion problem. While existing PSEs do take advantage of similar solvers, issues related to motion as it occurs in the context of a reactor safety analysis problem require special attention.

In particular this thesis will discuss what benefits could be realized through linking together a collection of meshes, what precise techniques are required to ensure that the linked meshes behave well, how errors associated with moving components can be measured and analyzed to establish parameters for the estimation of the correctness of the mesh, all done within the context of linear approximations so that the most recent advances in linear problem solving libraries can be taken advantage of.

The MOOSE is such a tool. Its overall design is not limited to the constraints of the nuclear engineering problem, however this problem presents a sufficient number of challenges that by studying it, the breadth of problem types that the MOOSE is capable of modeling can be illustrated. The MOOSE is able to model both transient and steady state problems characterized by elliptic and parabolic equations, and to a limited extent some hyperbolic equations. Both linear and eigenvalue problems can be specified with the MOOSE. Simulations are limited to two dimensions, and the MOOSE provides only a very rudimentary user interface. Despite these limitations, the MOOSE is still extremely flexible, and in the course of this thesis a variety of studies will demonstrate the breadth of problem types the MOOSE is able to address.

1.4 The McMaster Nuclear Reactor

The McMaster Nuclear Reactor (MNR) has a 45 year history on the McMaster campus. It has an operating budget of several million dollars per year, and it hosts one of the few (if not the only) commercially self sustaining research group on campus, deriving funding not only from research grants but also from the sale of products and services. The reactor brings together individuals from a variety of disciplines including physics, electrical engineering, computer science, mechanical engineering, and materials science as well as the interests of members from areas not typically associated with engineering like medicine and archaeology. Over a dozen scientists and engineers have dedicated themselves to the maintenance and study of the experimental Materials Testing and Research Reactor (MTRR).

Reactor modeling problems are multi-physics problems. Several physical processes need to be modeled concurrently, including fluid flow through the core, neutron flux produced within the core, and other effects like heat distribution across the core el-

ements are also important. In addition to the physical processes of the core the electrical and control mechanisms play an important part of the safety analysis. The reactor as a mechanism is not only a complex physical system which combines fissioning uranium with a heat transport model, but also links these physical phenomena through a complex system of sensors, control relays and safety circuits which must respond with 100% reliability in small fractions of seconds. Reactor components can change position both in a gross way, as in the reorganization of fuel assemblies in the core, but also in a fine way as in the repositioning of a control rod by a small fraction of a centimeter. Developing a system capable of modeling all of these processes concurrently presents a wide variety of technical difficulties at the cutting edge of computational science. This thesis will address some of these simulation models.

1.5 The MOOSE

A major subject of discussion will be the presentation of a problem solving framework that models the motion of components within a multi-physics view of engineering objects and makes use of existing scientific software components. Issues regarding solution method recommendation and knowledge bases will not be addressed. The validation of the models will be treated with specific examples, rather than the more generalized automatic validation discussed by some authors.

The MOOSE was developed with the goal of being able to examine problems that study moving components. Its particular focus is maintaining a high degree of precision without resorting to brute force tactics like using highly refined meshes or excessively small or strictly regimented time increments to model position changes in its components. As a prototype PSE, the MOOSE attempts to reduce the amount of work required for the implementation of high performance finite difference simulations.

As a high performance framework, the MOOSE addresses several open questions related to the mathematical approximations necessary for the precise solution of steady state eigenvalue problems and linear transient problems for a variety of problem areas.

This thesis provides 5 major contributions

1. a clearly defined methodology for the linking of meshes as it applies to moving components within advective and diffusion based finite difference simulations
2. detailed error analysis which address two major questions:
   (a) the extent to which using coarse meshes with special motion techniques can improve upon performance (in terms of precision and execution times) over classical linear techniques using dense meshes, or alternatively classical nonlinear techniques based on coarse meshes
   (b) whether interpolation is sufficient to connect meshes, or whether conservation techniques are required, along with several special considerations relevant to diffusion problems not before discussed in the literature
3. detailed re-examination of the estimated power peak reported in the 1997 MNR technical report
4. a prototype implementation of the MOOSE framework clearly identifying a variety of design issues and solutions to those problems using various recent techniques
5. the first highly developed nuclear application based on the Krylov-Schur method implemented within the SLEPc project and a practical examination of this solver's performance, precision, and capabilities

While the MOOSE framework is generic enough to address a broad array of problem types, its development is inspired by a particular problem, that of modeling the motion of components. Chapter 2 will present a review of literature as it relates to this thesis, a review of classical sparse matrix solution techniques, a review of time integration methods, a discussion of established mesh linking methodologies as they relate to the simulation of motion, and a general history of PSEs. Chapter 3 presents the MOOSE architecture focusing on major framework components and methodologies, including a discussion of special code generation techniques as used by the MOOSE, mathematical libraries and external packages used by the MOOSE, and details specific to the mathematics behind the MOOSE's mesh linking procedures. Chapter 4 develops a collection of simple steady state and transient models drawn from standard examples from a variety of areas and compares the solutions generated by the MOOSE with closed form solutions for the purpose of validating the MOOSE. Representatives from the three fundamental types of PDEs, parabolic, elliptic and hyperbolic equations are used to verify the implementation of the MOOSE. Chapter 5 develops a single detailed application for the study of the 1994 refueling incident at the MNR, drawing conclusions about the MOOSE's precision versus naive methods, and providing a re-examination of the maximum power reached by the core. The refueling incident is studied as both an eigenvalue problem and as a transient problem. The final chapter will discuss issues unexplored by this thesis, suggest problem areas other than neutron diffusion that could benefit from the presented techniques, as well as several proposed future projects.

Chapter 2

Literature Review

This chapter presents a review of literature related to the development of the MOOSE. The MOOSE, as a PSE framework, draws on a variety of implementation techniques. Its design was modeled on similar tools developed for various applications. The first section of this chapter presents a short review of physical models used in subsequent chapters. A classification of models as elliptic, parabolic and hyperbolic is presented followed by several details necessary for the understanding of the neutron diffusion model as it is presented in Chapter 5.

The second section of this chapter presents a summary of some of the fundamental techniques used by the MOOSE to solve linear problems and eigenvalue problems. The MOOSE is based on a collections of numerical linear packages. Linear and eigenvalue solvers are not implemented within the MOOSE. Rather than presenting the details behind the algorithms developed by other researchers this section focuses on the characteristics of the solution methods and addresses questions related to why certain methods were preferred over others rather than a discussion of how the methods are implemented. Special attention is given to iterative linear solvers. Transient integration methods are also addressed.

The third section of this chapter presents a short discussion of techniques for mapping engineering problems on to computers. Various mesh techniques are discussed and reasons for selecting one mesh design over another are addressed. One to one geometric mappings are contrasted with strategies that deal with components that are linked more abstractly in terms of input and output ports. Nuclear engineering nodal methods are presented as a specialized compromise between one to one mappings and linked component methods.

The last section of this chapter presents a review of problem solving environments described in engineering literature in the last 10 years. A fair degree of latitude for what might be considered a problem solving environment is taken, consequently a wide variety of projects are described. The breadth of representatives taken from the literature gives a good indication as to what has been accomplished to date and gives a sense as to where future research can be directed.

2.1 Problems of Interest

This section will summarize some of the physical problems that fall within the MOOSE's domain. Partial differential equations are involved in the description of virtually every physical situation where quantities vary in space and time. The field $U = U(x, y, z, t)$ used to describe these quantities must contain space and time coordinates as independent variables. The independence of each variable means that the derivatives in the equations must be partial derivatives. PDEs include phenomena as diverse as diffusion, electromagnetic waves, hydrodynamics, and quantum mechanics (Schroedinger waves). In all but the simplest cases these equations cannot be solved analytically and so numerical methods must be employed for quantitative results. In a typical numerical treatment the dependent variables (such as temperature or electrical po-

tential) are described by their values at discrete points of the independent variables (e.g. space and time). Through appropriate discretization the PDEs are reduced to a large set of difference equations. As time evolves, the changes in the field $U(x, y, z, t)$ at any one position affects the field at neighboring points.

2.1.1 Classifying PDEs

Most of the physically important PDEs are of second order and can be classified into three types: parabolic, elliptic, or hyperbolic. Roughly speaking, parabolic equations involve only a first-order derivative in one variable, but have second order derivatives in the remaining variables. Examples are the diffusion equation and the time-dependent Schroedinger equation, which are first order in time, but second order in space. Elliptic equations involve second order derivatives in each of the independent variables, each derivative having the same sign when all terms in the equation are grouped on one side. This class includes Poisson's equation for the electrostatic potential and the time-independent Schroedinger equation, both in two or more spatial variables. The hyperbolic equations involve second derivatives of opposite sign, such as the wave equation describing the vibrations of a stretched string [106]. These description are often presented more formally [109] by expressing a general 2-D time independent PDE as

$$A(x,y)\frac{\partial^2 U}{\partial x^2} + 2B(x,y)\frac{\partial^2 U}{\partial x \partial y} + C(x,y)\frac{\partial^2 U}{\partial y^2} = F\left(x, y, U, \frac{\partial U}{\partial x}, \frac{\partial U}{\partial y}\right) \tag{2.1}$$

In the special case where

$$B^2(x,y) = A(x,y)\,C(x,y) \tag{2.2}$$

for all x and y the equation is called parabolic. An example is the 1-D heat equation with $B = C = 0$

$$\frac{\partial T(x,t)}{\partial t} = \frac{k}{C\rho}\frac{\partial^2 T(x,t)}{\partial x^2} \tag{2.3}$$

where T represents the heat potential, k represents the thermal conductivity, C represents the specific heat capacity and ρ is the material density.

When $B^2 > AC$ for all x and y, the equation is called hyperbolic. An example with $B = 0$ and $AC < 0$ is the 2-D wave equation

$$\frac{\partial^2 \psi(x,y,t)}{\partial x^2} + \frac{\partial^2 \psi(x,y,t)}{\partial y^2} = \frac{1}{c^2}\frac{\partial^2 \psi(x,y,t)}{\partial t^2} \tag{2.4}$$

where ψ is the wave displacement in the media, and c is the propagation speed.

When $AC > B^2$ for all x and y, the equation is elliptic. An example is Laplace's equation

$$\frac{\partial^2 U(x,t)}{\partial x^2} + \frac{\partial^2 U(x,t)}{\partial y^2} = 0 \tag{2.5}$$

where U might represent electrical potential within a static field.

A collection of examples of these fundamental problem types and typical solutions in one and two dimensions will be presented in Chapter 4.

2.1.2 Simulation Problems of Interest to the MNR

The McMaster Nuclear Reactor is a pool type reactor used for research and isotope production purposes. It uses non pressurized light water as a moderator and coolant and enriched uranium as a fuel source. Fuel assemblies are about 1 meter in length, are expensive to acquire and expensive to dispose of. A reactor core, like the MNR's,

can load about 30 fuel assemblies at a time in a rectangular array, and consumes about 6 complete assemblies during the course of the year operating at about 2 megawatts. The reactor operates 5 days per week 16 hours per day.

The positioning of the fuel assemblies determines how efficiently the uranium fuel is used. Efficient loading of fuel leads directly to cost savings for the reactor. Cost savings can potentially be on the order of $100,000 per year for the MNR even for only a small improvement in the core arrangement. Finding an optimal core design is a non trivial problem, and is potentially intractable if approached in a brute force way [130].

The MNR uses the radiation produced by the core for both academic and commercial applications. Simulation models are essential to understand the core's behavior since there is no comprehensive way to measure the radiation density at all points in the core.

Classical reactor core studies often make assumptions about which processes to include and which processes to neglect. Usually these decisions are made to keep problems tractable by limiting the geometry, dimensionality, number of tightly coupled processes, or the time domain over which the model applies. There are no models at the MNR which combine fluid flow and neutron flux in a detailed way, and there are no transient models which capture the motion of control rods or fuel assemblies.

Current tools include CATHENA [40, 76], a numerical modeling tool designed for CANDU reactors. CATHENA analyzes a thermal hydraulic system at a coarse level, that of pipe length, valve, and grossly segmented core. CATHENA has been used mainly for steady-state calculations at MNR, e.g. temperature distributions within the fuel, clad, and coolant under operating conditions. Some transient work has been done with this code [69]. RELAP [147] and RETRAN [2, 61, 127] are related tools used in the US for power reactors.

MCNP [35, 146] is a probabilistic steady state neutron transport tool which works by tracking individual neutrons, this allows for very fine details to be modeled. Neutron transport is computed based on Monte-Carlo models, MCNP simulations are limited to steady state and require prohibitive amounts of computational resources. Material interfaces can be examined, and arbitrary core geometries can be designed. MCNP is sometimes coupled with a fuel management code such as 3DDT or REBUS.

WIMS [102, 112, 171] is a deterministic transport model used for the estimation of material cross sections. It provides a bridge between the theoretically precise but computationally expensive transport simulations and the more computationally efficient, but more approximate diffusion based simulations. WIMS solves the neutron transport equation for some given segment of geometry, for example a fuel assembly (in 1D or 2D) and then homogenizes the solution, collapsing the energy groups, so that the result is then representative of the entire cell. The results are the detailed flux and power distributions within the cell, although the main result of interest are the homogenized constants.

3DDT and REBUS are deterministic diffusion theory codes able to solve three dimensional steady state core models, and some transient models related to the burning of fuel. Since they are diffusion-theory based they can not handle detailed heterogeneity of materials.

Additional simulation codes are discussed in articles by [16, 27, 93, 94, 133, 179].

2.1.3 Diffusion Methods for Reactor Kinetics

Since the principle case study presented in Chapter 5 focuses on a reactivity insertion incident described in [69] some extra attention is devoted to the neutron diffusion equation. The neutron population at a point $\left(\vec{r}, E, \hat{\Omega}, t\right)$ is characterized by $n\left(\vec{r}, E, \hat{\Omega}, t\right) dV dE d\hat{\Omega}$, the number of neutrons at time t in volume element dV surrounding the point $\vec{r}$ and in energy band dE about E moving in direction $\hat{\Omega}$ in solid angle $d\hat{\Omega}$. The neutron energy is characterized as a velocity v when multiplied by a cross section term to compute a reaction rate. In most cases the neutron population is so large (typically, ~10^8 neutrons/cm^3) that the neutrons can be treated as a continuum. At the same time the density of neutrons is so low compared to the atomic density of the medium that neutron-neutron interactions can be ignored. While migrating in a reactor core the neutrons interact with nuclei of the core materials until they are either absorbed or leak out. The neutron-nuclear interactions are often characterized by the macroscopic cross section Σ_α which specifies the probability per unit distance of travel that a neutron will suffer a collision leading to a reaction of type α (where α could represent absorption 'a', fission 'f', scattering 's', etc.).

The neutron transport equation [53, 154] is essentially an expression of conservation for the the neutron density within an arbitrary volume V about $\hat{r}$. The rate of change of neutron density with respect to time is equal to the sum of all local sources and sinks of neutrons within a volume V.

Neutron populations can be described very precisely by the neutron transport equation as

$$\begin{aligned}\frac{\partial n}{\partial t} = & -\nabla \cdot \hat{\Omega} v n - v\Sigma_t\left(\hat{r}, E\right) n\left(\vec{r}, E, \hat{\Omega}, t\right) + \\ & \int_{4\pi} d\hat{\Omega}' \int_0^{\infty} dE' \left[v\left(E'\right) \Sigma_s\left(\hat{r}, E' \rightarrow E, \hat{\Omega}' \rightarrow \hat{\Omega}\right) n\left(\vec{r}, E', \hat{\Omega}', t\right)\right] + \\ & s\left(\vec{r}, E, \hat{\Omega}, t\right)\end{aligned} \tag{2.6}$$

Where

$-\nabla \cdot \hat{\Omega}$	neutron transport into and out of control volume
$\Sigma_t\left(\hat{r}, E\right)$	probability that a neutron will suffer a collision
$\Sigma_s\left(\hat{r}, E' \rightarrow E, \hat{\Omega}' \rightarrow \hat{\Omega}\right)$	probability that a neutron will change energy level or change direction
$s\left(\vec{r}, E, \hat{\Omega}, t\right)$	sources including fission

Although neutron transport theory provides the most exact description of the neutron behaviour in a reactor, modeling the neutron kinetics in the framework of the transport theory would be prohibitively expensive. Multi-group diffusion theory is an approximation to the neutron transport process. It has been found to be adequate for many reactor analysis problems of practical interest. Multi-group diffusion theory, while simpler than transport, theory can also present a host of difficulties. If approached in a naive way problems derived from diffusion theory can be intractable.

Multi-group neutron diffusion theory uses a variety of approximations to model the continuous terms of the neutron transport equation. Multi-group diffusion, as the name implies, makes two significant simplifications to the transport model. The first is treating the continuous energy integral term $\int_0^{\infty} dE'$ as a discrete space, usually with a small number of divisions. The second important simplification is dropping the angular dependence term $\int_{4\pi} d\hat{\Omega}'$ by assuming that scattering is for the most

part isotropic. The velocity and neutron population terms, nv, are normally lumped together and treated as a single quantity called flux symbolized as ϕ for convenience. Neutron flux is perhaps an unfortunate name since in other disciplines flux is a generic term for the transport of material from one region to another. In the case of the neutron diffusion problem the flow of neutrons from one region to another is called neutron current.

The multi-group transient neutron diffusion equation is classified as a parabolic equation and is written as

$$\frac{1}{v_g}\frac{\partial \phi_g}{\partial t} = \bigtriangledown \cdot D_g \bigtriangledown \phi_g - \Sigma_{Rg}\phi_g + \sum_{g'=1,g'\neq g}^{G} \Sigma_{Sg'g}\phi_{g'} + S_g \tag{2.7}$$

Where

- g the discretized energy group #, 1 being the most energetic
- D the diffusion constant characterizing the material interaction rate
- Σ_R the removal cross section
- $\Sigma_{Sg'g}$ scattering cross section from group g' to g
- S_g source term

The source term S_g is composed of several additional terms including both prompt neutrons which are the result of a fission event and delayed neutrons which appear with an appreciable delay from the decay of certain fission products.

The source term can be written as

$$S_g = (1-\beta)\chi_g \sum_{g'=1}^{G} \nu\Sigma_{fg'}\phi_{g'} + \chi_g^D \sum_{i=1}^{6} \lambda_i C_i + s_g \tag{2.8}$$

$$\frac{\partial C_i}{\partial t} = \lambda_i + \beta_i \sum_{g'=1}^{G} \nu \Sigma_{fg'} \phi_i \tag{2.9}$$

where

- χ the prompt fission spectrum
- Σ_f the fission cross section
- χ^D the delayed neutron fission spectrum
- λ the decay rate of the precursor group
- C the delayed precursor concentration
- β the relative yield of each delayed precursor group
- s_g source term independent of the fission process

The transient problem can be transformed into an eigenvalue problem by setting the derivative with respect to time, $\frac{\partial \phi_g}{\partial t}$, equal to zero. This version is classified as an elliptic problem and is written as

$$- \bigtriangledown \cdot D_g \bigtriangledown \phi_g + \Sigma_{Rg} \phi_g - \sum_{g'=1, g' \neq g}^{G} \Sigma_{Sg'g} \phi_{g'} = \frac{1}{k} \chi_g \sum_{g'=1}^{G} \nu \Sigma_{fg'} \phi_{g'} \tag{2.10}$$

where $\lambda = \frac{1}{k}$ is the eigenvalue of the system. For the steady state case all the higher moments decay and only the first fundamental mode remains. Designing a reactor which maintains a steady state is a non-trivial task. Under long time behaviour there may be multiple eigenpair solutions. Higher frequency solutions to the flux shape often correspond with larger eigenvalues, and decay rapidly in time. These values of λ_n are known as the time eigenvalues of the equation, since they characterize the time decay. Solutions to the transient neutron diffusion equation are always dominated by some exponential terms. Transient solutions tend to grow very rapidly or decay very

quickly.

Few group diffusion equations (of 2 - 8 energy groups for thermal reactors and 15 - 20 for fast reactors) with six precursor equations are often considered to be an adequate model of the neutron kinetics in a nuclear reactor. In order to solve for the neutron group fluxes in space and time, the system of PDEs for the group fluxes and precursor concentrations must be discretized in space and time.

Finite difference methods are the simplest and most direct approach to the solution of any space-time problems. The method consists of replacing the spatial derivative in the neutron kinetics equation by the corresponding finite difference approximation. The reactor core volume is partitioned into a number of sub-regions. In each region, the material properties are spatially averaged and hence are assumed to be uniform. Either cell centered or vertex centered discretizations can be used. Cell-centered discretizations define the unknowns (group fluxes and precursor concentrations) within a typically square region which is used as the basis of the integration volume.

Neutron diffusion problems are discussed again at the end of Chapter 4 and throughout Chapter 5. This presentation of the neutron diffusion problem omits many details. A rigorous derivation of the neutron diffusion equation from the transport equation, as well as a discussion of the delayed precursor effects on transients, and a variety of closed form analytical problems are presented in the classical text by Duderstadt and Hamilton [53].

2.2 Mathematical Techniques

The MOOSE is implemented so that the model designer retains control of a variety of model design details. This section will discuss some of the background behind the mathematical tools built into the MOOSE and will also discuss some of the discretiza-

tion and integration techniques which the MOOSE makes available to the user.

2.2.1 Iterative Spectral Solution Techniques

A few central algorithms are presented in the following sections however the focus is placed on a discussion of the properties of the methods and their performance characteristics. None of these methods are implemented within the MOOSE, the MOOSE instead relies on third party implementations of linear solvers. The rationale behind why certain methods are preferred to others is the focus of this section. The two most important characteristics for selecting a method were whether it was able to solve matrix forms which corresponded with the problems focused on in the case study, and whether the implementation was sufficiently efficient. The implementation of these methods are described in [21, 141].

Iterative solution methods are commonly used for the solution of systems of PDEs for a variety of reasons. Usually they have significantly less memory overhead than their direct counter parts, and for some cases they can be very easy to implement. In situations where iterative methods are being used to solve a sequence of similar problems the solution from the most recent problem can be used as the preconditioner for the next problem.

Systems are categorized by general properties like symmetry, positive definiteness, condition number, and size. Solution methods are constrained by available machine memory, desired accuracy of calculation, available time, hardware and software packages. Several solver techniques are discussed including Jacobi's method, successive over-relaxation, conjugate gradient, biconjugant gradient, biconjugant gradient stabilized and Chebyshev iterations.

A matrix is symmetric when $A^T = A$, or alternatively when $a_{ij} = a_{ji}\ \forall i, j$.

Symmetry is important and necessary for many of the simpler more efficient solvers, most notably the conjugate gradient method. Symmetric matrices have the additional advantage that only half of the matrix information needs to be stored.

A matrix is considered to be positive definite if $x^T Ax \geq 0 \ \forall x$. The conjugate gradient method attempts to minimize the value of $\left(x^{(i)} - \widehat{x}\right) A \left(x^{(i)} - \widehat{x}\right)$, where $x^{(i)}$ is the ith estimate of the exact value $\widehat{x}$, of the equation $Ax = b$. The minimum is guaranteed to exist only if A is symmetric positive definite. The vector $x^{(i)}$ is constructed from a sequence of orthogonal residual vectors as defined by the conjugate gradient algorithm. Conjugate gradient is efficient with memory because it only needs to maintain 2 vectors $x^{(i)}$ and $r^{(i)}$, the algorithm requires only the successive update of each of these.

An iterative method is stated most generally as

$$x^{(k+1)} = Bx^{(k)} + c \tag{2.11}$$

If the matrix B is convergent then the method will converge. B is a convergent matrix if and only if the spectral radius of B is less than 1, where the spectral radius is defined as $\rho(B) = max\left\{|\lambda_i|, i = 1, ..., n\right\}$ and $\lambda_1 ... \lambda_n$ are the eigenvalues of B. Since $\rho(B) \leq \|B\|$ a straightforward way to decide if B is convergent or not is to look at its row-sum or column-sum norm and see if it is less than 1. Note that if $\|B\| \geq 1$ this does not imply $\rho(B) \geq 1$, and that the B matrix may be a complex part of a black box algorithm and so may not be readily available.

For the conjugate gradient type spectral methods the spectral condition number is the main measure of the rate of convergence. The condition number of a matrix B is defined to be $\kappa_2(B) = \lambda_{max}(B)/\lambda_{min}(B)$. The number of iterations to achieve and error ϵ is proportional to $\sqrt{\kappa_2}$. Some special cases occur, for example; elliptic

second order PDEs typically give rise to coefficient matrices A with $\kappa_2(A) = O(h^{-2})$, independent of the order of the finite elements or differences used, and of the number of space dimensions in the problem. For linear systems derived from PDEs in 2D, the condition number is proportional to the number of unknowns.

If the extremal eigenvalues of the matrix are well separated the convergence increases with each iteration. The conjugate gradient algorithm tends to eliminate components of the error in the direction of eigenvectors associated with extremal eigenvalues first. After these are eliminated conjugate gradient proceeds as though these eigenvalues did not exist. The conjugate gradient algorithm is a fundamental spectral technique, the algorithm is presented in Appendix 2. Other methods like GMRES (Generalized Minimal Residual) and BiCG (Bi-Conjugate Gradient) address problems which require the solution of non-symmetric matrices.

GMRES constructs a series of residual vectors and is guaranteed to converge after n steps where n, is the number of variables in the system. GMRES must retain intermediate calculations for each step, so that as n grows large, the storage requirements become prohibitive. The usual strategy is to restart GMRES every m steps. During the restart cycle all of the accumulated information is discarded and the current estimate is used as the new starting point. Until GMRES is restarted the work associated with it grows linearly with each accumulated vector.

BiCG uses two orthogonal sequences to deal with matrices which are not symmetric. If it is applied to a symmetric positive definite system then it will converge at the same rate as conjugate gradient, although it will require twice the amount of work. Sometimes BiCG converges in an irregular way, and convergence can in fact break down for some matrices.

Chebyshev Iteration's convergence rate depends on the estimate of the extremal eigenvalues. If a good estimate is used, and the matrix is symmetric positive definite,

then Chebyshev will converge as fast as CG. If poor estimates of the eigenvalues are made then Chebyshev Iteration can converge very slowly or diverge in some cases.

Solver	Sym	Positive Definite	Iterations needed for convergence	Operations/ Iteration	memory consumption
Jacobi	No	Yes	slow $\sim$ n	$w*n$	$(w+3)*n$
SOR	No	Yes	GS/10	$(w+1)*n$	$(w+2)*n$
CG	Yes	Yes	$\sqrt{\kappa_2}$	$(w+5)*n$	$(w+6)*n$
GMRES	No	No	depends on m	$(w+2*i+2)*n$	$(w+i+5)*n$
BiCG	No	No	$\geq$2*CG, unstable	$(w+7)*n$	$(w+10)*n$
BiCGSTAB	No	No	CGS, stable	$(2*w+10)*n$	$(w+10)*n$
Chebyshev It.	No	No	CG $\rightarrow$eigen est.	$(w+1)*n+\lambda_{est}$	$(w+5)*n$

Table 2.1: A Summary of Spectral Method Performance Characteristics

The number of iterations to converge is a function of which algorithm, or mathematical method is used to solve a system. The cost of each iteration of that algorithm is dependent on precisely how the algorithm is implemented, (i.e. how many additions/ multiplications are required). The performance of several iterative methods is summarized in Table 2.1, this information is taken from [21].

All of the iterative methods involve vector sums, scalar-vector products, inner products, vector-matrix products, but no matrix matrix products. SAXPY is an operation where a scalar vector product is computed along side a vector sum as $z = \alpha x + y$.

Memory consumption is related to the number of intermediate vectors that the method stores. Each method must store the entire sparse matrix, so usually the memory consumption required to do this will dominate the total memory consumption. Matrix storage is presented in Table 2.1 as the row width times the number of elements $w * n$. There are a variety of storage schemes for sparse matrices so this figure is given as an estimate.

The linear multi-grid method [181] can be an extremely fast solution technique.

Its implementation diverges quite radically from the previously described iterative methods because it requires multiple problem representations. Although not used in this thesis, it has been used very recently by [118], and so is described in Appendix 2.

Multi-grid methods should not be confused with composite grid techniques presented in the subsequent sections. The multi-grid method is an iterative solution technique which provides excellent performance and good parallelization characteristics for elliptic problems. In contrast composite grid techniques are solely for the purpose of building an appropriate spatial discretization for a problem. Composite grid techniques can be used to focus computational resources on a certain segment of the geometry, or they can allow sections of the geometry to move relative to one another as will be the main focus in later chapters. Composite grid techniques are not tied to any particular solver, likewise multi-grid methods do not necessarily imply any particular geometric interpretation of a problem. In principle the two techniques can be combined, although this was not attempted as part of this thesis.

The GMRES method was the preferred iterative solver used in later chapters of this thesis for its good convergence rate and reasonable memory consumption. The problem formulations constructed by the MOOSE are often not symmetric, so this prevented the use of the conjugate gradient algorithm.

Other Techniques

There are a wide variety of important linear solution techniques which go beyond the scope of this chapter.

Sparse direct methods are able to solve linear systems without resorting to iterative techniques. Sparse direct methods have properties similar to standard dense matrix solution techniques like Gaussian elimination or LU decomposition while still taking

advantage of matrix representations which do not explicitly represent all of the zeros in the system. They generate solutions in a fixed number of steps, which in some cases are more precise than their iterative counterparts. Sparse direct methods have the disadvantage that they tend to consume more memory than iterative methods and they can also be very difficult to implement. Recent implementations of direct methods are discussed by [10, 48].

Parallelization of iterative techniques is accomplished by segmenting solution vectors either by the solution vector's indices, or through partitioning which is optimized based on the geometric references that each entry makes. As already mentioned the Jacobi method requires many thousands of iterations to achieve the same accuracy on a well conditioned matrix as the conjugate gradient method, but each iteration is very cheap, and parallelization of the Jacobi method is trivial, whereas parallelization of the conjugate gradient method is not trivial. The selection criteria which is used to choose an algorithm for a single CPU problem may not be the same as the criteria used to choose a parallel implementation. Parallel linear solution tools and techniques are presented briefly in [21, 49], with specific reference to neutron diffusion by [7, 14, 15, 61, 121, 143, 147, 148, 156].

2.2.2 Eigenvalue Calculation Techniques

The standard eigenvalue problem [17, 79] is formulated as

$$Ax = \lambda x \tag{2.12}$$

A non-trivial solution for λ and x is sought where λ is a scalar, $x \in \mathbb{C}^n$ and $A \in \mathbb{C}^{n \times n}$. Problems which involve complex numbers have not been addressed by the MOOSE, while they are included within the complete domain of eigenvalue problems

the MOOSE was developed with a more restricted set of target problems in mind.

The general eigenvalue problem also includes a second square matrix B which has the same dimensions as A and is formulated as

$$Ax = \lambda Bx \tag{2.13}$$

This problem is often solved by reformulating it in standard form. If B is non-singular then the problem can be rewritten as $B^{-1}Ax = \lambda x$. If B is singular, which is relevant to the nuclear diffusion problem, then slightly more complex reformulations are necessary. If the problems are large and sparse then some of the issues of concern to linear problem solvers are also relevant to the eigenvalue problem.

Many methods have been proposed to compute eigenvalues and eigenvectors for large spares matrices. Methods like QR iteration are not appropriate for large sparse matrices because they are based on modifying the matrix by certain similarity transformations which destroy sparsity [80]. Many eigenvalue applications only require a few selected eigenvalues and not the entire spectrum.

Methods for sparse eigenproblems usually obtain the solution from the information generated by the application of the matrix to various vectors. Matrices are only involved in matrix-vector products. This not only preserves sparsity but also allows the solution of problem in which matrices are not available explicitly.

The maximum eigenvalue can be estimated in a variety of ways, the power method computes a series of approximations of eigenvectors and eigenvalues and is defined iteratively as

$$\widehat{x} = Ax^{(k-1)} \tag{2.14}$$

$$x^{(k)} = \widehat{x}^{(k)} / max\left(\widehat{x}_i^{(k)}\right) \tag{2.15}$$

$x^{(0)}$ is usually chosen with random entries, the algorithm is repeated until $x^{(k)}$ converges. As $k \rightarrow \infty$ $max\left(\widehat{x}_i^{(k)}\right) \rightarrow \lambda_{max}$, and $x^{(k)} \rightarrow$the associated eigenvector. This convergence takes place at a rate proportional to $\lambda_{max}/\lambda_{min}$. If the power method is used for a few iterations to generate a first approximation to $x^{(k)}$ then by using the Rayleigh quotient, defined as

$$R_q = \frac{x^T A x}{x^T x} \tag{2.16}$$

an estimate of the eigenvalue associated with $x^{(k)}$ can be computed. An iterative method for finding the eigenvalue, eigenvector pair, which converges faster than the power method can be constructed based on the Rayleigh quotient.

Subspace iteration is a generalization of the power method in which the matrix is applied to a set of m vectors simultaneously, and orthogonality is enforced explicitly to avoid the convergence of all the vectors toward the same eigenvector. A projection technique is often used to compute approximations to the eigenpairs of matrix A, extracting them from a given low dimensional subspace on which the problem is projected. The projection scheme is common to many other methods. The Krylov methods use a projection onto a Krylov subspace.

The most basic Krylov subspace method for finding eigenvalues is the Lanczos method. The Lanczos algorithm needs to access the matrix only in the form of matrix-vector operation, similar to the linear spectral solution methods. The Lanczos algorithm is presented in Appendix 2.

The Arnoldi algorithm [80, 81, 82] can be used for non-symmetric problems. It computes approximations of invariant subspaces from Krylov subspaces of increasing

size. During the course of the algorithm vectors are accumulated which will tend to consume large amounts of memory. These algorithms are often restarted when a maximum is reached. The Krylov-Schur [153] algorithm represents an improvement on the Arnoldi algorithm which uses a more refined restarting strategy.

A variety of numerical details must be addressed when implementing Krylov subspace algorithms. A scheme must be chosen for constructing a basis. The technique selected will have an impact on round-off errors. Locking already converge eigenvalues can considerably reduce the cost of an algorithm.

Convergence problems can arise in the presence of clustered eigenvalues. Acceleration techniques consist of computing eigenpairs of a transformed problem and then recovering the solution of the original problem. The most commonly used spectral transformation is called the shift-and-invert. The value of the shift, σ, is chosen so that the eigenvalues of interest are well separated in the transformed spectrum. The actual problem solved is

$$(A - \sigma B)^{-1} Bx = \theta x \tag{2.17}$$

This transformation is effective for finding eigenvalues near σ since the eigenvalues θ of the operator that are largest in magnitude correspond to the eigenvalues λ of the original problem that are closest to the shift σ in absolute value. This transformation is also effective in that it can be used to avoid inverting a singular matrix. The relationship between the eigenvalues of both problems is

$$\theta = 1/(\lambda - \sigma) \tag{2.18}$$

A linear system of equations must be solved whenever a matrix inversion appears in the algorithm, since directly inverting the matrix would destroy the sparsity of the

problem. That is to say that when a product of the form

$$y = A^{-1}x \tag{2.19}$$

appears rather than inverting A and multiplying this result by x what is done instead is the problem is reformulated as

$$Ay = x \tag{2.20}$$

where x is known then y can be solved for using any of the already described methods. This is an important detail since the cost of finding an eigenvalue for the general eigenvalue problem may be dominated by the cost of the algorithm which solves the inverted system. Either an iterative scheme or a direct scheme may be used to compute this result. Using a spectral transformation like the shift and invert will tend to reduce the number of steps in the eigenvalue calculation routine, although the cost is quite high since each step will require a matrix inversion accomplished via the solution of a linear system. SLEPc handles this detail automatically, this is part of what makes it so attractive.

2.2.3 Transient Integration Techniques

The MOOSE provides the necessary building blocks to assemble well understood and commonly used integration methods [70, 77, 173]. This section will present a brief overview of the theory behind integrating PDEs over time. Several methods will be presented including Euler's method, the trapezoid method, Runge-Kutta's method, multi-step methods and multi-value methods.

Two basic problems present themselves. The integration technique must be chosen, and the size of the time step must be chosen. Higher order integration methods

may be more precise than lower order methods, but each step will typically be more expensive to compute. A sequence of small steps may produce a reasonably precise solution with any integration method, but if the problem is complex enough the total number of steps required for a simple method may make the calculation impossible. Explicit methods, while attractive for their simplicity of implementation, may exhibit stability problems which can be avoided by their more expensive to compute implicit versions.

Stiff systems are those which are characterized by tightly coupled processes which represent both very fast and very slow moving phenomena. Solving stiff systems presents additional complexities because if one is forced to choose step size based on the fastest moving terms in the system an overall solution may be difficult to derive. Some special considerations for the handling of stiff systems will be presented.

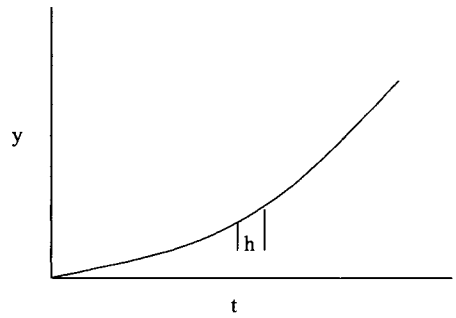


Figure 2.1: Integrating a basic ODE

For simplicity this section will focus on ODEs. The principles of integrating PDEs are similar, and in many cases the same formulations can be used. As illustrated in Figure 2.1 a function y is integrated with respect to t using discrete steps h. An initial point, y_0 is usually known, and the derivatives of y are also known, they may be dependent on both y and t.

In later chapters these techniques are used by the MOOSE framework to solve

several transient problems. In Chapter 4, the framework is tested using both Euler's explicit method and the semi-implicit trapezoid method. In Chapter 5 a third order multi-value method using stiffly-stable coefficients is used as part of the major case study. All the techniques discussed in this section can be used to specify solution PDEs for the MOOSE. Implicit methods require the application of one of the linear solvers described in the previous section to handle the large sets of simultaneous equations which often result from defining a system in terms of an implicit formulation. An implicit solution requires a rather intimate understanding of the problem at hand and so precludes the use of many pre-existing solution libraries. To develop a basic appreciation of these issues, a brief summary of integration techniques is presented.

Euler's Explicit Method

$$y_{n+1} = y_n + h \cdot \left.\frac{dy}{dt}\right|_n \tag{2.21}$$

Euler's method is a first order method. It is simple to program, numerically unstable, and can violate the Courant Friedrich Lewy (CFL) condition. The CFL condition is relevant in multidimensional simulations and relates the speed of propagation of the numeric solution in space with its speed of propagation in time. Euler's methods are usually discussed for pedagogical reasons they are rarely used in practice.

Implicit Euler

$$y_{n+1} = y_n + h \cdot \left.\frac{dy}{dt}\right|_{n+1} \tag{2.22}$$

The method is implicit since it requires $\left.\frac{dy}{dt}\right|_{n+1}$ to compute y_{n+1}. Since knowledge of $\left.\frac{dy}{dt}\right|_{n+1}$ in principle requires knowing y_{n+1} finding a solution for an implicit method will tend to be complex and require the use of a fixed point solver or some other

technique for solving simultaneous equations. Implicit formulas are always much more expensive to compute than their explicit counterparts. The implicit version of Euler's method is completely numerically stable which means that it is possible to take large steps with the method and compute solutions that will not grow without bound.

For PDEs, implicit methods can be more stable than explicit methods for reasons related to their dimensionality. Consider the illustration in figure 2.2. The point $\phi(x_j, t_k)$ when computed explicitly under a finite difference scheme can only be under the influence of the larger points in the triangle, those which precede it in time. If the phenomena being studied propagates through space faster than the simulation is allowed to evolve in time then the CFL condition may be broken and the simulation may become unstable.

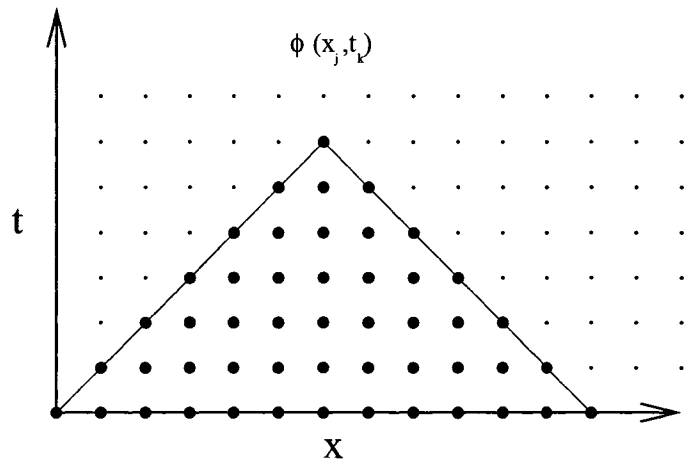


Figure 2.2: Schematic Illustration of the Limits of Explicit Methods

The CFL condition can be written formally as a relationship between the speed of the propagating phenomena in space c, the space between grid points Δx, and the size of the time step h as[1]

[1]Taken from [77].

$$h \leq \frac{\triangle x}{\sqrt{c}} \tag{2.23}$$

where it should be noted that stability does not necessarily imply accuracy. For practical purposes a step size many times smaller than that required by the CFL may be needed to compute a meaningful solution to the problem. While the implicit version of Euler's method guarantees stability, stability alone is not a sufficient condition for an accurate solution. Improved accuracy can be achieved through the use of higher order integration methods.

Trapezoid

$$y_{n+1} = y_n + h \cdot \left(\left.\frac{dy}{dt}\right|_n + \left.\frac{dy}{dt}\right|_{n+1} \right) /2 \tag{2.24}$$

The trapezoid method is a second order semi-implicit method. It is also sometimes called the Crank Nicholson method. It requires $\left.\frac{dy}{dt}\right|_{n+1}$ to compute y_{n+1}. Because it is a semi-implicit method it cannot take steps as large as the implicit Euler method, however since it is second order accurate the steps that it does take are much more precise.

Runge-Kutta

The Runge-Kutta methods use derivatives computed at a variety of positions. While previously it was convenient to write $\left.\frac{dy}{dt}\right|_n$ to represent the first derivative of y computed at the position (t_n, y_n), for Runge-Kutta methods the derivative $\left.\frac{dy}{dt}\right|_{(t_n,y_n)}$ is written as $f(t_n, y_n)$ so that complex expressions can be used for t and for y.

$$k_0 = f(t_n, y_n)$$

$$k_1 = f(t_n + h/2, y_n + \frac{h}{2}k_0)$$

$$k_2 = f(t_n + h/2, y_n + \frac{h}{2}k_1)$$

$$k_3 = f(t_n + h, y_n + hk_2)$$

$$y_{n+1} = y_n + \frac{h}{6}(k_0 + 2k_1 + 2k_2 + k_3) \tag{2.25}$$

Runge-Kutta methods have a variety of formulations. The above representation is a common implementation, it is fourth order accurate and explicit so each step is very inexpensive to compute. A drawback of this method is that each step requires several calculations for $\frac{dy}{dt}$ at points defined by the particular implementation. Runge-Kutta methods are appropriate when an expression exists for the calculation of the first derivative which can be evaluated independently of the solution being solved for.

Implicit Runge-Kutta

The implicit Runge-Kutta methods also have multiple possible definitions. Here a general form for a two stage implicit Runge-Kutta is defined in terms of several constants, α, β, and γ. The details of this method are discussed by [70].

$$k_1 = f\left(t_n + \alpha_1 h, y_n + h\beta_{11}k_1 + h\beta_{12}k_2\right)$$

$$k_2 = f\left(t_n + \alpha_n h, y_n + h\beta_{21}k_1 + h\beta_{22}k_2\right)$$

$$y_{n+1} = y_n + \gamma_1 h k_1 + \gamma_2 h k_2 \tag{2.26}$$

Implicit Runge-Kutta has better stability characteristics than explicit Runge-Kutta, although Implicit Runge-Kutta methods are used less frequently. Like the other implicit methods it is more expensive to compute than its explicit version, and like the explicit Runge-Kutta methods it requires multiple $\frac{dy}{dt}$ evaluations.

Multi-Step

Multi-step methods are often expressed using a slightly different notation than the previous methods, where the first derivative of y with respect to t at a point k, i.e. $\frac{dy}{dt}\big|_k$ is written simply as y'_k. Time steps are usually strictly regimented. A multi-step method normally computes all derivatives from previously known positions in time and space, indicated by the subscripts.

$$y_{k+1} = y_k + \frac{h}{24}\left(55y'_k - 59y'_{k-1} + 37y'_{k-2} - 9y'_{k-3}\right) \tag{2.27}$$

$$y_{k+1} = y_k + \frac{h}{24}\left(9y'_{k+1} + 19y'_k - 5y'_{k-1} + y'_{k-2}\right) \tag{2.28}$$

Multi-step methods have a variety of implementations much like Runge-Kutta methods, both explicit and implicit methods of various orders can be defined. Each particular definition has its own stability properties and accuracy characteristics. The

trapezoid method can be considered a second order multi-step method. The above implementation is a fourth order predictor corrector method. Equation 2.27 provides an initial estimate for y_{k+1}, equation 2.28 provides a correction. Multi-step methods work by accumulating a history of first derivatives and using nonlinear extrapolation methods to estimate the next step that they take. Changing step size with multi-step methods can be difficult, and multi-step methods also require some start-up method to prime their history. If a system starts in steady state then the start-up method can be as simple as initializing the derivative history with zeros. Sometimes methods which do not use a history, like the Runge-Kutta methods can be used to initialize a multi-step method.

Multi-Value Methods

Multi-value methods require the maintenance of several variables as part of their calculation. A vector of derivatives of several higher orders is maintained for each step, called $\mathbf{y_k}$, which is computed from a temporary vector of derivatives, called $\hat{\mathbf{y}}$, and a constant α, also evaluated at each step. In addition the method itself is defined by a transformation matrix and vector, called $\mathbf{B}$ and $\mathbf{r}$ respectively. Details of the method, and various choices for $\mathbf{B}$ and $\mathbf{r}$ are discussed by [70].

$$\mathbf{y}_k = \begin{bmatrix} y_k \\ hy'_k \\ (h^2/2)y''_k \\ (h^3/6)y'''_k \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 1 & 1 & 1 \\ & 1 & 2 & 3 \\ & & 1 & 3 \\ & & & 1 \end{bmatrix}$$

$$\hat{\mathbf{y}}_{k+1} = B\mathbf{y}_k$$

$$\alpha = h(f(t_{k+1}, y_{k+1}) - \hat{y}_{k+1})$$

$$\mathbf{r} = \begin{bmatrix} 3/8 \\ 1 \\ 3/4 \\ 1/6 \end{bmatrix}$$

$$\mathbf{y}_{k+1} = \hat{\mathbf{y}}_{k+1} + \alpha\mathbf{r} \tag{2.29}$$

Multi-value methods [70] are computationally equivalent to multi-step methods although they use a somewhat different technique to represent the problem. Instead of maintaining a history of individual points, a sequence of derivatives is kept at the current point. The advantage that this has over a multi-step method is that it is relatively easy to change step sizes. The preceding example is a common formulation of a 4th order multi-step integration method. The choice of the vector $\mathbf{r}$ will have an

impact on the accuracy and stability of the method. As mentioned in the opening part of this discussion, some integration problems are classified as stiff and are difficult to solve. Several vector choices for **r** are listed in Table 2.2, these are taken from [70]. Defining the vector **r** from the entries in the first table will yield a multi-value method with reasonable stability properties, and a high degree of precision. Defining **r** based on the elements listed in the second table provides a method which trades precision for stability. While slightly more steps will be required with the stiffly stable coefficients, the integration procedure is much more likely to converge to a correct result especially if the problem suffers from a mixture of fast and slow moving components. The results derived in Chapter 5 use the multi-value integration method with a third order integration scheme with constants taken from the stiffly stable table.

degree	r[1]	r[2]	r[3]	r[4]	r[5]
3	5/12	1	1/2		
4	3/8	1	3/4	1/6	
5	251/720	1	11/12	1/3	1/24

Multi-Value Coefficients

degree	r[1]	r[2]	r[3]	r[4]	r[5]
3	2/3	1	1/3		
4	6/11	1	6/11	1/11	
5	25/50	1	35/50	10/50	1/50

Stiffly Stable Multi-Value Coefficients

Table 2.2: Coefficients for Multi-Value Methods Vector r

Step size selection can be accomplished by a variety of techniques. There are special methods for selecting step size for both the multi-step and multi-value methods based on careful analysis of the properties of those methods. A general technique which applies to all integration methods is step doubling. A step is computed in two

ways, first the step is taken in the normal way for the method. A second estimate of the new value is computed by taking two steps each one half the size of the step taken in the first estimate. The two results are compared, if their difference is above a certain threshold then the step is rejected, the step size is reduced, and the procedure is repeated. If the difference between the two estimates is below a certain threshold then the step size is accepted, and the step size can be increased.

2.3 Modeling Engineering Geometries

Physics simulations tend to fall into two broad categories in terms of their mapping between the physical world and the simulated space of the computer. The first common mapping is based on linked components. In this simulation design components are connected in an abstract fashion, and the precise position of each simulation element is not as important as how it is connected to its neighbours and the behaviour that it models. Electrical circuits are normally simulated through a component based design where the behaviour of the overall circuit is determined by the connectivity of individual components rather than their precise physical position on a circuit board. The Berkley SPICE simulator is an example of such a simulator. Fluid flow models which model a pipe network are also often represented in a component wise fashion where the relative position of the end points of the pipe are of concern, but the actual location of the pipe in space and its overall shape are unimportant. For example see [120].

The second common mapping between simulation elements and physical geometries is a one to one mapping, where the simulation space is discretized and represented by some sort of regular pattern of points which can be calculated on a computer. This thesis has focused its efforts on this style of discretization. Several meshing techniques

are discussed in the following section.

2.3.1 Mesh Techniques

Mesh elements can take a variety of forms, the simplest two dimensional mesh is a Cartesian square grid. Meshes with regular non-square elements are also possible, hexagonal and triangular elements are also common. Often the spacing of grid lines in a Cartesian mesh is adjusted so that there are more mesh lines in an area of interest to the simulator, typically this requires only a relatively simple adjustment of the computational model to account for the unequal spacing. If a simple regular mesh can be applied to a problem this is always preferable. Regular conformal meshes for which each mesh vertex is connected only to other mesh vertices in a simple predictable pattern are easy to analyze mathematically and are also easy to program on a computer.

If computer resources are not limited, or if a problem is not very complex, a simple grid can provide a very effective solution to a simulation problem. Unfortunately this is not usually the case. For many problems the simulation features of interest may only occur in a small localized region within the problem domain and using a simple grid to mesh the entire domain can be very wasteful of computational resources. For other problems it may be necessary to compute approximations to curved surfaces which do not conform to simple geometric subdivisions. The irregular Cartesian grid pictured in Figure 2.3c can be used to focus in on certain parts of a simulation but this method also tends to create additional areas of focus which are not necessarily of interest to the modeler.

There are a variety of solutions to problems that need either local grid focus, or grid shapes that are well fitted to the exact shape of a problem. Several examples

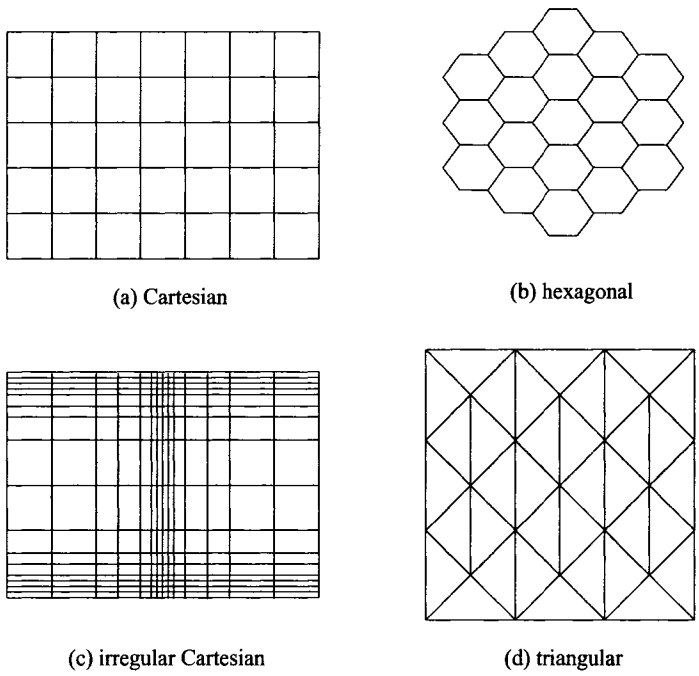


Figure 2.3: Simple Grid Types

are illustrated in Figure 2.4, see [158, 159] for a comprehensive discussion. One of the most commonly applied solutions in fluid mechanics is the use of unstructured grids. An unstructured grid uses a collection of polygons to fill in a region, usually triangles. The polygons need not be of the same size or in the same orientation. Unstructured grids (Figure 2.4a) are very good at representing arbitrary curves and unusual geometries and have become the preferred griding mode for fluid dynamics problems. Unstructured grids are more complicated to implement and have larger memory requirements than structured grids, so they are not necessarily the best choice for all problems.

Structured grids need not be simple Cartesian grids, they are often bent to fit surfaces and can provide excellent approximations to curved shapes for certain applications, see Figure 2.4b. In such situations curved meshes are often assembled to-

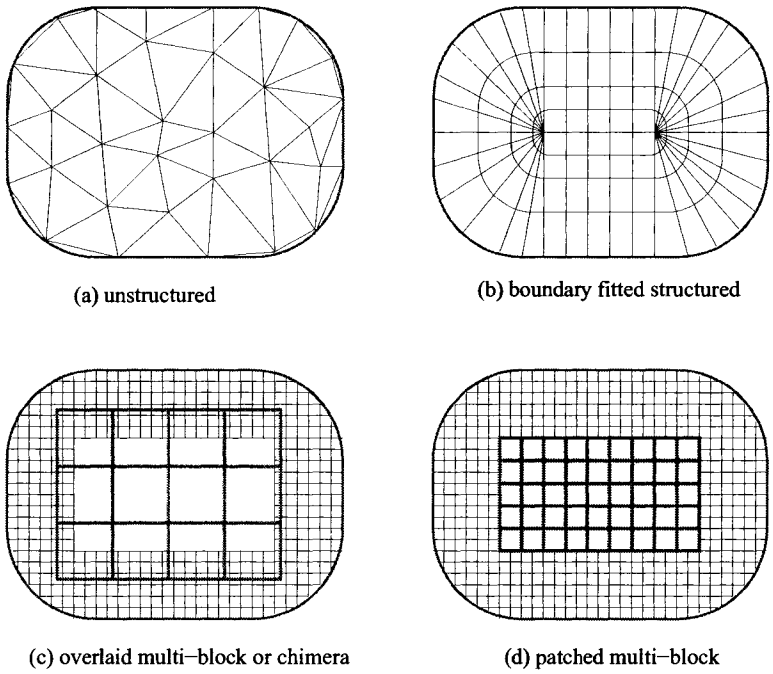


Figure 2.4: General Grid Types

gether to construct multi-block meshes. The literature on multi-block meshes points out that multi-block meshes are also good for modeling moving components [158]. The MOOSE uses geometrically Cartesian grids and uses individual grids to model separate components, curved grids are not implemented. The MOOSE's grids are structured, but need not be conformal. A non-conformal mesh allows for vertices to connect with grid edges, i.e. in Figure 2.4 examples a) and b) are conformal, but c) and d) are non-conformal.

Two major schemes exist for handling the boundaries between connected grids. Overlaid multi-block grids, (sometimes called Chimera grids, see Figure 2.4c) use interpolation to communicate between individual grid structures. A certain degree of overlap is required to ensure good communication between the grids. In the patched grid scheme (see Figure 2.4d), grids are nested exactly inside of each other, usually

a finite volume interpretation of the mesh is used where variables are considered to exist at cell centers rather than mesh vertices.

The MOOSE uses a patched multi-block grid system to model motion. As discussed in the next section there are several computational reasons for why this particular scheme was chosen versus an overlapping grid. The analysis of patched meshes is more direct, and provides a variety of simplifying mechanisms which facilitate the correct linking of grids, as will be illustrated in Chapter 3.

2.3.2 Composite Grid Methods

There is much debate in the literature [155] which discusses the correctness of using interpolation methods to link overlapping meshes. Conservation is a generic physical property; it often refers to conservation of mass, or conservation of energy. Poorly designed simulations will fail to maintain this general property. In the case of patched meshes and overlapping meshes it is often the case that the meshing technique is responsible for some small source or sink of energy or material which leads to overall inaccuracies in the simulation. Some papers [24, 25, 131, 132] argue that using interpolation of any order will lead to errors induced by the failure to conserve the flow across mesh borders and that mesh overlap should be avoided altogether. Other papers argue that under certain circumstances connecting meshes with higher order interpolation methods can be sufficiently precise [38, 39, 122], up to the order of error induced by the mesh size. Still other papers [99, 169] discuss the idea of avoiding the problem altogether by linking structured mesh elements by unstructured partial meshes or clipping overlapping sections of meshes to restructure the geometry of the problem. See also [26, 41, 116, 128, 152, 157, 175] for additional background on this discussion.

Most of this discussion has taken place in the context of fluid mechanics simulations where the goal is usually to model applications of the Naiver Stoke's equations around a smoothly curving structure. The main motivation in this problem area for using multiple meshes to build a smooth boundary that represents the surface of something like an airplane wing. Building such contours from a collection of structured meshes rather than a single structured mesh or an unstructured mesh has a variety of implementation advantages which prompted research in this area. Many authors who work in the nuclear engineering area recognize the potential benefits for using either moving mesh techniques, or adaptive meshes [177, 182, 183]. To date, few (if any), neutron diffusion projects have been built using either adaptive geometric methods, or linked meshes.

Linked structured meshes were chosen over an unstructured mesh for several application specific reasons related to the modeling of neutron diffusion. Neutron diffusion physics breaks down at the resolution of the neutron's mean free path, or the distance that a neutron can travel without colliding with anything. This distance ranges between a few cm, and about 10 cm depending on the characteristics of the reactor and the model. Diffusion physics makes anisotropic scattering assumptions, so not only do neutrons travel large distances without collisions, but when they do collide the direction of their scattering is only modeled in a very rough way. Reactor core designs are also often very simple, using either a cylindrical or a rectangular prism for the overall geometry. Given the geometric approximations employed by neutron diffusion physics there is no real advantage to be obtained by precisely modeling reactor geometry with an unstructured mesh.

The paper [170] presents an example of an unstructured mesh solved with a finite element method used to model a reactor. Its author uses a transport model to simulate a cylindrical core rather than a diffusion model so in this case the more complex mesh

is somewhat justified. Recall that transport models do not make anisotropic scattering assumptions and precisely track the motion of neutrons and their interactions with material interfaces, the cost is usually a much more complex model.

An unstructured mesh does not necessarily provide all the solutions needed by a moving mesh. If a single mesh is constructed for a given geometry and points are translated only limited motion can be modeled. This style of motion may be sufficient to model the flexing of a bridge or vibrations in an airplane wing, but as discussed by [78], for problems which involve large deformations, the mesh can easily become entangled. Even in cases where neither twisting nor tangling of the mesh are at issue finite element methods have certain limitations. The elements are limited in the shapes that they can have, how big they can be, and how they can be connected, which may place limitations on component motion.

Despite these problems with mesh deformation, re-meshing the problem for new configurations is possible. There are also other novel strategies such as defining components within structured regions and then using an unstructured mesh to connect those regions, see [99].

The strategy employed by the MOOSE, of using moving structured patched meshes, should be understood as one possible approach, but not the only possible approach. Using moving structured patched meshes was preferred partly because it is acknowledged in the nuclear literature as a possible approach by [177], and most closely follows the existing discussion of nuclear simulations. Structured meshes are also simple to deal with, in the third chapter where non-conformal mesh construction issues are addressed along side arbitrary physics, the advantage of limiting the mesh design to simple linked Cartesian grids becomes more obvious.

2.3.3 Nodal Methods and Transient Simulations

Nuclear Engineering transient models are often implemented using a compromise between standard mesh methods and linked components. Coarse mesh and nodal methods represent a compromise between explicitly representing the simulation domain on a fine mesh which is capable of accurately modeling the geometry of a reactor core, and a completely abstract model which links regions which use nonlinear models to approximate the behaviour of a large section of the reactor core. These methods provide certain advantages over standard discretization methods, but at the cost of rather complex implementation.

Coarse mesh methods [110, 154] are motivated by the fact that in some instances a reactor may be adequately described by a model consisting of homogeneous regions that are relatively large. A region is defined as large when it is greater than the diffusion length of a neutron, typically on the order of more than 10 cm. While a coarse mesh may be adequate to describe the geometry, a finite difference method will require a relatively fine mesh to maintain accuracy. Coarse mesh methods are able to use mesh sizes which are much larger than finite difference methods because they use higher order approximations to the spatial variations of the unknowns within a mesh cell. The rational is that although the computational effort per mesh cell is increased, the reduction in the number of mesh cells results in an overall reduction in the amount of work required to solve the problem.

Like coarse mesh methods, nodal methods utilize relatively large computational mesh cell to solve multi-dimensional reactor problems using significantly less computer resources than the fine-mesh finite difference method. Early nodal methods required a variety of schemes to deal with face-averaged partial currents and the node averaged fluxes. Coupling parameters for a node are defined as the ratios of

the interface integrated out-going partial currents to the node-averaged flux. The homogenized parameters are usually computed by weighting the spatially dependent cross sections with the flux solution obtained in an assembly calculation with zero net current boundary conditions. These parameters are computed using a reference fine mesh calculation. While these methods work well in situations in which the conditions analyzed using the nodal method closely resemble the reference condition at which the coefficients were computed, they often breakdown when the difference between the analyzed and reference conditions becomes large.

Nodal methods work in part by solving a nonlinear one dimensional approximation to the flux in the X, Y and Z directions for each cell. This is less work that solving a fully three dimensional approximation to the entire cell. If the flux is to be reconstructed for the entire cell higher order polynomial techniques can be used. Special leakage terms are normally included to deal with neutron flux which is unaccounted for near the far boundaries of the cell.

Transverse integrated nodal methods assume that nodes are either truly uniform throughout their entire volume, or that they may be adequately represented using node-averaged values of the cross sections and diffusion coefficients. This assumption of uniformity of intra-nodal composition does not apply to most reactor calculations that employ assembly or quarter-assembly sized nodes. These issues are addressed by advanced nodal homogenization schemes that yield equivalent diffusion theory parameters that allow transverse integrated nodal codes to compute node-averaged quantities that agree closely with the results of fine mesh calculations in which the heterogeneity within the node is explicitly represented.

The rod cusping problem [20, 96, 103, 117] results when a naive approximation for the motion of either a control rod or fuel assembly is used as part of a coarse mesh solution. The naive approximation models a large cell which is partially occupied

by the leading or trailing tip of a moving assembly by taking a weighted average of the cross section constants that represent the fully roded cell and the unrodded cell. When reactivity is plotted versus assembly insertion distance the naive approximation typically results in a reactivity curve with a series of cusps which fall in between positions where the assembly is aligned with the mesh. The size of the cusps are related to the precise problem being studied, but at least for problems which use coarse meshes the deviations from the correct reactivity are considerable and not acceptable.

Many techniques exist for the treatment of the rod cusping problem in the context of nodal solutions including the approximate flux weighting method, the analytical flux weighting with discontinuity factor methods, the bi-linear weighting method and the equivalent-node method. All of these methods provide satisfactory approximation for the motion of assemblies with the exception of the volume weighting method [177].

Difficulties with Nodal Methods

Nodal methods are very complicated to derive, some variations require careful calibration and are only valid for a narrow simulation range, and often require the computation of a reference solution. Early nodal methods have been criticized for being inconsistent with the neutron diffusion equations. The modern nodal methods, also known as the transverse-integrated nodal methods, are consistent but add more complexity so that they are often restricted to two energy groups. The error of nodal discretization is difficult to analyze hence the benchmarking of nodal codes still relies on a finite difference counterpart. The unusual choice of nodal unknowns, the node-averaged and face-averaged quantities, makes the resulting discretized system incompatible with fast iterative methods. It is usually very difficult to accelerate the

convergence rate of a nodal system of tens of thousands of equations. While flux values can be easily extracted from a finite difference simulation the nodal solution is for a node-averaged quantity which requires dehomogenization to obtain the reaction rate distribution within the node [118].

2.4 Problem Solving Environments

A Problem Solving Environment (PSE) automates the process of model construction by creating a reusable tool for a domain of problems. The classical simulation design and construction process is unified in a single reusable tool that encapsulates expertise from a variety of domains. A PSE collects together several solution methods and models addressing issues such as appropriate software reuse, intrinsic model validation, and intelligent algorithm selection.

In the last 15 years the idea of a PSE has penetrated into a variety of engineering disciplines. The basic explanation for this recent surge of interest lies with the development of graphical user interfaces, improvements in overall hardware speed, as well as the standardization of scientific software tools. One early discussion of PSEs appears in an article by Gallopolous, Houstice and Rice [66], where the history of the idea of an all-purpose scientific solution tool going back as far as 1960 is summarized. In the same article the report provided by the 1991 workshop on PSEs sponsored by the National Science Foundation is given. Since this article, hundreds of articles have appeared which directly use the term PSE. With the exception of the text by [89], there are few discussions which attempt to categorize or summarize the body of literature on PSEs. While the idea of a PSE is itself intuitive, few projects come close to fully actualizing the idea. The fully fledged PSE must somehow be all things to all people, so while on the one hand the idea is relatively easy to appreciate, its

realization is still largely elusive.

The final section of this chapter presents a collection of PSEs roughly categorized as

- Special Purpose PSEs
- Multi-Physics PSEs
- Networked Scientific PSEs
- Collaborative PSEs

For each category several representatives are discussed. These categories serve to paint a rough picture of the current state of the art with respect to PSEs dividing their designs into four broad groups. Given the wide variety of PSE projects, a proper taxonomy and analysis of properties and trends could easily fill an entire text. Many projects and trends have been left undiscussed.

2.4.1 Special Purpose PSEs

Many PSE systems limit their design to a specific range of physical phenomena. These tools are often built by specialists in the area who are attempting to generalize some of their modeling techniques. Presented here are several different examples of domain specific PSEs including GEANT4, WBCSim, Entero and ICEPIC. Each effort is driven by a relatively restricted domain. Also of relevance but not discussed in this section are tools described by [11, 75, 78, 93, 94, 119, 138].

The tools described in this section are PSEs in the sense that they address a well understood but limited range of problems and provide a flexible framework which is applicable to only a select set of related phenomena. In some senses these may be the

most effective tools described in this chapter since they are driven by the efforts of domain experts to collect together a set of closely related tools, and make those tools inter-operate in a very practical way for the benefit of their peers. In contrast some of the very general tools described in subsequent sections attempt to address issues from a broad genera of related problems. It is arguable that increasing the genericity of a tool reduces its effectiveness in all of the areas that tool attempts to address.

GEANT4

GEANT4 [3, 12, 60, 92, 111, 142, 160, 180] is discussed in some detail here because it represents a PSE which is very broad in scope and includes many aspects of Nuclear Engineering and Electrical Engineering. GEANT4 is also very well documented in the literature by dozens of research publications. The work for GEANT4 is based on two studies done independently at CERN and KEK in 1993. Both groups sought to investigate how modern computing techniques could be applied to improve what was offered in the existing GEANT3 program. A proposal was submitted to the CERN director to build a new program built using object oriented technology, the project resulted in a worldwide collaboration of 100 scientists and engineers drawn from more than 10 experiments in Europe, Russia, Japan, Canada and the United States. While geographically distributed software development and large-scale object-oriented systems are no longer a novelty, the authors consider the GEANT4 Collaboration, in terms of the size and scope of the code and the number of contributors, to represent one of the largest and most ambitious projects of this kind. Shortly after the release of the first version in 1999, the GEANT4 Collaboration was established to continue the development and refinement of the toolkit and to provide maintenance and user support. The Collaboration Board a Technical Steering Board and several working

groups manage the groups' resources and monitor the agreed responsibilities of the affiliates. GEANT4 is freely available for download and runs on multiple platforms.

GEANT4 Overview

GEANT4 is a toolkit for simulating the passage of particles through matter. It includes a complete range of functionality including tracking, geometry, and physics models. The physics processes offered cover a comprehensive range, including electromagnetic, hadronic and optical processes, a large set of long-lived particles, materials and elements over a wide energy range starting from 250 eV and extending in others to the TeV energy range. It has been used in applications in particle physics, nuclear physics, accelerator design, space engineering and medical physics.

Modern particle and nuclear physics experiments pose enormous challenges in the creation of complex yet robust software frameworks and applications. Of particular importance is the ever-increasing demand for large-scale, accurate and comprehensive simulations of the particle detectors used in these experiments. Similar considerations arise in other disciplines, such as: radiation physics, space science, nuclear medicine and many other areas where particle interactions in matter play a role.

GEANT4 acts as a repository that incorporates a large part of all that is known about particle interactions; moreover it continues to be refined, expanded and developed. Object-oriented methods have allowed the effective management of complexity and the limitation of dependencies by the definition of a uniform interface and common organizational principles for all physics models.

GEANT4 Design

GEANT4's design was driven by the software needs of modern experiments. A typical software model contains a components-event generator, detector simulation, as well as reconstruction and analysis methods that can be used separately or in combinations. Simulation models should be modular and flexible, its physical models should be transparent and open to user validation. It should allow the user to understand, customize and extend it in all domains. Its modular architecture should enable the user to pick only those components which are necessary.

The key domains of the simulation of the passage of particles through matter are:

- geometry and materials
- particle interactions in matter
- tracking management
- digitisation and hit management
- event and track management
- visualisation and visualisation framework
- user interface

These domains naturally lead to the creation of class categories with coherent interfaces and for each category, a corresponding working group with a well defined responsibility. GEANT4 is described as a toolkit by its authors because this term implies that a user may assemble a program at compile time from components chosen from the kit or supplied by the user.

GEANT4 allows the user to create a geometric model with a large number of components of different shapes and materials. The user can define sensor elements that

record information. GEANT4 also provides a comprehensive set of physics processes to model the behaviour of particles. The user can interact with the toolkit through one of several different graphical user interfaces. Both the geometry and the particle tracks can be visualised through a variety of graphics systems. The user interface is sufficiently flexible that its implementation can be combined with that of other simulation systems.

Openness was an important design goal for the authors. An object oriented implementation allowed for a clear and customisable correspondence between particles and processes and offers a choice of models for each process. Cross section computations as well as the parametrization and interpolation of databases are all completely exposed. The physics is implemented through 17 major categories of classes. Categories include

- global: covering the system of units constants and random number handling
- geometry: covering volumes for detector description
- intercoms: allows GEANT4 code to interact with the user interface and other plugins
- track: contains classes for tracks and steps
- processes: processes make use of tracks and contains models of interaction
- transportation: handles the transport of particles in the geometry model
- event: manages system events
- visualization: plotting and rendering of computed data
- persistency: checking pointing of simulated data

- user-interface: interactive graphical widgets and buttons presented to the user

The event category provides an abstract interface to external physics event generators, this isolation allows a GEANT4 based simulation program to not be dependent on specific choices for physics generators and also to be independent of the specific solution.

The geometry category offers the ability to describe a geometric structure and propagate particles efficiently through it. Some concepts have been borrowed from previous implementations but improvements, refinements and advances have been made in key areas. GEANT4 handles solids with simple shapes, like rectilinear boxes, trapezoids, spherical and cylindrical sections or shells and are stored through Constructive Solid Geometry (CSG). Solids may also be combined by Boolean operations, intersection, union and subtraction.

The tracking category steers the invocation of processes. Each particle is moved step by step with a tolerance that permits significant optimising of execution but that preserves the required tracking precision. All physics processes associated with the particle are defined by a step size. For a particle at rest this is a time rather than a length. The smallest of either the maximum allowed step as defined by the user, or the steps proposed by all of the attached processes is chosen.

A variety of different approaches are present for the various types of physics. Particle decay is straightforwardly calculated from the mean life of the particle. The electromagnetic physics classes handle the interactions of leptons, photons, hadrons and ions. The package is organised as a set of class categories:

- standard: handling basic processes for electron, positron, photon and hadron interactions

- low energy: providing alternative models extended down to lower energies than the standard category
- muons: for handling muon interactions
- optical: providing specific code for optical photons
- X-rays: providing specific code for X-ray physics
- utils: collecting together utility classes used by the other categories

Classes for particles and materials implement facilities for describing the physical properties that are necessary for the simulation of particle-matter interactions. The particles class describes basic properties like mass, charge, etc. and also must encode the processes to which a particle is sensitive. The materials category reflects what exists in nature: materials are made of a single element or a mixture of elements.

Various user interface tools like Motive, Tk/tcl, JAVA and others have been used to implement the command capturer. Various groups which participate in the GEANT4 Collaboration have contributed their own front-ends to the command system. Currently available implementations are as follows:

- batch: non-interactive configuration file driven
- tcsh-like: a command shell like implementation for interactive sessions
- GAG: a client/server adaptive GUI reflecting GEANT4 states
- OPACS widget manager implementation

GEANT4 visualisation can render detector geometry, particle trajectories, tracking steps, hits, and text labels. The visualization driver can directly access graphics

libraries, communicate with independent processes through either pipes or sockets, or can simply write an intermediate file for a separate viewer.

There are various analysis systems that generate histograms, analyse event data statistically. GEANT4 uses the AIDA abstract interface, there are several examples of data analysis systems compatible with AIDA including JAS, Lizard and OpenScientist.

Entero

The long-term goal for the Entero [68] environment is to research and develop a module-oriented, multi-physics, mixed-fidelity system simulation environment for engineers to enable rapid system performance analysis and design optimization. Major design goals for the environment include providing a systems view for analysis, a module-oriented view, enabling modules of different physics types to be coupled together, providing mixed fidelity modules and enabling optimization and uncertainty quantification studies. Coupling different physics types allow an engineer to model electrical circuit in a thermal or radiation environment and monitor its performance. Adjusting the fidelity of the model allows the designing engineer to replace a coarser finite element mesh with a finer one, or a linear model with a nonlinear model.

One focus for the Entero environment is modeling systems containing electrical circuits that are exposed to fires. Electrical circuits can be embedded in each module, but not connected between modules. Electrical activity is calculated using the SPICE circuit simulator and circuits are specified through standard SPICE netlist files. The coupling between the zero-dimensional thermal models and the circuit models is one way. It is computed using the zero-dimensional black body thermal modules and then this temperature is imposed on any circuit embedded in the module, any heat

generated by the circuit is neglected. In the case where higher dimensional models are used to compute the heat generated by an object, an average over the component is computed which reduces it to a zero dimensional figure before it is communicated.

ICEPIC Prototyping Environment

Improved concurrent electromagnetic particle in cell (ICEPIC) [30, 126] is being developed at the Air Force Research Laboratory. Of particular interest to the United States Department of Defense is the design of directed energy devices that generate high-power microwave (HPM) pulses. The Air Force Research Laboratory is working to bring about a paradigm shift in the design, analysis and construction of HPM sources. This shift involves harnessing high performance computing and using it throughout the research process.

ICEPIC is a relativistic 3-D Cartesian variable mesh electromagnetic parallel PIC code capable of simulating a wide variety of electromagnetic problems including high power microwave devices. HPMs are generated from the resonant interaction of intense relativistic electron beams with electromagnetic cavities. The interaction transforms electron kinetic energy into electromagnetic energy. Maxwell's equations are used to analyze these systems. ICEPIC is capable of managing millions of computational cells containing billions of modeled particles. Test cases run on networks with 100s of workstations. ICEPIC has successfully simulated various real-world HPM devices, such as the magnetically insulated line oscillator (MILO) and the relativistic klystron oscillator (RKO).

Having usable, reliable, high performance physics simulation has changed how the Air Force Research Laboratory engineer their designs. In the past, code runners, practical experimentalists and code developers all worked with different sets of

assumptions. The old parallel codes were difficult enough to use that too many unrealistic simplifications were made by the persons executing the codes in order to assure that simulations ran to completion in reasonable periods of time. The cumulative effect of simplifications by each group meant that effectively different devices were being studied. Tools like ICEPIC, have simplified the execution of high performance problems to the point where realistic details like full geometry, multiple sharp edges, and nonzero vacuums can be considered.

2.4.2 Multi-Physics PSEs

In sharp contrast to special purpose PSEs, multi-physics PSEs attempt to address a very broad scope of problem domains which link various physical phenomena. These tools usually identify a modeling technique which is applicable to a wide variety of problem domains. Many tools base their approach on a particular solution technique, like FEMLAB which applies the finite element method in combination with unstructured meshes against a wide array of problem types. Problem types in FEMLAB are catalogued within a sequence of templates and presented to the user in menu format through a verbose interface. SCINAPSE and CTADEL are academic projects which both employ ideas from computer algebra techniques allowing the user to specify solution methodologies through their own specially developed scripting languages. Although CTADEL's authors focused there research efforts on weather prediction it is included in this category because it embodies many of the same principles of flexible model description as well as code generation. Also of interest and applicability to a broad set of problems but not discussed in this section are [32, 33, 37, 84, 87, 95, 101, 174]

While each of these tools is able to address a broad array of problem types, and

although each is described in generic terms to some extent, it should be kept in mind these tools rely on a limited set of problem characteristics. No tool can be perfectly generic and at the same time, simple to use.

CTADEL

CTADEL [161, 162, 164] is a programming environment capable of transforming high-level PDE problem specifications into efficient codes for serial, vector, and parallel computer architectures using computing-cost heuristics and architecture-specific symbolic transformations. Software synthesis is the automatic translation of a problem, defined at a high level of abstraction, into executable code, by stepwise refinement. Code generation is distinct from compilation in that a typical compiler simply performs a fairly literal translation from a high language to a lower language, whereas a code generator typically makes more inferences, and builds executable code which might contain loops, subroutines, and conditionals which are not explicitly spelled out in the specification.

CTADEL implements a translation mechanism with inherent vector and matrix semantics to transform vector equations into scalar equations. The translation mechanism follows standard notational conventions for PDE operators and adopts a MATLAB-like programming style for symbolic matrix and vector operations. The bridge between a model with continuous derivatives and integrals and the numerical schemes with stencil operations and quadratures is laid by employing operator overloading techniques. CTADEL's system incorporates a symbolic and algebraic simplifier to transform problem specifications into intermediate representations and for applying simplification and optimization on the intermediate problem representations and code.

The original version of CTADEL used semi-Eulerian methods. Semi-Lagrangian schemes, in contrast, can take much larger time steps. Since the CTADEL software is based on code generation it was possible to extend its calculation methods with additional interpolation schemes including linear and quadratic methods.

FEMLAB

FEMLAB [42] is a general tool for solving PDEs that arise in a variety of disciplines including heat transfer, fluid flow, electromagnetics, structural mechanics and many other areas. Models can be constructed in 1-D, 2-D or 3-D. FEMLAB provides a detailed graphical user interface which facilitates model construction as well as output rendering. FEMLAB allows several physical disciplines to be combined together. This forms FEMLAB's definition of multi-physics.

FEMLAB allows users to enter PDEs directly and does not hard code equations for particular physical regimes. Direct entry of PDEs provides much of FEMLAB's generality and flexibility. FEMLAB's standard capabilities can be extended through script programming. Simulations can be paused and check-pointed at any stage in a calculation, calculation methods can be changed in the midst of a computation. FEMLAB is built on top of MATLAB so FEMLAB simulations can be easily incorporated with other MATLAB tools. FEMLAB employs code generation by converting the user's graphical input into MATLAB code. Exported code can then be modified or specially tweeked by the user.

FEMLAB solves numerically elliptical, parabolic and hyperbolic nonlinear differential equations using the finite element method. FEMLAB uses the Galerkin principle for nodal finite elements for transformation of differential equations into equivalent systems of algebraic equations.

FEMLAB has been aggressively marketed to the academic community as the first software tool to solve nontrivial differential equations in a fast and accurate fashion. FEMLAB is discussed by [1, 56, 105, 113, 149, 163, 176].

FEMLAB's generality does not satisfy everyone's needs. Afeyan writes [1] that FEMLAB does not provide a way to control step size which would be appropriate to the particular equation the author is considering. Afeyan also writes that FEMLAB prevents certain kinds of nonlinear coefficients from being defined which are important to optical semiconductors. Komarov writes [105] that for certain waveguide structures FEMLAB omits boundary conditions for some cases which would prevent it from being able to solve certain microwave heating problems. The reservations reported by both authors are fairly domain specific, and do not seem to be fundamental design issues with FEMLAB.

SCINAPSE

SciNapse is a code generating PSE for solving scientific computing problems without low level programming [5, 6]. SciNapse has generated codes that solve the transient version of Maxwell's equations in 3D dispersive, anisotropic media, the Black Scholes equations for valuation of multiple asset derivative securities in computational finance, nonlinear, multidimensional, multispecises reaction diffusion equations for chemical and nuclear applications and time domain solution of viscoelastodynamic equations in 3D anisotropic media.

The codes that SciNapse generates can include features such as general coordinate transformations and grid generators, various linear solvers and preconditioners, higher-order differencing techniques, automatic interpolation of equation parameters from multidimensional tabular input data, jump conditions in both space and time

dimensions, free boundaries, and imposition of solution constraints such as positivity. The goal is to generate codes in which the finite-difference PDE solution becomes the forward engine for solving multiparameter inverse problems via nonlinear optimization. Problem specifications in SciNapse typically range from several lines to a half a page, and the synthesized codes can be thousands of lines long. SciNapse is written in Mathematica code, and is about 120,000 lines long

SciNapse's high level problem specification language supports natural descriptions of geometry, mathematics and desired interfaces. The knowledge base includes coordinate free constructions (such as the Laplacian), equations (such as Navier-Stokes or Black-Scholes), discretization rules (such as Crank Nicholson), time stepping algorithms, solvers (such as preconditioned conjugate gradient and SOR). The system chooses appropriate data structures and generates a pseudo-code solution that it then translated into the desired target language. Mathematical code is optimized along the way.

SciNapse automatically refines a specification in a stepwise fashion from the most abstract level though several more concrete levels, finally creating a numerical code. After each stage SciNapse checks the problem state for consistency appropriate to that level of abstraction.

The code synthesis system is built on top of a general purpose knowledge based system written in Mathematica. The system includes an integrated object system, rule system and planning system. SciNapse objects explicitly represent common mathematical constructs such as a geometric region or part thereof. Objects also represent programming constructs such as a linear solver, a subroutine or a program variable.

SciNapse internal representation of numerical programs is independent of target language. In this abstract representation context dependent global optimizations are easy to implement. SciNapse generates codes in C and FORTRAN 77.

2.4.3 Networked Scientific PSEs

Many PSEs take advantage of networked architectures for the purpose of exploiting either parallelism in the target problem or distribution of the computing facilities. In some cases part of the computational problem is figuring out how to distribute the problem across computing resources so it can be solved effectively. In other cases, part of the problem is the selection of which resource to use. A PSE may contain a data base of networked computers only some of which are able to solve the problem at hand. This will be true in situations where the problem type submitted by the user requires software that is only available on a specific platform. Some PSEs are able to make judgements about which solvers, or which hardware platforms are best suited to solving a problem. Projects not discussed in this section but also of interest are [11, 45, 47, 51, 65, 64, 67, 75, 78, 86, 125].

Not all problems are amenable to distribution, and not all problems can take advantage of parallelism. For those that can, the constantly changing hardware and software base presents a complex set of configuration problems to the user. While the automation of distribution as part of a PSE may not be a mature science, it is a necessary one if parallelism is to be effectively exploited by everyday users. A collection of frameworks are presented in this section all of which address issues related to networked problem solving.

The SAMRAI Framework

The SAMRAI [172] (Structured Adaptive Mesh Refinement Application Infrastructure) Framework is a parallel data communications framework for structured adaptive mesh refinement multi-physics applications. Structured adaptive mesh refinement is an effective technique for focusing computational resources in numerical simulations

of PDEs that span a range of disparate length and timescales. AMR is used to dynamically increase grid resolution locally to resolve important fine-scale features in the solution. The goal is to achieve a more efficient computation. SAMR is a particular variety of adaptive mesh refinement where the locally refined grid is defined with structured grid components.

SAMRAI was developed to support a wide range of parallel multi-physics problems. The difficulties associated with implementing applications using SAMR design often makes the implementation prohibitive. Principle problems solved by SAMRAI include the handling of numerical methods for locally-refined grids and the management of data exchange. Data exchange patterns must be modified each time the grid changes.

Multi-physics application often couple different algorithmic components each of which provides a distinct part of an overall scheme. Users can easily describe data transfer phase of a computation by specifying communication operations to be performed, such operations include copying, temporal and spatial interpolation, and the application of user defined physical boundary conditions.

The SAMRAI framework represents a layer of automation and communication necessary for reducing the overall complexity of developing parallel application codes which take advantage of the adaptive mesh refinement.

NetSolve

The NetSolve Grid Computing system [13, 36] provides users with access to remote computational hardware and software resources. Grid computing describes a conceptual fabric of computing resources analogous to the electrical power grid, which ideally uniformly and seamlessly channels computational services to clients who plug

in to the grid. NetSolve's first motivation was to address the ease-of-use, portability and availability of optimized software libraries for high performance computing. The system uses a client/agent/server model.

Three major components are employed by NetSolve: the NetSolve agent, information service and resource scheduler, the NetSolve server, a networked daemon providing computational hardware and software resources, and the NetSolve client libraries which allow users to instrument their application code with calls for remote computational services. NetSolve provides a functional programming model based on RPC in which the client is used to pass NetSolve objects to and from services as inputs and outputs. NetSolve supports objects like, MATRIX, a 2 dimensional array, SPARSEMATRIX, a two dimensional array stored in compressed row storage format, VECTOR, a one dimensional array and other similar structures. The NETSOLVE client supports both synchronous and asynchronous calls. NETSOLVE currently supports APIs for MATLAB and MATHEMATICA environments. NetSolve enhances these environments by expanding the numerical functions available to the user and allowing for increased performance by executing code remotely on more efficient machines.

NETSOLVE has incorporated a large number of solver algorithms from a variety of packages like BLAS, LAPACK, ScaLAPACK, ItPack, PETSc, AZTEC, MA28, SuperLU and ARPACK. NETSOLVE input routines can analyze user input and intelligently select algorithms depending on input data characteristics.

Net Pellpack PSE Server

Pellpack [85] is a PSE for PDEs, Net Pellpack [36, 115] the software's Web-based counterpart, lets users solve complex PDE problems with a graphical user interface, a

stateful text based protocol, and Net Pellpack servers running on machines anywhere on the network. The main design objective was to provide the Pellpack GUI to remote users in an effective secure and efficient manner.

There are several possible design scenarios for a web based PSE. The first approach is to make the whole PSE available over the web, where the web based GUI drives all aspects of the PSE. This is usually only feasible over a high speed LAN connection. The second scenario uses a networked software bus to created virtual libraries by distributing the library to multiple service providers. Communications technologies like remote procedure calls or CORBA may be utilized to enable the network connections. Net Pellpack utilizes these two fundamental design procedures. Once the user has selected a solution path through interactions with Net Pellpack, either library software modules are downloaded from a repository and used locally, or the problem is sent to a computation server with an implementation of the algorithm. Net Pellpack automatically decides for the user which approach is most appropriate depending on the users problem description. Net Pellpack library interfaces follow a standard so that bodies of mathematical software can be developed and maintained for a wide variety of computer systems.

2.4.4 Collaborative PSEs

One common trend among PSE designers is to build PSEs with the purpose of enabling multiple users to collaborate on a given project. Collaborative PSEs go beyond simply distributing a PSE across a network as described in the previous section. An attempt is made to construct an environment for the cooperative solution of a problem for a broad group of individuals who may be physically separated. Collaborative PSEs make the work of sharing engineering design and simulation results between developers

more manageable. Networking technologies are normally employed, much use is made of the Internet and world wide web. Some projects focus on sharing simulation results, other projects focus on the sharing of computational resources. Collaborative PSEs focus on the processes by which scientists interact, and provide tools for bridging vast distances that often separate specialists who wish to work on the same problems. This topic is also discussed by [23, 55, 62, 165].

PNNL's design for CPSE

At Pacific Northwest National Laboratory (PNNL) the design of collaborative PSEs for scientific computing in various domains is being studied [97]. PNNL's project seeks to characterize the nature of scientific problem solving and searches for innovative ways to improve it. The ultimate goal is to allow scientists and engineers to enhance their collaborative problem solving capabilities through the improved and integrated usage of resources and tools.

Cognitive researchers describe the act of reasoning or problem solving as a higher order skill that encompasses specific processes and abilities. Problem solving occurs in the context of the activities that scientists perform and the knowledge that they possess. Providing access to computational resources is not enough, rather engineers need support for how they utilize domain knowledge.

PNNL's project team met with 5 different groups of scientists and engineers consisting of computational chemists, regional climate modelers, nuclear magnetic resonance experimentalists, automotive engineers and fluid dynamics modelers. Through an interview process, several common problem solving needs were determined.

1. Easy and effective access to computational resources. Resources should be represented in a way that is comprehensible and intuitive to the domain engineer.

2. Experimental design and execution support. Better tools are needed it assist in defining, managing, executing analyzing interpreting and sharing experiments.

3. The ability of scientists to solve problems hinge on their knowledge of domain concepts and theories. By making knowledge explicit and concrete, scientists may be able to better maintain and evolve this knowledge.

4. The experimental process is highly repetitive, tools are needed which support a repetitive cycle while allowing the modification of initial conditions and comparison of generated output.

Domain scientists and engineers do not naturally think of computational resources as applications, computers and files but rather as models, calculations and spatial and temporal data. PSEs need to be designed to promote the appropriate level of abstraction such that scientists may utilize these resources in a form consistent with their specific domain concepts and views.

Scientific problem solving is inherently a collaborative effort among researchers as they share information, models, tools resources and results. More than just sharing specific research artifacts, scientific problem solving also involves the sharing of one's expertise and experience. As scientists run computational models, they apply a vast amount of procedural and domain knowledge. Scientists may have valuable experience in running particular computational models. The ability to capture this kind of knowledge and share it with others is the goal of the collaborative PSE. Scientific collaboration does not occur in isolation but is driven by the functions of the scientific research.

DLR's VirtualLab

The goal of DLR's VirtualLab [58] is to provide Web access for electromagnetic scattering and radiative transfer simulation applications developed at DLR's remote sensing technology institute. All scientific components provide coarsely unified Web based user interfaces supporting data input, execution steering, and output. Documentation is integrated through Web hyper-links providing context sensitive on-line help. Users can retrieve components based on search keywords matched against meta-data that are part of the components documentation.

Each user has a personal area where all data resulting from work with the VL are stored along with the selection of components the user is working with, and all experimental simulation data results. Each experiment started by the user launches the task manager which constructs a job control file invoking the necessary configuration of the components. This system supervises the VirtualLab cluster's compute nodes and schedules job execution using a load balancing strategy.

The web interface is tailored towards interactive command line applications. These applications operate in batch mode but can accept various structurally different sorts of input data sets. The VirtualLab offers a mechanism for abstractly describing all the relevant details of the applications input behaviour so that the VL can provide a responsive dynamic user interface.

The virtual laboratory is used to execute a variety of applications of interests to DLR. The focus is on scattering codes which are used to study light scattering optics on various classes of nonspherical particles such as irregular ice particles and Chebyshev-like particles. The individual applications include Mieschka, Pmieschka, CYL and QCACP.

Cactus Computational Collaboratory

The Cactus Collaboratory toolkit for solving PDEs was originally designed to simulate Einstein's equations for studying black holes, gravitational waves and neutron stars, and has more recently been adapted for use in bioinformatics and atmospheric sciences [8, 9, 31]. The system also provides scientists without a knowledge of parallel computing or mesh refinement with a simple framework for solving any system of PDEs on many parallel computer systems.

Cactus applications are built from a meta code which describes how applications in common computational languages, such as C, C++, FORTRAN 77, and FORTRAN90 interweave. Parallelism and portability are achieved by hiding MPI, the I/O subsystem and the calling interface under a simple abstraction API. Preprocessor macros implemented through make files and Perl scripts expand preprocessor macros to construct the arguments of the flesh and additional arguments defined by each thorn. Cactus is thus a meta-code, the user specifies a desired code and the system automatically generates the code containing only those routines requested.

Cactus takes advantage of emerging grid technologies. Although distributed resources offer many advantages there are downsides as well. The enormous terabyte data sets produced by the Cactus simulations tax bandwidth limits. Even with the best available international networking resources, downloading the data from the simulation run may take longer than it took to run the simulation itself. These problems have motivated many remote monitoring and steering efforts.

The ASC Portal is intended to deliver a collaborative simulation management framework for generic applications, with the development driven by a particular community of astrophysicists, numerical relativists and computational science researchers that use and develop their codes with Cactus. This community makes up a virtual

organization denoted as the ASC-VO. The Collaboratory enables a wide spectrum of researchers in the community to cooperate on code development and use. This has the effect to drastically increase scientific productivity by fostering collaboration, cutting down redundant efforts by different research groups, and maximizing the benefit of massively parallel computing to the community.

2.5 Discussion

This chapter has presented a broad array of topics and perspectives. Fundamental physical models and solution techniques are as important to the development of this thesis topic as a bird's eye perspective on some of the most ambitious simulation projects developed to date. One of the difficulties in designing this thesis topic was the effort required in balancing a specific problem domain which encompasses a fundamental set of questions against a suitably flexible methodology which might yield some general insights and perspectives on the state of the art in engineering science today. The proposition of studying a generic modeling system, as outlined in the original thesis proposal, is quickly rebuked by the obvious counter suggestion that many such projects already exist. Yet, what should be clear from the brief survey of projects presented in this chapter is that no matter how generic and flexible they claim to be in each case there is some fundamental perspective that drives the design of any given PSE. No PSE is a panacea of solutions to all problem types. The literature which describes computational science and engineering solution methods is quite vast and for a single tool to claim that it has addressed every possible technique is unfathomable.

Instead, new projects continue to explore various avenues by applying themes common to already existing PSEs in their implementations to novel design issues and

problem types. While the problem types and solution techniques presented in the first part of this chapter can be addressed or taken advantage of by many of the tools described in the section on PSEs, each of the PSEs described supplies its own particular perspective.

The topics discussed in this chapter supply the necessary background and motivation for subsequent chapters. To the best of our knowledge moving mesh frameworks have not been studied in the context of modeling fuel assembly and control rod motion. PSE examples discussed in section 2.4.1 and section 2.4.2 have provided overall design motivation. The flexibility of projects like SCINAPSE and CTADEL and their use of computer algebra systems as part of their model definitions have motivated certain aspects of the MOOSE design. The papers discussed in section 2.3.2 come closest to addressing the issues related to moving meshes. The Overture project and other papers by the same authors which modeled fluid mechanics problems using overlapping grids provided many insights as to how the MOOSE's grid connection algorithms should be defined. The algorithms discussed in the reviewed literature on linked meshes referred specifically to hyperbolic fluid mechanics problems, so these techniques can not be applied directly. However, as will be discussed in the next chapter, they provide the basic motivation for the MOOSE's mesh linking algorithms.

Many of the techniques presented in this chapter are not standard practice in nuclear engineering, and cannot be directly applied. Nodal methods, discussed in section 2.3.3, are the standard techniques currently used to model moving fuel assemblies. Although the authors who discuss nodal methods recognize the possibility of using moving meshes as an alternate strategy for modeling motion, to our knowledge, no attempt to do so has been undertaken. One author [177] criticises moving meshes as being both too complicated to implement, and if implemented too inflexible.

To address flexibility and complexity of implementation, the MOOSE moving mesh

is built upon a general computer algebra system able to implement the neutron diffusion equation discussed in section 2.1.3 under any of the implicit integration schemes discussed in section 2.2.3. Not only are the MOOSE methods mathematically expressive, but in order to solve real world problems high performance sparse linear and eigenvalue solution techniques discussed in sections 2.2.1 and 2.2.2 are taken advantage of. The MOOSE framework employs a code generation system which is able to bridge the gaps between abstract problem representations and high performance numerical solvers. The code generation mechanisms, associated solver libraries, and mesh linking rules are discussed in the next chapter.

Chapter 3

Implementation

3.1 Introduction

A framework is a reusable design of all or part of a software system described by a set of abstract classes and the way instances of those classes collaborate; this presentation of the MOOSE highlights abstract features typical of Problem Solving Environments and lays out a conceptual road map for subsequent work. For the purpose of testing and illustration certain elements of the MOOSE framework have been developed. This chapter will present some features of the prototype's implementation. The entire code base for the MOOSE is quite large: at over 45,000 lines of original computer code (900 pages) it cannot be presented in its entirety in this thesis.

The MOOSE framework is broad enough to capture a variety of physical phenomena in the modeling of steady state, and transient finite difference models. The focus has been on elliptic and parabolic problems which can be represented on a non-conformal patched Cartesian mesh which permits motion in two dimensions. As a general framework there is certainly room for expansion into other categories of physical problems.

The MOOSE framework supports the following principle components:

- User Interface
- Graphical output
- Mesh Construction Algorithms
- Symbolic Problem Representation
- Interfaces to State of the Art Linear and Eigenvalue Solvers

The MOOSE prototype has employed a variety of advanced implementation techniques:

- Mix of implementation languages, including C, C++, MAPLE and various scripts
- Code generation
- Symbolic Processing

The prototype implementation has focused on translating the user supplied symbolic representation of a problem into an efficient matrix generation program. The matrix generator, created by the MOOSE, rapidly builds a collection of sparse matrices and vectors using standard data structures which are compatible with various high performance numerical libraries.

For the purposes of writing a thesis several limitations were imposed on the development of the MOOSE to keep the project manageable. The user interface which was developed is quite simple. The MOOSE framework may be suitable for a variety of problem domains but only reactor physics problems are examined in detail in subsequent chapters. Geometry in the MOOSE is limited to rectangular two dimensional meshes and problems which model motion in vertical or horizontal directions.

3.2 Model Design

This section presents the conceptual breakdown of physical models used by the MOOSE from both a terminological and relational standpoint. At the same time the following set of terminology is generic enough that it can be applied to other problems as will be discussed in chapter 4.

Simulations are constructed from a collection of individual cells. A cell is the basic unit of the MOOSE's simulation. A cell depends on the definition of three other principle structures:

- A set of variables
- A set of constants
- A physical linear equation which relates the variables and constants

The set of variables can be defined by the user and it includes whatever the user is interested in modeling. Typical cell structure variables for a nuclear simulation might include:

- Temperatures
- Rate of Flow
- Flux Density
- Precursor Density
- Fuel Burn-up

Variables can be represented as vectors. In the case of flux density subscripts can represent various energy levels, although precisely how this representation is accomplished is left to the model designer. Zero, one or two subscripts are supported by the MOOSE for any cell variable.

A cell also has associated with it a set of constants. Constants are assumed to have some spatial variation in the represented artifact. For example, there is no advantage to associating the speed of light with a cell. Such universal constants can be specified independently of the cell definition if they are to be uniform throughout the simulated geometry.

Each cell also has associated with it at least one equation, which typically will be a partial differential equation represented in finite difference form. There is no limit on the number of equations that can be represented in a cell. Spatial references are limited to cell neighbours and are handled by special operators which are supplied by the MOOSE for finite difference approximations to first and second derivatives. Users may define initial conditions for time integration problems, or boundary conditions of any type for steady state problems. The MOOSE can be used to solve linear problems of the form of $Ax = b$, standard eigenvalue problems $Ax = \lambda x$ or general eigenvalue problems $Ax = \lambda Bx$.

Maps are collections of cells that specify the relative position of each cell within a map. The MOOSE only supports Cartesian maps, but allows map definitions to be nested and repeated under certain circumstances. This allows a user to define a geometrically complicated structure, and then repeat that structure in the context of a higher level map.

Motion is achieved by the MOOSE through the relative motion of maps. A MOOSE simulation can build a sequence of interdependent solutions. Each step in the sequence can involve the translation of a map or redefinition of a cell. Simulation

steps can refer to each other as determined by definitions imposed by the user.

3.3 MOOSE Framework

This section presents the basic elements of the MOOSE's framework describes how each element connects with its neighbour. Section 3.4 and section 3.5 will discuss in more detail some of the practical problems encountered during the construction of the prototype. Some of the details presented in this section are a necessary consequence of fundamental design choices, for example, the systems of interest are always represented by sparse matrices so only sparse numerical libraries are discussed. Others design details were a matter a choice, in some cases with the goal of minimizing implementation effort, for example the reduced and simplified text interface.

3.3.1 Framework Overview

The MOOSE is a reusable framework for the construction of programs that can model various simulation scenarios involving moving components. As a framework it presents a collection of abstract classes which can be concretized by the user. The MOOSE supplies the user with a collection of run time libraries to link their simulation against, some of which are external mathematical libraries, others are MOOSE application specific. In addition to supplying the user with libraries the MOOSE also supplies the user with several executable programs for simulation configuration file editing and code generation. The major elements of a MOOSE simulation are pictured in Figure 3.1.

The user interface discussed in section 3.3.2 provides the main configuration portal to the MOOSE. The user needs to supply cell definitions, map definitions as well as a simple C program that directs the execution of the simulation. Each of these

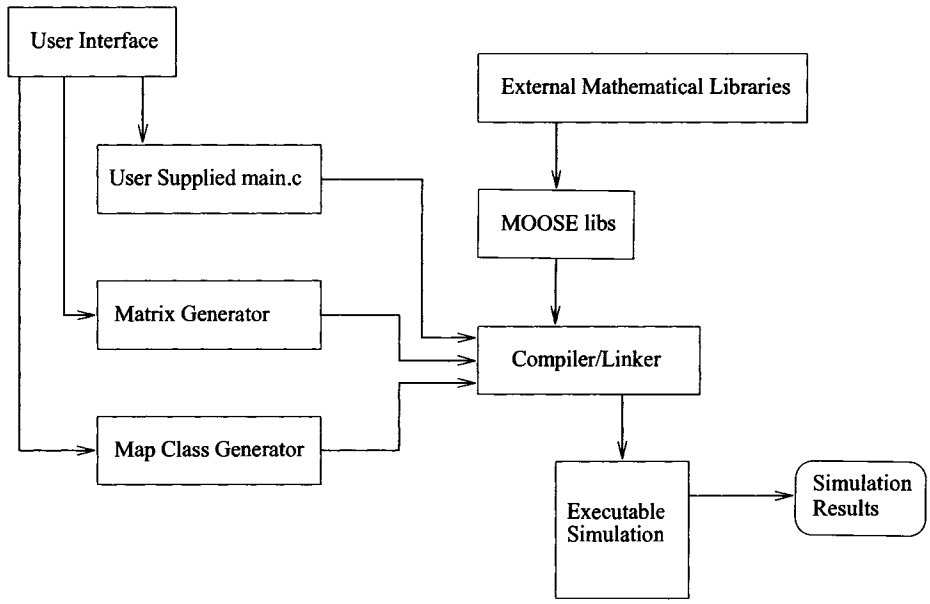


Figure 3.1: MOOSE Framework Overview

elements is organized through the user interface. The matrix generator is built from a collection of MAPLE scripts. The matrix generator reads configuration files which are generated by the user interface. The matrix generator is responsible for extracting the partial differential equation definitions from those files and building a C program capable of interfacing with the solvers upon which the MOOSE is based. A matrix generation function is required for each data structure type and each partial differential equation set. Maps which share PDE and data structure types will use the same matrix generation function to build their partial representation of the sparse matrix. Multiple PDE definitions are possible, as well as multiple data structure definitions, so one simulation model may require many matrix generation functions.

Each MOOSE map is defined as a C class which inherits its major functions from a parent class defined as part of the MOOSE libraries. Each map subclass contains some specific functions which are particular to the data structures represented by the

subclass. The map layout is stored in the subclass functions. The generic functions which can be applied to any map class are defined as part of the map base class.

The MOOSE libraries provide user level access to both the mathematical libraries as well as simplified interfaces to the MOOSE data structures. The MOOSE libraries also provide an interface between the matrix generator and the external libraries. Additional details which describe the MOOSE's mathematical library interface are presented in section 3.4.1.

The MOOSE libraries give the user simplified access to each map. Maps can be identified either by name (for example "grid_map"), or by coordinate position. While a variety of automatic graphical output options exist as part of the MOOSE package as discussed in section 3.3.4, the user can also extract individual floating point numbers from the simulated mesh or specific eigenvalues. Access functions which request vector minimums and maximums are also available as well as functions for manipulating specific mesh elements as described in section 3.3.3. In principle any result generated during the solution process can be extracted from the MOOSE's fundamental data structures. The matrices, vectors, or any solution vector value may be extracted. Using the MOOSE data access routines puts vector values in context and interpolates between values if necessary.

3.3.2 User Interface

Developing a robust and well designed interface is a complex task and was not the focus of this thesis. However, in the spirit of a PSE development project it was felt that at least a very simple prototype interface was necessary.

The MOOSE defines various input parameters through the use of a collection of configuration files. In principle a MOOSE simulation only needs a text editor to set

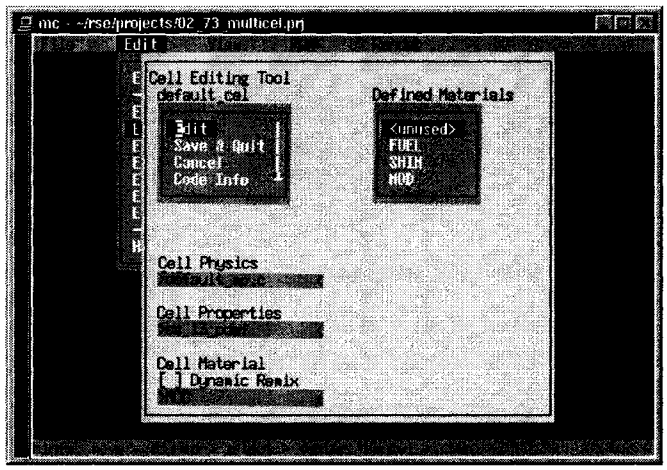


Figure 3.2: Cell Editor

up its various components.

For example cells are defined by a simple structure file which looks like this:

```
rs_cell MOD ss_13_pdef mphy {
      "default_mp.c"
} default_cel;
```

The MOOSE code generator interprets this declaration and reads it as a sequence of fields. The first field `rs_cell` identifies this as a cell declaration, the second field notes the materials structure to be used, the third field notes the property definition, the quoted name within the parenthesis contains the file name which the MOOSE will use to define the partial differential equations, and finally the name of the cell is given in the last field.

The prototype for the MOOSE includes a cell structure editor which allows the user to enter each of these fields with a little guidance. For example the cell editor will provide a list of names of valid material structures or property definition structures on request.

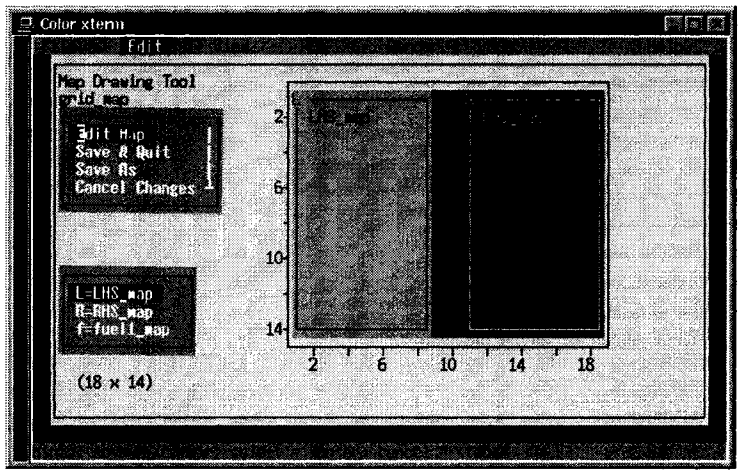


Figure 3.3: Map Editor

Maps are defined by text files. The grid map is understood to be the top level map in any simulation, much like the main() function of a C program. Since maps are hierarchical in nature only the lowest level maps are defined in terms of basic cells.

The prototype MOOSE map editor renders the configuration file in a natural looking way. Each individual map is assigned a color to help distinguish it from its neighbours. Submap names appear in the editors window if the region is large enough to accomodate the text, otherwise a single letter is used to lable the region. The editing session shown in Figure 3.3 decomposes the geometric problem into a sequence of adjacent vertical strips. Each map shown in the editor's window defines its own sequence of submaps. The submaps terminate in a map based on a simple cellular definition.

Every MOOSE simulation requires a short code segment. The user supplied code which accompanies a MOOSE simulation can specify a variety of precise geometric details at run time. For example, if the exact dimensions or position of the simulation components cannot be reasonably represented by the user interface they can be

adjusted during model execution.

The prototype user interface includes 6 main menu with various submenus headings:

- File
  - submenus: Open Project, Close Project, Delete file/output, Recover Deleted File, Reload libraries, Directory Editor, Set Project Read Only, Quit, File Help
- Edit
  - submenus: Edit File, Edit Map, Edit Cell, Edit Structure, Edit Materials, Edit Physics, Edit C source, Edit Font, Edit Help
- View
  - submenus: Compilation Warnings/Errors, Configuration Errors, Map Dependency Graph, Run Time Errors, Run Time I/O, View Help
- Run
  - submenus: Make all and Run, Make sim.conf, Make C++ Source, Make sim Executable, Force Full Rebuild, Run Sim, Debug Crashed Sim, Run Help
- Render
  - submenus: Show Output URL, Render Help
- Help

– submenus: About, Contents, Keyword, Man page

Each menu item is active in the prototype. The File, and Help items are relatively self explanatory and follow typical designs in other programs. The Edit menu options launch various structurally specific editors. The cell and map editor have already been discussed. In addition the prototype supplies a structure editor, a materials editor and a font editor. The Run menu provides various code generating and execution options.

The prototype editor was given only a low priority in terms of development. Although it comprises about 25% of the entire code base (11,000 lines), building it consumed less than 10% of the total development time since it was the least sophisticated element of the framework.

3.3.3 Operations for Moving Model Components

For model submeshes to be moved several fundamental operations on the relative position and proportion of mesh components are required. Within the prototype these operations are implemented as functions that can be called by the user after the initial mesh is built with the interface, but prior to generating a solution. Within a more developed PSE these operations could be built into a detailed graphical user interface, which could infer their use through a sequence of positions that the user specified for the model. The current user interface is not sophisticated enough to support the specification of a sequence of motion points.

Five operations are required for moving model components:

- Move Grid to absolute position
- Move Grid Relative to current position

- Rescale Grid
- Get Grid Position
- Remesh Grid

Since meshes within the MOOSE can be nested hierarchically each of these operations, when applied to a given mesh, will also be applied to each of its submeshes. This is helpful since it allows the construction of components from collections of submeshes which will behave in an expected way when an operation is applied to a parent.

Two operations for motion are provided, one which takes absolute positions in the overall parent mesh, and a second one which moves components relative to old positions. User functions may need to link the movement of otherwise unconnected mesh components. To facilitate this a function which reads grid positions is also provided.

Rescaling a mesh changes its dimensions, either its width or length, or both. Rescaling operations may be required to represent a variety of zones within a simulation which must gradually recede to allow a simulation component to move into a new geometric space. Squashing or stretching a region by only small degrees will have a minimalistic impact on a simulations if the region is continuous in terms of its material and PDE definitions.

It is always possible to redefine the mesh density for either the entire geometric definition, or at certain sub blocks of a given simulation. Increasing the mesh density either locally or globally is an important operation because it allows the user to make certain decisions regarding the overall precision of the model. In the validation chapter several models are tested at various ranges of mesh densities. Models may also require very localized remeshing. Being able to specify which areas of the model require extra

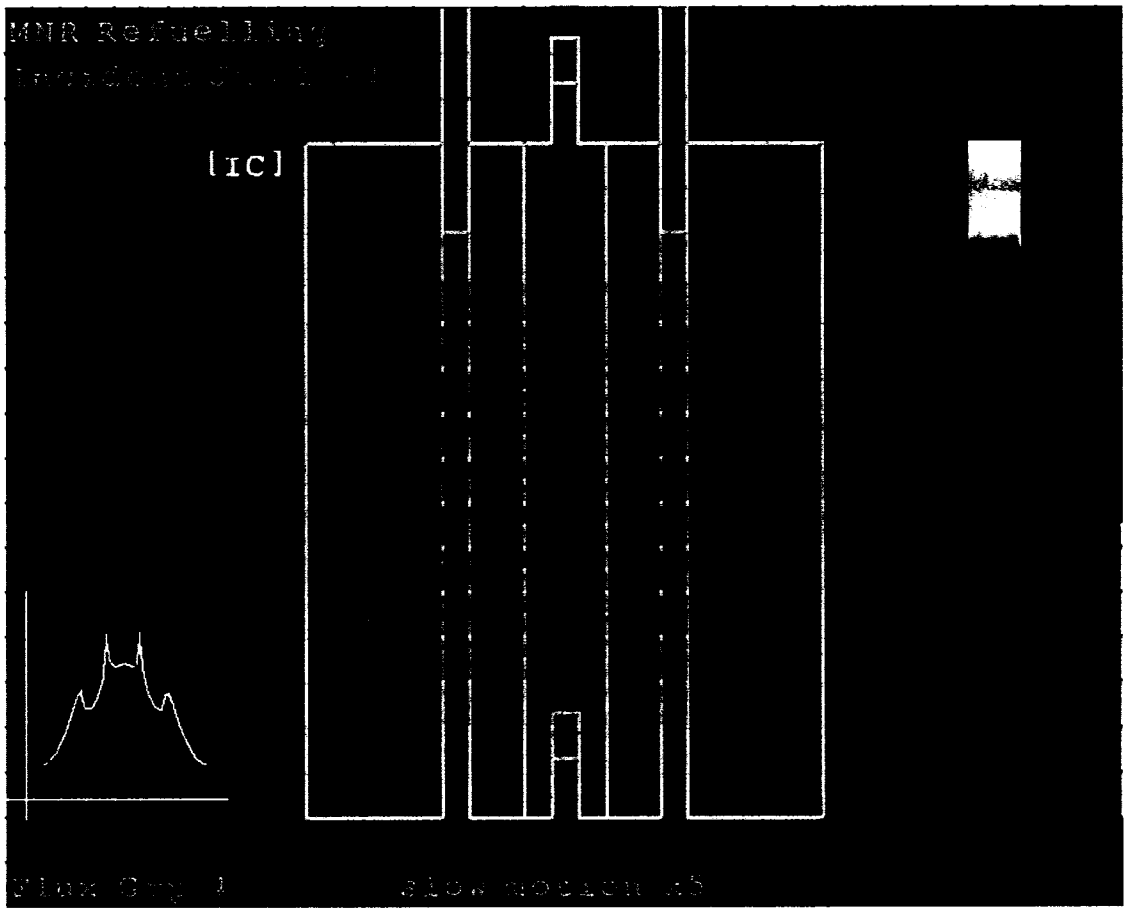


Figure 3.4: Example Thermal Plot

mesh points allows the user to focus the mesh on specific areas of interest or areas which are known to be troublesome.

3.3.4 Output Options

The MOOSE provides a variety of output options which automate the generation of graphics. The MOOSE's output library generates line and surface plots which are embedded in HTML files and placed in the users public_html directory, organized by project and simulation run. Several output graphic styles are supported, the thermal

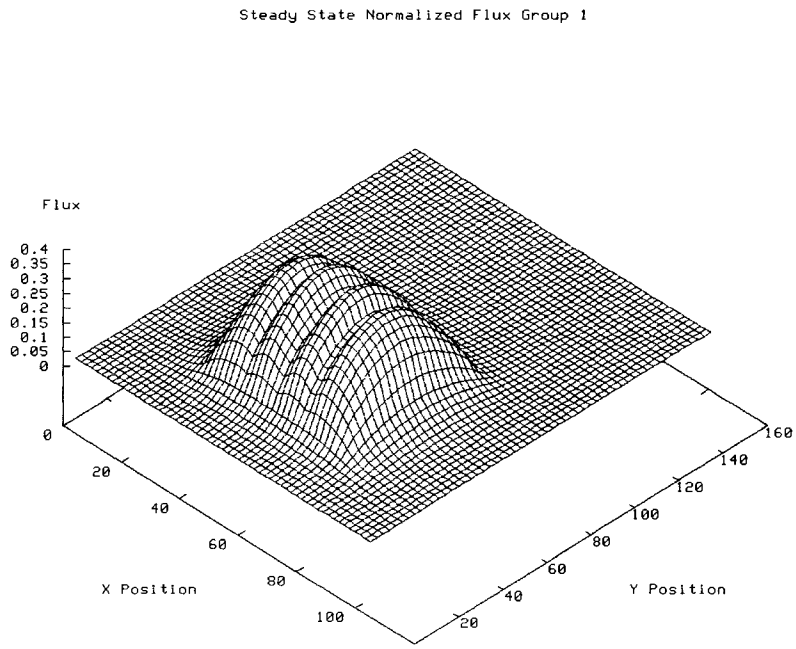


Figure 3.5: Example Surface Plot

and cell styles are provided by the MOOSE framework. The MOOSE is also able to spawn other plotting packages. GNUPLOT can be used by the MOOSE to generate a broad range of additional plot styles. The functions which link GNUPLOT to the MOOSE could in principle be easily extended to support other packages.

Figure 3.4 shows an example thermal plot for the rod insertion problem. This plot renders flux levels for the lowest energy group in a four group model as colors, a legend beside the graphic gives an indication of the scale. This plot shows where the cell boundaries are (light grey lines) as well as where the material boundaries are (white lines). Locating the material boundaries and the cell boundaries is important for model debugging and construction. The various labels and legends surrounding the plot are generated by the MOOSE's line and text drawing packages. Transparent frames are placed under the text so that as the thermal graph changes colors the

text remains visible. The mini-graph plotted in the lower left of the figure shows the shape of the thermal graph plotted as a unitless line graph for comparison with the thermal graph. This figure represents a single frame from a transient simulation discussed in more detail in chapter 5. The slow motion label refers to the frame rate, this simulation was rendered at 20 frames per second in a standard .AVI file. During the excursion 100 frames were rendered for each second of simulation time, when replayed at 20 frames per second this produced a slow motion effect in the video. The simulation computes many more steps than are necessary for the animation, this is also discussed in more detail in Chapter 5.

A surface style rendering of the same data generated by GNUPLOT via the MOOSE is illustrated in Figure 3.5. GNUPLOT offers a variety of other plot mechanisms many of which are also incorporated into the MOOSE including line plots and contour plots. The MOOSE also provides a simple mechanism to include GNUPLOT commands so that if the user is familiar with GNUPLOT additional labels and other graphical details provided by GNUPLOT can be incorporated.

3.4 Implementation Languages and Methodology

The MOOSE is written in several different languages and uses a variety of libraries, script interpreters, and helper applications to allow it to accomplish its simulation objectives. In the last 20 years a wide variety of programming languages and methods have come into common use. Along with very common and well established languages like FORTRAN and C, several methodologies have come into popular usage like parallel programming and object oriented programming. Advanced techniques like code generation and symbolic manipulation have also become more common in recent years. The MOOSE borrows from many different paradigms in the attempt to

satisfy a wide variety of needs. To some extent the implementation of the MOOSE has been undertaken in an experimental fashion. Some aspects of the MOOSE are more successful than others in this regard. This section will briefly touch on some of the more unusual aspects of the design of the MOOSE.

3.4.1 Library Interfaces

The MOOSE relies on several numeric solution libraries to generate solutions to the problems encoded by the user. Two basic assumptions are made: the problem is always assumed to be sparse, and the problem is always assumed to be governed by linear relationships.

PETSc [19] is a well known linear solver developed at the Argonne National Laboratory by the Mathematics and Computer Science division. PETSc can operate in either uni-processor or multi-processor modes. At its core it provides the user with a variety of methods and data structures for representing sparse linear matrices and can generate solutions to those matrices through a collection of well understood algorithms, most notably Conjugate Gradient and GMRES.

SLEPc [80, 81, 82] is an eigenvalue solver which is based on PETSc. SLEPc uses the data structures and software design methodology of PETSc for the solution of eigenvalue problems of both the standard $Ax = \lambda b$ and the general $Ax = \lambda Bx$ varieties.

Although these 2 solvers have proved to be the most useful, the MOOSE's sparse solver interface is not tightly tied to either package. The MOOSE only requires a package which supports any rough variation of the following set of basic function calls:

```
Initialize_Solver()
```

```
Matrix_Create(size)
Vector_Create(size)
Matrix_Add(row,col,element)
Vector_Add(col,element)
LinearSolve(Matrix,vectorB,vectorX)
EigenSolve(MatrixA,MatrixB,vectorX)
```

The generated code makes calls to these simplified virtual functions. A software layer exists between the generated code and the library package to facilitate these calls. PETSc requires several steps in the creation of a matrix, so the MOOSE has a special version of Matrix_Create() which handles all of the calls required by PETSc. In this way the MOOSE is solver package independent. The more rudimentary package LASPACK, can also be used by the MOOSE, to enable this package the appropriate flag is set prior to compilation, and when the fundamental functions are called the MOOSE uses basic function calls appropriate to LASPACK.

As a pair, SLEPc and PETSc are quite flexible so most of the development of the MOOSE was driven by their solution strategies. SLEPc and PETSc are themselves based on other packages and can also act as interfaces to other packages. SLEPc and PETSc are based on the dense linear solver packages LAPACK (note that LASPACK and LAPACK are unrelated) as well as a generic BLAS package. Hardware specific versions of LAPACK and BLAS are available. The MOOSE was developed with a high performance platform independent BLAS known as ATLAS (Automatically Tuned Linear Algebra).

The most recent version of SLEPc includes a Krylov-Schur solution method [153]. The SLEPc authors currently recommend the Krylov-Schur method as the default choice. Recent experimental tests with the MOOSE confirm that this is the best

choice. PETSc provides interfaces to a variety of other packages. SuperLU [48] is a sparse direct linear solver that also has a distributed version, SuperLU_DIST. SuperLU can be transparently built alongside of PETSc, and provides a direct method for inverting sparse matrices.

The direct solver often found solutions to problems much faster than the iterative solver and its results were often more precise, although the direct solver tended to use larger amounts of memory than the sparse solvers. Both solution methods proved to be useful in generating final results. In some cases, especially the steady state studies where high degrees of precision where essential, the direct solver was the most useful. In transient cases the iterative solver seemed to be more useful.

3.4.2 Merging MAPLE, C, FORTRAN and Other Languages

The MOOSE is built from several different languages. Each was selected for its dominance in a particular domain and its applicability to a particular development goal. Because the MOOSE aspired to be both capable of solving very computationally demanding problems as well as flexible and malleable, no single language seemed appropriate for its implementation. The MOOSE interfaces to a variety of external libraries. Some are written in C, some in C++, and some in FORTRAN. Most of the solving power of the MOOSE is leveraged from PETSc and SLEPc, which themselves are built on other libraries. Legacy FORTRAN77 solvers like ARPACK [166] are accessed via SLEPc's interface. To a large extent the developers of PETSc and SLEPc have seamlessly merged their libraries with the underlying FORTRAN code. While it is of some interest to recognize the important presence of FORTRAN libraries underneath the MOOSE, FORTRAN code is completely hidden by the libraries which utilize it.

The MOOSE's user level programmer interface is written in C and MAPLE. The basic text GUI is a C++ program which launches the various code generators, interpreters and compilers. The MAPLE matrix generator acts as the equation and data structure interpreter for the MOOSE. MAPLE is typically thought of as a symbolic processing language for solving certain continuous integrals in calculus. MAPLE was chosen for this project because it also provides a powerful scripting language which is adept at symbolic manipulation. Because MAPLE is a scripting language, the size of a task that can be solved with a MAPLE script is somewhat limited, and so the MOOSE cannot use MAPLE to analytically or directly solve the supplied system of equations. Instead the MOOSE uses a program written in MAPLE, the matrix generator, to translate a problem expressed in a symbolic continuous representation into a C++ program capable of generating a matrix which approximates the continuous problem in a finite differenced form. The generated program is compiled by gcc, and a special function is constructed for each simulation which can be called repeatedly. This function must be defined before simulation compile time.

Although the equations and data structures as specified by the user are fixed once the execution of the simulation begins, various simple parameters can still be adjusted; the coefficients of diffusion, the size of the region that is being solved for, or the magnitude of a time step. The user is allowed any number of equation definitions prior to simulation execution via the equation group parameter. So if a physical model requires a sequence of applications of alternating representations, say for example in a Black/White style Leap Frog method, this can also be accomplished. The MOOSE uses a C++ hierarchical object oriented representation of the model for the purpose of matrix generation and overall data organization. The user accesses top level object methods through a simplified C interface. While the MOOSE is written in C++, an understanding of C++ is not required to be able to construct models with the

MOOSE. Similarly PDEs expressed in MOOSE syntax require only the most rudimentary understanding of MAPLE's conventions. Where possible, syntax conventions for the MOOSE's interpretation of PDE's was selected so that it would appear similar to C.

3.4.3 Code Generation

Code generation [22, 28, 29, 37, 43, 44, 59, 63, 83, 101, 108, 144, 168] as a technique overlaps to some degree with compilation. The principle differences between the two methodologies are normally found in scope and intention. While a compiler takes the symbols from one general purpose language and converts them into the symbols of another general purpose language (for example a program which converts C into assembly), the role of code generation is more oblique and often very task specific.

Code generation will often imply a certain trade off involving complexity size or speed. It can be a very complicated option to choose, so the advantages and disadvantages should be carefully considered before it is undertaken. Consider a vector-based drawing package where a user draws shapes with connecting arcs and lines by specifying vertices with a mouse. An obvious way to record the image would be to simply save this list of vertices and note their connection order. Reconstructing this image would require the use of the same general purpose program which saved it.

If the drawing program used code generation to save its files, instead of simply saving the list of vertices, it could save in source code a sequence of function calls necessary to redraw the image. In this way the image could be transmitted to another user who did not have the same general purpose data file reader. The result of the program generator might be smaller, simpler or faster than the general purpose data file reader. The generated program would not be smaller than the simple data file

which stored vertices, and the code which built the generated program would also be much more complicated to implement than the code which simply saved a list of data points.

While code generation is intriguing as a program design methodology it has a wide variety of associated pitfalls that need to be avoided. The biggest problem with code generation is that while the code generator may seem to be generating code without any difficulty, fixing a bug in the code generator's progeny is not so simple as starting up a debugger and looking for mistakes. Any flaw in the child is corrected by carefully studying the parent. The notion of code generation as an option for program developers is a relatively recent one and as such there are few tools available to assist with the task. The MOOSE's code generator is built with large collections of simple formatted print statements. Lines of C code are constructed as strings by the MOOSE's code generation modules and written to files for later compilation.

Code generation became necessary for the MOOSE framework because while MAPLE had general flexibility in terms of symbolic manipulation, it lacked the execution speed necessary to generate large matrices. PETSc and SLEPc, the solvers of interest, provided C++ interfaces only. While the matrices of interest could certainly be hard coded in C++ for specific problems, given the low level and both rudimentary and strict variable typing system used by C++, the job of manipulating data structures and equations was quite onerous.

Code generation then became the bridge between the two languages. The matrix generation program builds a naive but very fast function compatible with the MOOSE C++ libraries. This function can be compiled and called by the MOOSE and used to construction matrix vector pairs in a data structure format which was compatible with the numerical solvers. The function generated by the matrix code generation program is flexible enough that a variety of model modifications can be done without

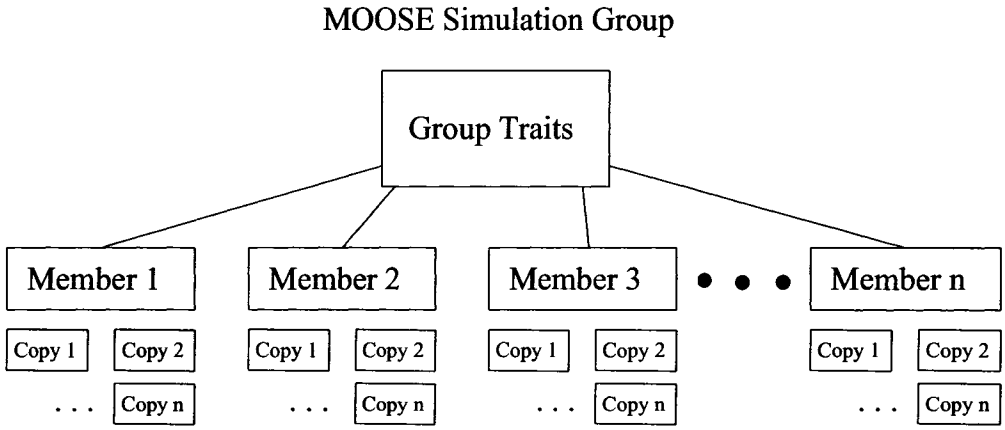


Figure 3.6: MOOSE Group Structure

regenerating the code for constructing the matrix. Some example code generated by the MOOSE is presented in the Appendices 3, the PDEs which were used to generate this code are presented in Appendix 3.

3.4.4 MOOSE's Simulation Groups

A single MOOSE simulation project can be configured as a group of related problems which share the same main traits characterized by the geometry and control software. Each simulation group member may however express its own specific traits in terms of which data structures, partial differential equations and solution techniques it applies. A simulation project may simultaneously define multiple group members prior to compile time.

Each group member uses the same geometry. The simulation group defines an initial position for all moving components, as well as provides a concrete name for each cell grouping within the geometry for all group members. Group members differentiate themselves by using different definitions for the collection and arrangement of cells which defines the group. Some traits, in particular those which determine the precise

choice of PDEs for a given data structure, can be expressed selectively at run time. Recall that the cell defines data structures, materials, and PDEs used at each location. Other traits, like for example the size of the data structures used to represent the model, must be chosen prior to run time.

Each group member uses the same general cellular structure of the group. For example, a group of simulations might all define the central region of the geometry as occupied by cells named "core". Exactly which definition for the core cells is being used will be determined by which group member is selected. A very simple definition might neglect certain parts of the physical model which other members could include.

Simulations can also copy themselves. Copying a simulation member means creating data structures and functions capable of generating matrices, as well as vectors capable of storing solutions. Steady state problems usually do not use copies, a single instance of a matrix vector pair is normally sufficient to solve a steady state simulation. Transient simulations which need to have multiple spatial references will often create several copies. A transient simulation which uses a multi-step integration technique will need a copy of the variable space for each stage in its integration. Recall the discussion of multi-step methods from Chapter 2 where a history of points was maintained. The history is created in the MOOSE data structures by constructing multiple copies of the simulation solution matrix and vector. Each copy of the solution vector is assigned an index which relates it to an instance in time.

These concepts are perhaps best illustrated with a simulation example. Suppose that an engineer is interested in modeling a nuclear reactor core, but is unsure of exactly how to calibrate the model so it is precise enough to capture the various features of interest. In order to set up a transient model the simulation engineer must first study the steady state case. The transient model requires a different set of PDEs and extra variables to model the same scenario as the steady state eigenvalue

problem. The transient model will also require several copies of the vector space to represent the evolution of the model over time. This simulation design was used in the case study presented in Chapter 5. Example equations are presented in Appendix 3.

To accomplish this the engineer constructs a simulation group by specifying the layout of the cells for the reactor model. Some cells represent the core, others represent the moderator. The physics and constants which represent these elements can be specified from first principles in the current implementation of the MOOSE. Some special cells represent moving components. A protocol to initialize the simulation as well as a definition of the run time placement of components must also be established. These global simulation elements are represented by the box at the top of Figure 3.7.

The individual group members are constructed by creating different formulations of the physical model. One simulation group member is used to model the steady state problem, the other two simulation group members will be used to examine variations on the transient problem. A trapezoidal transient integration scheme although stable, is not necessarily the most efficient. A multi-value transient integration scheme which maintains a history of multiple derivatives can potentially take much larger steps than a trapezoidal scheme and hence will require less execution time although has more demanding memory requirements. These various formulations will determine the size of the data structures, they are pictured in Figure 3.7, labeled as A, B and C.

Simulation A implements a five group steady state nuclear diffusion problem. 5 variables per cell are required, one to represent each of the 5 energy states of the flux. This model is developed in detail in the first part of Chapter 5 which examines the steady state simulations of the MNR. Simulation B requires 6 precursor groups to be represented in addition to the 5 energy groups for a total of 11 variables per cell. In addition to the 11 variables required by sibling B, simulation C also requires an

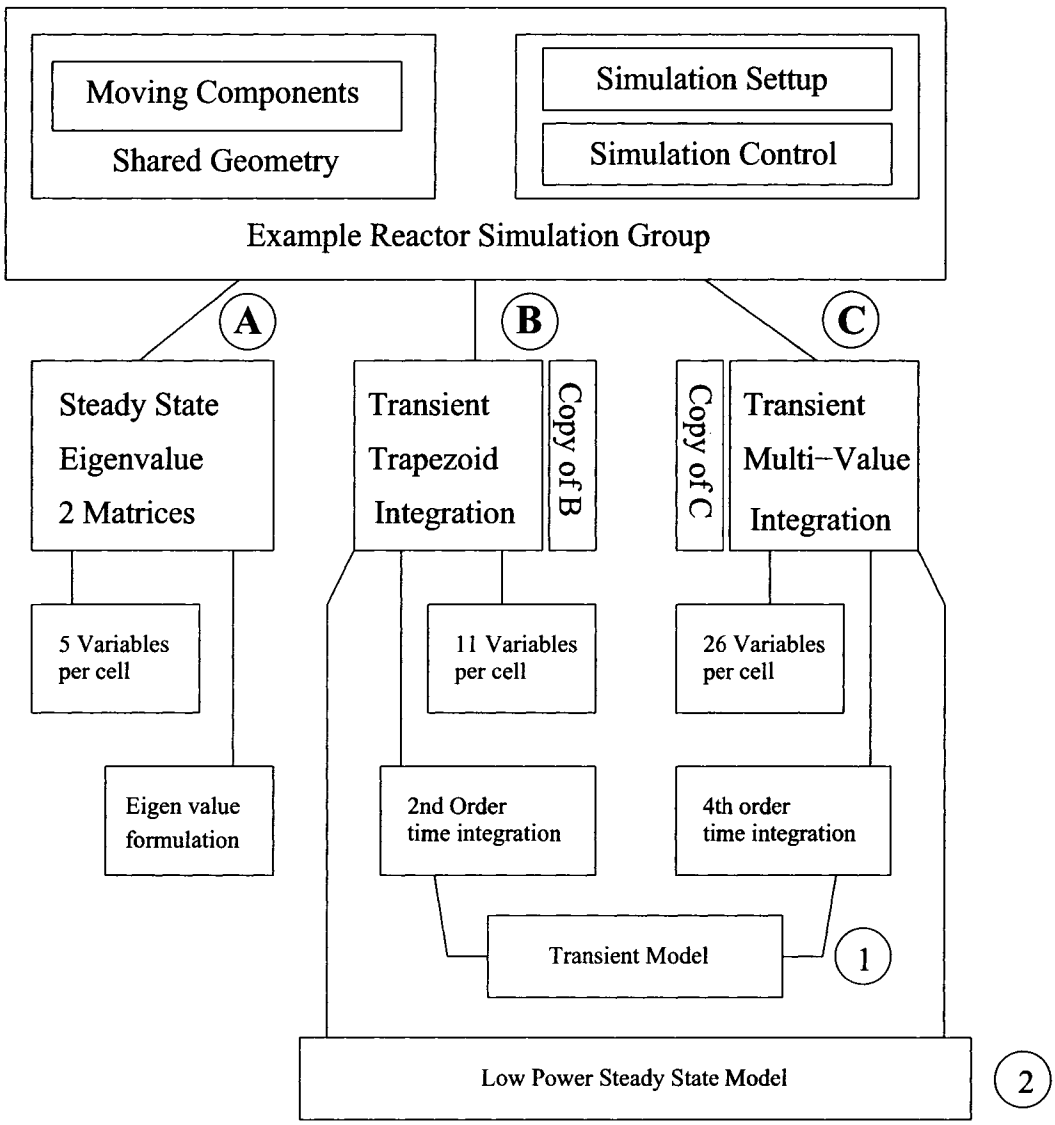


Figure 3.7: An Example MOOSE Simulation

additional 15 variables to keep track of the second, third and fourth derivatives with respect to time of each flux energy group for a total of 26 variables. Simulation C is discussed in detail in the second part of Chapter 5.

Simulation A only shares geometry and control traits with its other group members. Although simulation B and C use different transient order integration schemes (labeled 1) they use a common initialization method (labeled 2). Certain elements of the definition of their partial differential equations are identical as expressed by these equations. While individual simulation definitions which determine data structure size must be selected at compile time, the selection of individual PDEs may be done at run time. Simulation B and C must initialize their history prior to execution. This is done by solving a set of equations which assume that all rates of change of flux levels are zero. In the context of the particular example a low power steady state solution is needed to prime the history for both the neutron flux and the delayed precursors.

Transient simulations, like B and C, typically require the creation of a copy of the system. Recall that a copy means a duplicate of the entire vector representing the variables for the simulation, as well as a separate instance of the matrix used to solve that vector. Transient formulations can be written in a very general way as

$$\phi^{n+1} = \phi^n + \frac{d\phi}{dt} \triangle t \tag{3.1}$$

Rather than maintaining all solution vectors *1* through *n,* which would represent each time step taken by the model, a pair of problems is solved in a cyclic sequence

$$\phi^1 = \phi^2 + \frac{d\phi}{dt} \triangle t \tag{3.2}$$

$$\phi^2 = \phi^1 + \frac{d\phi}{dt} \triangle t \tag{3.3}$$

where the superscript refers to the copy either, *1* or *2*. The simulation engineer is responsible for setting up the time marching algorithm and specifying the details of the integration method through the PDEs. Typically equation 3.2 is solved followed by adjusting the time step or component positions after which equation 3.3 is solved. This process is repeated for every step in the simulation. Many problems can be solved with just one extra copy although in some scenarios, like for example a multi-step simulation, several may be required depending on the specific details of the problem.

Having defined the above simulation group, the simulation engineer now has three tools to investigate a moving mesh problem. The first tool, a steady state model, can be used to study still snapshots of the system. The two transient models provide alternate integration techniques. If memory is an issue it may not be possible to use the multi-value methods, if however stability or precision are issues then the simpler trapezoid model may be inappropriate. Being able to easily switch between subtle model variations is one of the fundamental strengths provided by a general framework. Additional examples which show how the MOOSE commands function is presented in Chapter 4.

3.5 Mathematical Principles Behind the MOOSE

Most of the mathematical principles implemented as part of the MOOSE are standard methods which were discussed in Chapter 2 and should be accessible to practitioners across a variety of fields. Since the MOOSE has taken the approach advocated by researchers interested in problem solving environments, the MOOSE packages together many different techniques. The MOOSE promotes the position that engineering tools should be built for reuse rather than single use. Re-usability and configurability when

achieved without sacrificing other desirable traits like efficiency or portability give a tool-set a marked advantage over other methodologies.

Section 3.5.1 will describe in some detail how the MOOSE generates matrices compatible with high-performance solution libraries from user defined input files. The remaining sections in this chapter present a collection of novel techniques mesh linking techniques implemented within the MOOSE which facilitate a high degree of accuracy in the results generated by the MOOSE.

3.5.1 Translating PDEs into Matrix Generating Functions

Part of the goal of the project is to take advantage of the highest performance solvers available, it is necessary for the MOOSE to generate a representation of the physical system of interest in terms that the solvers can process. The MOOSE must be able to take any physical equation and translate it into a sparse matrix which can be easily processed by an existing package.

The MOOSE targets problems which exhibit motion; thus the MOOSE may be required to generate a new matrix for every step in the calculation. Matrices with millions of elements cannot be efficiently generated by a scripting language like MATLAB, but a C or C++ program which links directly to a sparse solver package can generate thousands or potentially millions of matrices as part of a transient calculation.

One of the difficulties in working with the neutron diffusion equation is that most of the physics is embedded within the multi-dimensional constants. Exactly what the constants represent, how they are computed, and how many of them are used is something which is best left to the physicist.

The multi-dimensional nature of the neutron diffusion equation creates other com-

plexities as well. While for some physical systems it is possible to write a single matrix generator and allow its parametrization through adjustable physical constants, this is not the case for the neutron diffusion equation. To develop a program which generates a sparse matrix version of the neutron diffusion equation, flexible data structures and free form equation representation is required. While 2, 4, 8 and 12 energy groups are typical division in neutron diffusion studies any number of groups are possible, and the equation generator should be adaptable enough that it imposes as few restrictions as is possible.

Take for example a steady state version of the multigroup neutron diffusion equation for which the modeler has chosen to ignore upscatter, which was discussed in detail in Chapter 2, written as

$$- \bigtriangledown \cdot D_g \bigtriangledown \phi_g + \Sigma_{Rg}\phi_g - \sum_{g'=1}^{g-1} \Sigma_{Sg'g}\phi_{g'} = K \cdot \chi_g \sum_{g'=1}^{G} \upsilon\Sigma_{fg}\phi_{g'} \tag{3.4}$$

The symbols D, χ, Σ_R, Σ_s and $\upsilon\Sigma_f$ are spatially determined constants. K is the eigenvalue to be solved for. The main parameter of this equation is the energy group division number G which determines how many equations are represented and the exact structure that those equations will take.

If a system is constructed with $G = 2$ a pair of equations results

$$- \bigtriangledown \cdot D_1 \bigtriangledown \phi_1 + \Sigma_{R1}\phi_1 = K \cdot \chi_1 \sum_{g'=1}^{G} \upsilon\Sigma_{f1}\phi_{g'} \tag{3.5}$$

$$- \bigtriangledown \cdot D_2 \bigtriangledown \phi_2 + \Sigma_{R2}\phi_2 - \Sigma_{S12}\phi_1 = 0 \tag{3.6}$$

When $G = 2$, typically, $\chi_2 = 0$, so this term was left out of the second equation. The structure of each equation is dependent on G, and simultaneous solutions to

both equations are required for the solution of one spatial point. There are various interpretations for limits in the scattering term. When it is written as $\sum_{g'=1}^{g-1} \Sigma_{Sg'g}\phi_{g'}$ it represents down-scatter only. This term may also be more generally written with full up-scatter and down scatter as $\sum_{g'=1}^{G} \Sigma_{Sg'g}\phi_{g'}$, or more conservatively with down-scatter to one group only as $\Sigma_{S(g-1)g}\phi_{g-1}$. The syntax for the MOOSE has been developed to be generic enough to capture all of these representations, although none of them are hard coded into the MOOSE.

In MOOSE notation the neutron diffusion equation with down scattering only is expressed as:

```
seq(
   [- LAPL(D*Phi[g]) + Phi[g]*Sigma_r[g] - sum(Phi[g]*Sigma_s[j][g],j=1..g-1)
           =  K(Chi[g]*(sum(nu_Sigma_f[j]*Phi[j],j=1..G))), Phi[g]],

   g=1..G)
```

The interpretation of this version follows term for term the interpretation of the original, although it only represents down-scatter. The `LAPL()` operator is MOOSE specific, and provides a linear approximation to the $\nabla \cdot D_g \nabla$ operator in the original equation. The `seq()` and `sum()` operators represent a sequence of equations and summations respectively. The `K()` operator indicates to the MOOSE that two matrices need to be derived from this problem so a general eigenvalue problem of the form $Ax = kBx$ can be solved. The symbols `D`, `Chi`, `Sigma_r`, `Sigma_s` and `nu_Sigma_f` represent their Greek counterparts. The symbol `phi[g]` is specified on its own so that the MOOSE understands which terms in the PDE are variables which need to be solved.

Although MAPLE does provide code generating functions these only play a small role in translating the above PDE into a matrix generation program. They are used in the late stages of code generation for the elimination of common expressions, which

tend to turn up in some of the constants. MAPLE as a programming language offers a broad variety of expression search and substitution functions (which are relied on heavily by the MAPLE code generation program). MAPLE is also very good at rearranging the simple polynomials which tend to result in various terms which will cancel each other out. For example, the difference `Sigma_s[g][g] - Sigma_s[g][g]`, can be easily removed by the matrix generator before any code is actually output.

To build a matrix from the PDE, the matrix generator program must identify which variable is to represent the matrix diagonal. For the A matrix this must be non-zero, although for the B matrix the same requirement does not hold. The MOOSE generates an approximation to any finite difference operators, and estimates new values for constants as they are modified by those operators. For each variable in the PDE the constant and variable terms are first separated, and then those algebraic terms are translated into their corresponding C code.

So for example, using equation 3.6, the MOOSE first symbolically isolates the diagonal multiplier for one spatial dimension given that ϕ_2 is to represent the diagonal element. An approximation of $\bigtriangledown \cdot D_2 \bigtriangledown \phi_2$ in finite differenced form is generated. This term represents the flow into and out of a unit cell in the reactor core. A typical finite differenced approximation is derived from the first order Taylor series approximation to the first derivative which is applied twice. In one dimension the difference of the forward derivative and the reverse derivative is

$$\frac{d\phi}{dx} \mid_{x_i+\frac{\Delta}{2}} \cong \frac{\phi_{i+1} - \phi_i}{\Delta} \tag{3.7}$$

$$\frac{d\phi}{dx} \mid_{x_i-\frac{\Delta}{2}} \cong \frac{\phi_{i-1} - \phi_i}{\Delta} \tag{3.8}$$

where x_i represents the position in space, Δ represents the distance between points.

A centered average is used for D

$$D(x_i + \frac{\Delta}{2}) = \frac{1}{2}(D_{i+1} + D_i) \tag{3.9}$$

$$D(x_i + \frac{\Delta}{2}) = \frac{1}{2}(D_{i-1} + D_i) \tag{3.10}$$

Combining these and taking the second difference

$$\nabla \cdot D_2 \nabla \phi_2 = \frac{1}{2\Delta}(D_{i+1} + D_i)\left(\frac{\phi_{i+1} - \phi_i}{\Delta}\right) - \frac{1}{2\Delta}(D_{i-1} + D_i)\left(\frac{\phi_{i-1} - \phi_i}{\Delta}\right) \tag{3.11}$$

When the multiplier for ϕ_i is solved for

$$\frac{1}{2\Delta^2}D_{i-1} - \frac{1}{2\Delta^2}D_{i+1} \tag{3.12}$$

Including the remaining terms the final prefix derived is

$$-\frac{1}{2\Delta^2}D_{i-1} + \frac{1}{2\Delta^2}D_{i+1} + \Sigma_R \tag{3.13}$$

The code generator is able to identify $\frac{1}{2\Delta^2}$as a repeated sub expression, so prior to code output the preceding approximation is rewritten as a pair of expressions

$$t1 = \frac{1}{2\Delta^2}$$

$$VALUE = -t1 \cdot D_{i-1} + t1 \cdot D_{i+1} + \Sigma_R \tag{3.14}$$

As a last step this polynomial must be expanded into code which can be compiled

by gcc using references to data structures used by the MOOSE. The final code fragment for setting the diagonal matrix value of equation 3.6 for the two group model will look like this:

```
ROW_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
matrixdr_OPEN(Mid, ROW_POS);
t[1] = 1. / (2 * dx * dx);
VALUE = t[1] * (grid[x][y]->c->D[2] + grid[x+1][y]->c->D[2]) +
t[1]*(grid[x][y]->c->D[2] + grid[x-1][y]->c->D[2])
grid[x][y]->c->Sigma_R[1];
matrixdr_ADD(ROW_POS, VALUE);
```

The variable `X_st` stores the starting position in the vector for this map. Since multiple linked maps are possible, `X_st` is only zero for the first one. `P_SIZE` stores the property structure size. In the preceding example since `G=2`, and no other properties were defined for the cell `P_SIZE` was also two. There is no limit on `P_SIZE`. Typical values range anywhere from one to twenty depending on how the problem is specified. The array `grid[x][y]` stores geometry information about the problem. This data structure is automatically generated by the MOOSE from geometry input files supplied by the user. The C code generated by the MOOSE attempts to retain as many of the symbols as possible from the original PDE, so that constant names like `Sigma_r` or `D` which appear in the PDE will also appear in the generated code. This kind of bookkeeping was invaluable in debugging the MOOSE.

Notice that the above code fragment only sets the diagonal value. Additional code fragments are required to complete the row. Three more polynomials need to be computed as row matrix entries for this example. In addition to these requirements, another row must be built for equation 3.5, which requires three more polynomials. Matrix rows must be built for border sharing either using interpolation or conserva-

tion. The decision process involved in constructing a conservation based border is discussed in section 3.5.4. A partial listing for the code generated by the MOOSE for a two dimensional two group transient problem is given in Appendix 4.

The sparse matrix generator built by the MOOSE is built from collections of simple polynomials as in the preceding example. While the diagonal coefficient for the two group one dimensional case generates a fairly simple polynomial, this is not always the case. Some polynomials have literally dozens of terms in them, even after the identification and elimination of 5 or 10 repeated sub-expressions. This is especially true in the transient calculation for constant entries in the b vector.

Code generated by the MOOSE can be quite verbose. It is not unusual for the output from the MOOSE sparse matrix generator to be in excess of 20,000 lines. As a rough measure of complexity, a human programmer normally codes 1,000 lines in one month. The MOOSE therefore is doing the work of a engineer programmer translating an equation into a computer program at a remarkably fast rate.

It may be argued that since the code generated by the MOOSE is generated automatically that this code lacks certain optimizations that a good human programmer would use. While this may be the case, it is also the case that the automatically generated code demonstrates performance that is quite acceptable. During development, the MOOSE libraries and the automatically generated code were analyzed for efficiency using several profiling tools, such as gprof and others. These tests showed execution time was always dominated by the numerical solution libraries. For the fastest linear solver, initial sparse matrix setup and MOOSE map class construction consumed no more than 20% of the total execution time. For many of the eigenvalue problems matrix setup time takes less than 1% of the total execution time.

The time saved by using an automatic equation translator for the wide variety of possible PDEs of interest far outweighs any efficiency that might be gained by hand

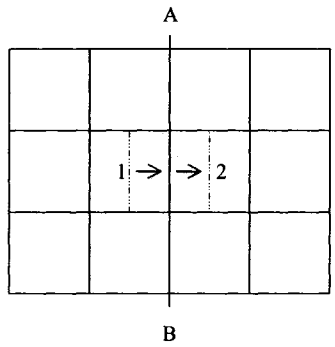


Figure 3.8: Conservation of Flow

coding the setup procedure for individual problems. The initial effort in building the MOOSE libraries and equation translators is not regained immediately, but after several hundred problem instances are constructed. This is the argument for building a carefully designed generic tool intended to be reused, opposed to tools which are only appropriate for a single use.

3.5.2 The Problem of Conservation

Conservation, in the context of this thesis, refers to whether the simple property of conservation of mass or energy at a cell or grid boundary holds or not. Normally when material passes into a boundary the same material must pass out of the boundary. Conservation, as a mathematical property of numerical balance, always holds for a finite difference formulation on a regular Cartesian grid. Consider the illustration in Figure 3.8. The equation, or set of equations which represents the value of the function for cell 1 will estimate a flow of material out of cell 1 and into cell 2. The expression which cell 2 uses to estimate the flow into it from cell 1 must exactly balance it. If this condition does not hold, then particles are being artificially created or destroyed and the problem ceases to be meaningful.

Referring to the previously discussed example of the neutron diffusion equation in section 3.5.1, on a Cartesian mesh the flux across the boundary AB will be computed using two expressions

$$\frac{d\phi}{dx} \mid_{outAB1} \cong \frac{\phi_1 - \phi_2}{\Delta} \tag{3.15}$$

$$\frac{d\phi}{dx} \mid_{inAB2} \cong -\frac{\phi_2 - \phi_1}{\Delta} \tag{3.16}$$

These expressions are a natural result of the symmetry of the problem, with a little attention to the direction of the signs, they work out to be equivalent. When working with more complex interfaces along AB the symmetry of the arrangement of the problem is often disrupted, and in these situations conservation can no longer be assumed to hold and special techniques must be used to minimize errors.

The previous work on linked meshes which addressed the general question of conservation of flow focused on fluid flow problems. Neutron diffusion is arguably more sensitive to errors induced by the artificial production of extra neutrons. For the scenarios discussed in this thesis moving mesh boundaries must often coexist with the fissioning fuel. Neutron fission is essentially a process of multiplication, any error in the neutron population which is calculated within the fuel is therefore multiplied by the modeled reaction rate, especially in the case of modeling an uncontrolled excursion. Consequently the details of neutron conservation within a simulated nuclear reactor core are quite different than those of other model types.

There are 3 basic scenarios that are considered in the following sections where it is not obvious how to maintain conservation. Solution methods which reduce errors are proposed which are demonstrated through numerical experiments in the next chapters. The three situations are

1. equally sized grid elements which are misaligned, solved with nonlinear interpolation techniques
2. unequally sized grid elements, solved by a geometric conservation rule
3. material discontinuities along boundaries, solved by a material discontinuity rule

3.5.3 Nonlinear Interpolation within Linear Models

As discussed in Chapter 2, in the section on composite grid methods, many authors argue that nonlinear interpolation methods are sufficient to link meshes, in particular see [39, 155]. Several papers present mathematical proofs on the subject which quantify the degree of error introduced by linking meshes with various orders of interpolation methods for certain cases related to fluid dynamics.

Given the success described by these authors it made sense to work with nonlinear interpolation methods as a mesh linking principle. During the course of developing the MOOSE and experimenting with various moving meshes it was learned that this principle holds under certain specific circumstances. If a pair of meshes is linked, but misaligned, and the mesh sizes on either side of a boundary are equal, and there are no special material discontinuities, then it was found that the meshes could effectively communicate using a nonlinear interpolation method for either side of the interface. For example, in Figure 3.9 where two equally sized meshes meet at a boundary line AB, despite the fact that they are misaligned, the material leaving cell 1 can be correctly estimated and balanced with the weighted partial sums of material entering cell 2 and cell 3. The justification is that this situation preserves symmetry.

In the case of the neutron diffusion equation neutron current is computed by estimating the gradient of the neutron flux on either side of the boundary AB. The neutron flux at the center of cell 1 is known, the gradient is computed by estimating

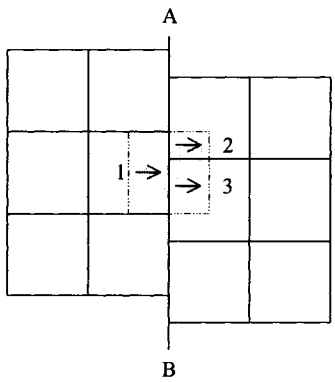


Figure 3.9: Conservation at Non-Aligned Boundary

a value for the flux which lies somewhere between the centers of cell 2 and cell 3 and using this to compute the gradient. As will be illustrated in tests at the end of chapter 4 and throughout chapter 5 this technique alone works reasonably well so long as there are no material discontinuities, and so long as the cells on either side of the boundary are of the same dimensions.

Several nonlinear interpolation methods were experimented with during the course of the development of the MOOSE mesh linkage including Lagrange and Newton interpolation, although spline based interpolation methods were found to yield the best results. A spline is a piecewise polynomial of degree k that is continuously differentiable k-1 times. A cubic spline is a piecewise cubic polynomial that is twice continuously differentiable.

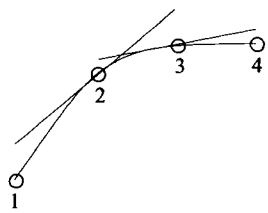
Despite the suggestion of nonlinearity in the name of the interpolation, using nonlinear interpolation methods is consistent with an overall linear solution method since the nonlinear terms are resolved before the system is solved. Since the MOOSE assumes that each submap uses a uniform mesh spacing certain simplifications in multi-point interpolation methods can be realized.

The situation illustrated in Figure 3.9 is the simplest case where the interpolation

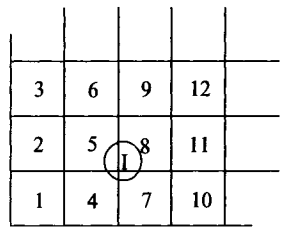
scheme can exclusively use points along the vertical axis. If the linked meshes do not use the same cell sizes the interpolation routines may be required to use more points. Interpolation routines are used both to connect meshes spatially but also to connect mesh points from differing time frames. Due to the motion of components, and the allowable arbitrary alignment of meshes a variety of interpolation situations must be handled. For example, requests for interpolated data made near the corners of meshes, or exactly in line with a row or column of data points will limit the number of points used by the algorithm. Conversely, interpolation requests made in the middle of a grid may use as many as 16 data samples to estimate the value at an arbitrary location. The MOOSE interpolation routines support the following scenarios

- One dimensional interpolation is required (a point along a line along the X only or Y only axes)
- Two dimensional interpolation is required (a point bounded by four or more points)
- Three points are available in either the X or Y directions
- Four points are available in either the X or Y directions

In the illustrated case in Figure 3.9 only 3 points are available, this case occurs near grid corners. Further away from corners it was found that a piecewise polynomial which used 4 points to define the shape of each section was beneficial. Figure 3.10 a) illustrates how the 4 point spline works. To estimate a value between points 2 and 3 a line is constructed which passes through those points. The slope of the line at points 2 and 3 is estimated using a finite difference. In figure b) this procedure is generalized for two dimensions near a corner of a grid. A value for the circled point (I) is to be estimated from the neighboring cells numbered 1-12. 4 points can be used



a) 1D, 4 point spline



b) 2D surface, 12 points used to interpolate

Figure 3.10: Interpolation Schemes

to estimate the shape of the surface in the X direction, but only 3 points are available to estimate it in the Y direction due to proximity with the edge.

In the simplest case where 3 points are available along a line a parabola is constructed using the equation

$$y = ax^2 + bx + c \tag{3.17}$$

and 3 points, where $x_1 - x_0 = x_2 - x_1$

$$(x_0, y_0), (x_1, y_1), (x_2, y_2)$$

A simple finite difference approximations for the first and second derivative is based on the given points. Equating the approximation to the exact first and second derivatives for the line it is possible to solve for a, b, and c in terms of y_0, y_1, and y_2.

$$a = \frac{y_2}{2} - y_1 + \frac{y_0}{2} \tag{3.18}$$

$$b = -\frac{y_2}{2} + 2y_1 - \frac{3y_0}{2} \tag{3.19}$$

$$c = y_0 \tag{3.20}$$

During the execution of the model some additional steps can be avoided by rearranging the solution so that for a specific x location a weighted sum of y_0, y_1, and y_2 is used. Given

$$w_0 = 1 - \frac{3x}{2} + x^2 \tag{3.21}$$

$$w_1 = 2x - x^2 \tag{3.22}$$

$$w_3 = -\frac{x}{2} + \frac{x^2}{2} \tag{3.23}$$

a new approximation can be written as:

$$y = w_0 y_0 + w_1 y_1 + w_2 y_2 \tag{3.24}$$

Although an x^2 term appears in these formulas the value of x^2 is determined prior to the solution of the matrix. The MOOSE always assumes that the position of all components are known in a given time frame so the weight vector can be computed at run time just prior to the solution of the matrix.

3.5.4 Boundary Sharing Conservation Rules

The MOOSE grid system connects meshes by using a ring of phantom cells around each mesh. Phantom cells reproduce values from other meshes and provide a convenient methodology for linking meshes. The example in Figure 3.11 shows an exploded

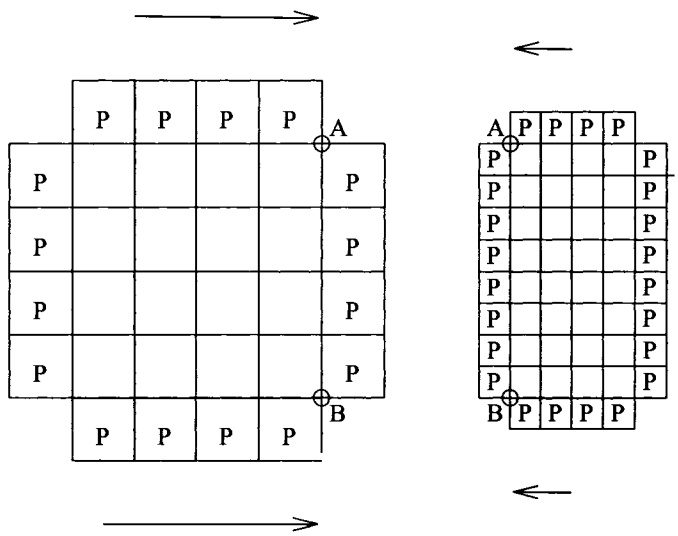


Figure 3.11: Phantom Cells, Exploded View

view of two meshes of different resolutions. The phantom cells are marked with the letter 'P'. When the two meshes are brought together the points marked A and B are coincident and the phantom cells of each mesh are tucked under the actual cells of the other mesh. This section explains how the MOOSE algorithms compute the phantom values.

There are two basic methods for computing the phantom values. The first method, already discussed in the previous section on interpolation, is to compute phantom values using a nonlinear interpolation method. This method works well when the two meshes use cells of the same size (or at least very close to the same cell size). However, as test results in the next section will show, when the interface is more complicated nonlinear interpolation is not sufficient to estimate phantom values for the calculation of flow.

Consider the interface illustrated in Figure 3.12. The equations for linking these two regions are no longer as simple as those described at the beginning of the section.

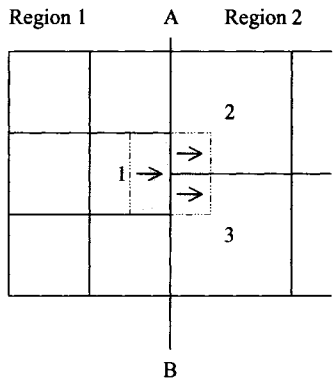


Figure 3.12: Elements of Flow

If region 1 computes values for its phantom cells using interpolation, and then cell 1 uses these phantom cells to estimate the flow of particles across the interface then cell 1 uses ***an interpolation rule to compute flow***. If instead region 2 estimates values for phantom cells using an interpolation method and then cell 1 estimates its flow by peeking at the flows estimated by region 2, and approximates the flow out of cell 1 by a weighted average of the flows into cells 2 and 3, then cell 1 uses ***a conservation rule to compute flow***. Using a conservation rule to compute flow is less direct, and more complex. Essentially what conservation rules do is allow an adjacent mesh to estimate flows using interpolation, and then assemble in a piecewise fashion, an estimate of flow which will exactly match.

3.5.5 Geometric Conservation Rules

The following analysis will follow a cell-centered, or finite volume approach that treats simulation variables as though they are defined within rectangular regions. The sample problem under consideration will be assumed to be the neutron diffusion problem, although the same arguments will hold for a variety of other diffusion style problems. Figure 3.13b shows a detailed view of the interface between region 1 and region 2

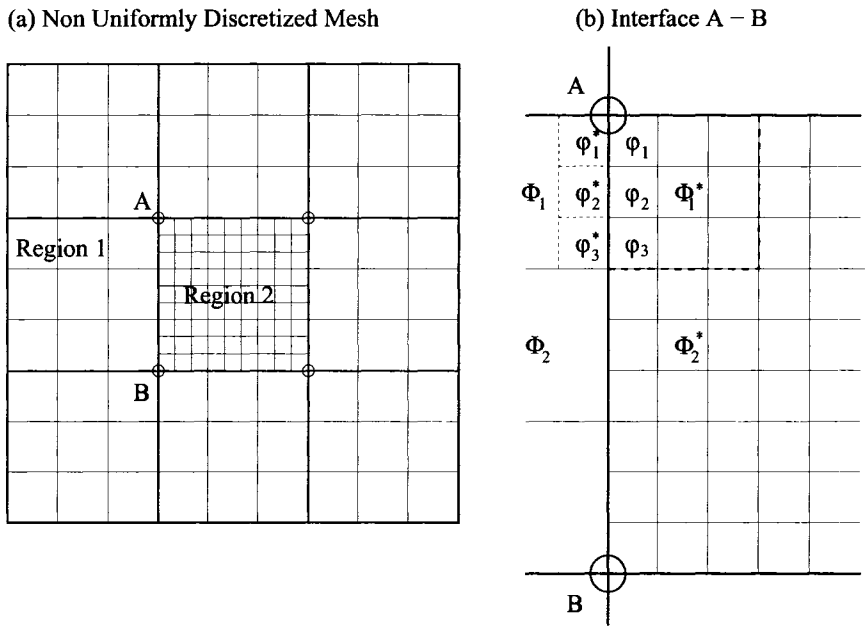


Figure 3.13: Interface 3:1

which is marked out as the line AB. A single flux point for region 1 is marked as Φ_1, and the flux for region 2 for the first 3 cells along the boundary are marked as φ_1, φ_2, φ_3. To compute neutron currents, or effectively represent the $D \bigtriangledown \phi$ term of equation 3.4, a boundary condition for region 1 which allows it to communicate with region 2 is needed. Within region 2 a cell of equal dimensions to the cell in region 1 is shown as a dashed line. The neutron flux for this phantom cell is identified as Φ_1^*. The second set of phantom values $\varphi_1^*, \varphi_2^*, \varphi_3^*$ represent the neutron flux in region 1, but in grid elements of dimension similar to region 2, these are the reverse analogue of Φ_1^*.

A first order difference $\frac{\Phi_1 - \Phi_1^*}{\Delta x}$ will be used to approximate the neutron current $\frac{\partial \Phi}{\partial x}$. In a case like the one expressed in Figure 3.13, the divisions on either side of line AB are an integer multiple of one another. Conservation of flux for this specific case can be written as

$$\frac{D(\Phi_1^* - \Phi_1) \triangle y_{R1}}{\triangle x_{R1}} = \frac{D(\varphi_1 - \varphi_1^*) \triangle y_{R2}}{\triangle x_{R2}} + \frac{D(\varphi_2 - \varphi_2^*) \triangle y_{R2}}{\triangle x_{R2}} + \frac{D(\varphi_3 - \varphi_3^*) \triangle y_{R2}}{\triangle x_{R2}} \quad (3.25)$$

where D is the diffusion constant previously discussed, $\triangle x_{R1}$ and $\triangle x_{R2}$ are the cell widths in region 1 and region 2 respectively, $\triangle y_{R1}$ and $\triangle y_{R2}$ similarly represent the cell heights in region 1 and region 2. Suppose that the phantom values $\varphi_1^*, \varphi_2^*, \varphi_3^*$ are computed using a nonlinear interpolation method, it would then be possible to compute Φ_1^* using equation 3.25, and then conservation could be guaranteed. This is an example of computing flow for region 2 using an interpolation rule, and computing flow for region 1 using a conservation rule. Alternatively if Φ_1^* is computed using a nonlinear interpolation rule then $\varphi_1^*, \varphi_2^*, \varphi_3^*$ could be computed using the following expression

$$\frac{D(\varphi_1 - \varphi_1^*) \triangle y_{R2}}{\triangle x_{R2}} = \frac{D(\varphi_2 - \varphi_2^*) \triangle y_{R2}}{\triangle x_{R2}} = \frac{D(\varphi_3 - \varphi_3^*) \triangle y_{R2}}{\triangle x_{R2}} = \frac{1}{3} \cdot \frac{D(\Phi_1^* - \Phi_1) \triangle y_{R1}}{\triangle x_{R1}} \quad (3.26)$$

Equation 3.26 is the reverse scenario which computes flow for region 2 using a conservation rule, and the flow for region 1 using an interpolation rule. For this particular case estimating $\varphi_1^*, \varphi_2^*, \varphi_3^*$ through interpolation and then computing Φ_1^* using equation 3.25 tends to produce better results than that estimating Φ_1^* through interpolation and solving for $\varphi_1^*, \varphi_2^*, \varphi_3^*$ with equation 3.26. The reason for this is that equation 3.26 discards information by assuming that the flow into each of the smaller cells is an equal fraction of the flow out of the larger cell. If the first formulation is used, equation 3.25, then the flow into the more detailed region maintains extra definition, while still being correctly balanced with the flow out of the larger cells.

A collection of tests (discussed in Chapter 4) which compared closed form diffusion

results with results computed on linked meshes showed that the side of an interface which is more detailed should estimate flow using an interpolation rule. The side of an interface which is less detailed should attempt to balance the flow using equation 3.25. When mesh sizes on either side of an interface are about the same, the most effective strategy was to use an interpolation rule for both.

The actual implementation of the MOOSE framework is not limited to dealing with integer multiples of cells. Built into the matrix generator are systems for expressing both equation 3.25 and equation 3.26 for arbitrary cell sizes and the appropriate rules for deciding when to construct matrix entries representing these different variations. In practice cell sizes should be approximately similar, cell size ratios of more than 3:1 tend to produce unsatisfactory simulation results. The decision about when to implement either rule can be made prior to solving the simulation problem itself. Code which decides on which rules to apply works on a cell by cell basis, and scans the perimeter of each mesh prior to constructing a matrix which represents the problem. The automatically generated code which constructs these rules tends to be quite long, an example is presented in Appendix 4.

3.5.6 Material Discontinuity Conservation Rules

Conservation can be determined by relative cell size alone if there are no material discontinuities along the borders. Most papers (see chapter 2, discussion on conservative meshes) that have investigated linking multiple meshes with either conservation rules or interpolation rules have simply advocated keeping mesh discontinuities far away from mesh boundaries. For the nuclear rod insertion problem this is not possible. In order to simulate the insertion of either a fuel assembly or control rod, material discontinuities must be present along the mesh boundary. This presents an additional

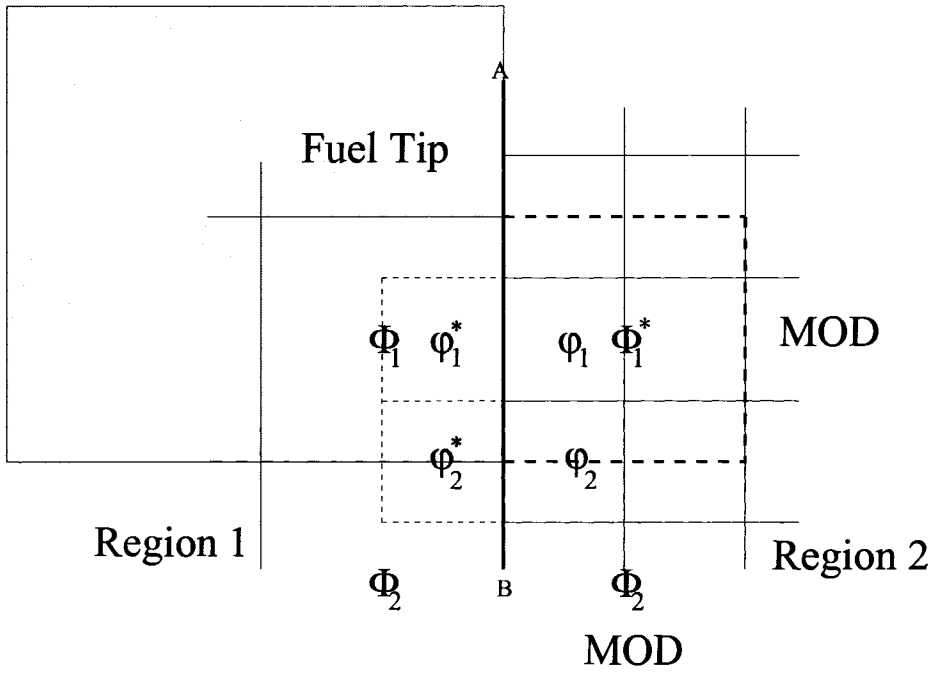


Figure 3.14: Material Discontinuities

complication, illustrated in Figure 3.14.

Figure 3.14 illustrates a situation where the top two thirds of region 1 represents some physical discontinuity of the simulation (in this case the leading tip of a fuel assembly), while the bottom third of region 1 represents another material, in this case the moderator for the fuel. Region 2 is entirely composed of the moderator.

In order to compute the neutron current going across the boundary an average constant which represents the material cross section in both regions is needed. Referring back to the section which discussed the individual elements of the Laplacian operator, the first term of equation 3.11 represents the flow across one edge of a cell, in this example the flow out of the cell labeled φ_2 and across the AB boundary is considered. This can be rewritten in terms of phantom cells as

$$\left(\frac{\varphi_2 - \varphi_2^*}{\triangle x_{R2}}\right) \Delta y_{R2} \cdot \left(\frac{D + D^*}{2 \triangle x_{R2}}\right) \tag{3.27}$$

The errors incurred by using mismatched mesh sizes are not nearly as large as the errors which result from the heuristic estimate of constants. In the above case D^* as estimated for the region inhabited by φ_2^* must be represented in an ad-hoc way as a volume weighted average of constants. This is problematic, since part of the point of using moving meshes is precisely to avoid the naive volume weighted approximation.

The previous section which discussed the geometric conservation rules indicated that region 1 should compute flow using a conservation based rule rather than an interpolation based rule due to the differences in cell sizes. Before deciding to compute flow for region 1 with a conservation rule a check is done to test that region 1 has no material discontinuities in the area of Φ_1. This check ensures that the cells to the north and to the south of Φ_1 are the same as Φ_1. In the above example this test fails and it indicates that φ_2^* should be computed using a conservation rule because of material discontinuities in region 1. For the following equation, the D term is subscripted with the variable that it is associated with

$$\frac{D_{\varphi_2}(\varphi_2 - \varphi_2^*) \triangle y_{R2}}{\triangle x_{R2}} = \frac{1}{4} \cdot \frac{D_{\Phi_1}(\Phi_1^* - \Phi_1) \triangle y_{R1}}{\triangle x_{R1}} + \frac{1}{4} \cdot \frac{D_{\Phi_2}(\Phi_2^* - \Phi_2) \triangle y_{R1}}{\triangle x_{R1}} \tag{3.28}$$

Equation 3.28 can be used to compute φ_2^* using a conservation rule. Strictly speaking the corrupt term D_{φ_2}, computed from weighted averages, is still used to compute the flow across the boundary AB. However, since this term is only relevant in relation to φ_2^* and since φ_2^* has been computed in a way which forces flow to be conserved, the inaccuracies of D_{φ_2} are corrected.

The material conservation rule works because it avoids using a heuristic approxi-

mation in the linking of meshes. This rule is normally applied along all edges prior to solving a system of equations. It is often the case that only a few instances of this rule come into play, for example near the leading or trailing edges of a moving component. The statement that this rule is more important than the geometric conservation rule is somewhat problem dependent. Experience gained during the case study presented in Chapter 5 suggested that the material conservation rule had a large impact on solution accuracies.

3.6 Summary

During the course of developing the mesh linking strategy several basic principles were identified. They were presented in this chapter in order of priority, lowest priority first. To summarize, the basic mesh linking rules are

1. Use an interpolation rule to compute flow on both sides of an interface only if the cell dimensions on either side are roughly the same, and only if rule 2 and 3 are not violated.

2. Use an interpolation rule to compute flow on the side of an interface which has smaller cells. Use a conservation rule to compute flow on the side of an interface which as larger cells, only if the last rule is not violated.

3. Use an conservation rule to avoid computing flows with heuristically computed constants.

The justification for each of these rules has to do with avoiding estimates of quantities, either by assuming certain quantities are equivalent which may not be, or by computing values in an ad-hoc fashion. The first rule only applies in the specific case

where cell sizes are equal but simply misaligned and no material discontinuities are present, this rule holds as a matter of symmetry since there is no clear reason to apply a conservation rule to either domain.

In the case of a conflict, where it appears that there are too many material discontinuities on either side of a mesh interface, interpolation is chosen as the default for both sides. Finally, if conservation rules are used on opposite sides of an interface which exactly oppose each other, a singular matrix and an unsolvable problem will result. It amounts to specifying a set of equations similar to $\varphi_1 - \varphi_2 - \varphi_3 = 0, \varphi_2 - \varphi_1 - \varphi_3 = 0$ which gives no information about $\varphi_1, \varphi_2, \varphi_3$ unless further equations are specified. This situation must be avoided by always forcing cells on one side or the other of the interface to use interpolation should a conflict arise. Normally the check for material purity in the current region will reduce the likelihood of this conflict, but it can still occur.

This chapter has summarized some of the essential details behind the MOOSE framework. As already noted the code for the framework is quite extensive, the presentation in this section should give an indication of the level of complexity of the MOOSE algorithms, and the details behind some of the components. In an effort to keep this chapter short many details regarding the implementation have been neglected, and the presentation of the principles behind the MOOSE have focused on simplified examples rather than on the fully general scenarios implemented within the framework.

Despite this, the mesh linking principles themselves are not that complicated and should be easy to appreciate from a first principles stand point. A simulation expert interested in implementing a linked mesh need not employ all the details of the MOOSE framework. The mesh linking rules can be implemented on their own for a specific mesh layout, and the same results should be achievable, either for the case of

moving meshes, or for the case of a stationary mesh with varied resolution.

Similarly a simulation expert interested in applying principles of computer algebra and code generation should find some of the details presented in this chapter insightful. For the MOOSE code generation techniques provided a bridging point between a pre-existing computer algebra language and high performance numerical solvers. Some authors who write about code generation describe it as a panacea. This kind of hype is typical of trends in computing, and while code generation has its place in program development, it should be undertaken only with good justification.

Chapter 4

Verification

This chapter will present a collection of problems and their solutions to verify the results generated by the MOOSE. Along side problem verification some concrete examples of the procedures involved in setting up a problem for the MOOSE will be given. Problems were chosen from a variety of areas including electrostatics, heat conduction, wave propagation, as well as the target area of reactor physics. All of the problems involve typical field and potential calculations that utilize the most important features of the MOOSE. The example problems are chosen to represent each of the fundamental problem types, either parabolic, elliptic or hyperbolic in nature. One and two dimensional problems are considered in both steady state and transient variations. Finite differences are used to approximate first and second derivatives.

Problem choice was made in favour of those examples that have closed form solutions. Good sources of problems include [34, 106, 109, 129, 167].

It should be noted that much research effort has been invested in the subject of software verification and validation and while this project acknowledges the importance of these two subject areas this thesis does not address either but focuses rather on model verification and validation. The use of library routines independently imple-

mented and tested by other researchers can greatly reduce verification and validation efforts. Software verification and validation properly falls within the domain of software engineering and deals with issues like proving the correctness of an algorithm and demonstrating that the algorithm generates expected results.

The method of manufactured solutions (MMS) [137] is a technique which has been used in recent years to verify problems for which closed form solutions are not available. The goal of MMS is to manufacture an exact solution to a slightly modified variation of the target problem for the purpose of verifying a simulation. The modified equations need not represent an actual physical scenario, but are rather based on the same equations as the physical model with additional source terms and special boundary conditions that permit comparison with the apriori determined solution. Manufactured solutions should be chosen to be smooth analytical functions with smooth derivatives. Care must be taken to ensure that no single term in the governing equation dominates any other term. Realizable solutions should be also be used. For example, if the problem includes water flow, the manufactured solution should not include temperatures for frozen or boiling water. Since MMS requires the ability to include arbitrary source terms, initial conditions and boundary conditions, it must be possible to include the specific form of the manufactured solution in the code. MMS is thus a code intrusive methodology and cannot be used for black box analysis. MMS is predicated on having smooth solutions, the analysis of non-smooth solutions (shock-waves, material interfaces, etc.) is an open research issue.

While MMS was seriously considered as a verification technique for this chapter, as a methodology it is still very young, and hence there are few introductory examples in the literature to illustrate its use. Instead, this chapter focuses on the use of exact solutions for verification which has a long history and convincingly demonstrates a degree of confidence for the MOOSE framework. Simulation errors may be difficult

to detect simply because there may be no other existing models and no physical data to compare with. The confidence developed in this chapter through simple examples will provide the foundation on which the results for the next chapter will be laid.

Surveys of verification techniques are presented by [137, 140].

4.1 Verification versus Validation

Verification and validation procedures provide a set of tools and methodologies for building confidence in computational simulations. In common usage the words verification and validation are synonymous; however, in current engineering usage they have very different meanings. There are no standardized meanings for verification and validation, this section presents definitions of these terms based on [98, 137, 140].

Verification asks questions related to the mathematics, computer science and software engineering as they apply to a simulation's implementation. Verification of a model can be addressed entirely though apriori techniques and makes no connection between the model and any observable phenomena. A model which is verified is a model which has been shown to be internally consistent, or which follows the rules of logic and mathematics in a rigorous, accepted and reproducible way. For some classes of models a variety of well understood properties have been proven to be invariant, for example the rate of change of error compared with reductions in mesh size. Demonstrating that a new model can reproduce these same invariant properties provides evidence that weighs in favour of that model being verifiable.

In contrast, validation deals with the physics and engineering principles of the model and addresses the ability of the model to reproduce experimental data. Physical models are themselves open to interpretation and subject to a variety of simplifying assumptions, which may or may not be appropriate. Validation has two main aspects:

conceptual validation, the faithfulness to which the implementation on a computer mirrors an accepted physical model, and results validation, the comparison of the simulation's output with an appropriate referent to demonstrate that the model or simulation can in fact support the intended use.

A model that is verifiable but cannot be validated is not very useful. This scenario can happen when the implementation is correct and consistent, but a fundamental physical assumption is incorrect. For example, assuming that some component in a model is weightless may result in a verifiable model, but might not generate convincing real world results. Similarly it might be possible to demonstrate that a particular model can be validated for certain cases even though the model fails verification tests. This can indicate errors in the solver or mesh implementation.

If a model is verified, and validated for several cases then a certain degree of confidence is established that in the future it will correctly predict results. Meeting both conditions still only increases the confidence that the model is correct, it never establishes 100% correctness. The remainder of this chapter will focus on issues central to verification, the next chapter implements a fairly detailed case study of the McMaster Nuclear Reactor and provides some evidence of model validation.

4.1.1 Issues Related to Verification

In any simulation there are several typical sources of errors

- Physical Modeling Errors
- Discretization Errors
- Numerical Errors
- Programming Errors

Physical modeling errors are those induced by the choice of equations to model the system. Most physical models make a variety of assumptions and simplifications, which focus the model on a phenomenon of interest, while ignoring terms which are not of interest. Typical modeling simplifications might include frictionless media, the use of lumped masses or continuous quantities to model large numbers of small particles, modeling a phenomenon like the advection of fluid while ignoring the vorticity of that fluid, studying one and two dimensional representations of three dimensional phenomena, the use of symmetrical models, and so on. Such simplifications make problems tractable. So long as the implications of simplifications are understood they can be of great assistance.

Discretization errors are those that are introduced when a physical model is converted into a computerized model. Some physical problems can be solved through classical calculus. Programs like MAPLE or Mathematica are able to integrate continuous functions algebraically. Despite the many recent advances in symbolic solution of physical problems, most simulation work is still done with numerical approximations to derivatives. Numerical approximations to derivatives are normally derived from an infinite Taylor series, for which only the most significant, or lowest order terms are retained. The terms which are neglected become part of the error. An example of how such a system is derived is given in the next section.

Errors can also be the result of round off, or the discritization of continuous quantities which a computer must undertake to represent floating point numbers in binary registers. When an algorithm must repeatedly multiply and add several million individual registers together the machine's inability to keep track of the least significant bit in a floating point number's representation will accumulate. Algorithms which are well designed can address these issues to a certain degree.

Programming errors are the bane of any large project and it is fair to suggest that any software system will have some. Systematic errors are perhaps the simplest to identify and correct, for example the software generates negative results instead of positive ones, or consistently generates predictions that are 10% too low. Programming errors that cause a simulation to fail under certain circumstances are not nearly so serious as programming errors that cause a software system to report incorrect results or worse even, correct results some of the time, and incorrect results at other times.

For many problems the time available to compute the solution will provide the ultimate limit. The modeler must often choose between acceptable error introduced by discretization, and the amount of time they are willing to wait for this solution. A fast model that yields a result with an uncertainty outside a practical range is just as useless as a precise model that will yield an exact result too late to be of any use.

4.1.2 Consistency and Convergence

For a numerical scheme to be consistent, the discretized equations must approach the original partial differential equations in the limit as the element size approaches zero. For a stable numerical scheme errors due to round-off, iterative truncation or other similar sources must not grow in the marching direction. This discussion of stability, consistency, and convergence is taken from [70, 140].

Convergence addresses the issue of whether the solution to the discretized equations approaches the continuum solution of the partial differential equation in the limit of decreasing element size. Convergence is addressed by Lax's equivalence theorem, which states that given a properly-posed initial value problem and a consistent numerical scheme, stability is the necessary and sufficient condition for convergence.

Consistency then is a property of the discretization of the equations while convergence deals with the solution method of those equations.

For verification purposes it is convenient to define the discretization error as the difference between the solution to the discretized equations and the solution to the original partial differential equations.

One approach to evaluate the truncation error for the example of a finite difference scheme is to start with a Taylor series expansion of the solution variables. For example (see [140]), consider the function $T(x)$ expanded about the point x_0, the Taylor series expansion can be written as

$$T(x) = \sum_{k=0}^{\infty} \left. \frac{\partial^k T}{\partial x^k} \right|_{x_0} \frac{(x - x_0)^k}{k!} \tag{4.1}$$

Consider the one-dimensional transient heat equation given by

$$\frac{\partial T}{\partial t} - \alpha \frac{\partial^2 T}{\partial x^2} = 0 \tag{4.2}$$

where α represents the constant of thermal conductivity. This equation can be discretized with finite differences using a forward difference in time and a centered second difference in space, resulting in the simple explicit numerical scheme

$$\frac{T_i^{n+1} - T_i^n}{\triangle t} - \alpha \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\triangle x)^2} = 0 \tag{4.3}$$

where the subscripts denote spatial location and the superscript denote the temporal step. To determine the truncation error for this numerical scheme, each of the above temperature values can be expanded in terms of the temperature at location i and time step n.

$$T_i^{n+1} = T_i^n + \left.\frac{\partial T}{\partial t}\right|_i^n \frac{\triangle t}{1!} + O\left(\triangle t^2\right)$$

$$T_{i+1}^n = T_i^n + \left.\frac{\partial T}{\partial x}\right|_i^n \frac{\triangle x}{1!} + \left.\frac{\partial^2 T}{\partial x^2}\right|_i^n \frac{(\triangle x)^2}{2!} + \left.\frac{\partial^3 T}{\partial x^3}\right|_i^n \frac{(\triangle x)^3}{3!} + O\left(\triangle x^4\right)$$

$$T_{i-1}^n = T_i^n + \left.\frac{\partial T}{\partial x}\right|_i^n \frac{(-\triangle x)}{1!} + \left.\frac{\partial^2 T}{\partial x^2}\right|_i^n \frac{(-\triangle x)^2}{2!} + \left.\frac{\partial^3 T}{\partial x^3}\right|_i^n \frac{(-\triangle x)^3}{3!} + O\left(\triangle x^4\right) \quad (4.4)$$

Substituting these expressions into the discretized equation and comparing the finite differenced derivatives with their exact counterparts allows us to evaluate their error.

$$\frac{\partial T}{\partial t} = \frac{T_i^{n+1} - T_i^n}{\triangle t} + O\left(\triangle t^2\right) \quad (4.5)$$

$$\alpha \frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\triangle x)^2} + O\left(\triangle x^4\right)$$

The difference between the original partial differential derivatives and the discretized equation is the truncation error. This simple explicit scheme for the transient heat equation is consistent since the truncation error goes to zero as $\triangle x$ and $\triangle t$ go to zero. The formal order of accuracy of the scheme is first order in time and second order in space since the error terms contain the factors $\triangle t$ and $(\triangle x)^2$.

4.1.3 Measuring Accuracy

The observed order of accuracy is the accuracy computed from code output for a given simulation or set of simulations. The observed order of accuracy can be adversely affected by mistakes in the computer code, solutions which are not sufficiently smooth, defective numerical algorithms and numerical solutions that are not in the asymptotic mesh convergence range. The asymptotic range is defined as the range of discretization sizes where the lowest-order terms in the truncation error dominate.

Supposing that an exact solution is known consider a series expansion of the discretization error in terms of h_k, a measure of the element size on mesh level k.

$$DE_k = f_k - f_{exact} = g_p h_k^p + \mathbf{HOT} \tag{4.6}$$

where f_k is the numerical solution on mesh k, g_p is the coefficient of the leading error term, and p is the observed order of accuracy. The main assumption is that the higher-order terms (**HOT**) are negligible, which is equivalent to saying the solutions are in the asymptotic range. In this case, the discretization error equation for a fine mesh and a coarse mesh is

$$DE_1 = f_1 - f_{exact} = g_p h_1^p \tag{4.7}$$

$$DE_2 = f_2 - f_{exact} = g_p h_2^p \tag{4.8}$$

Since the exact solution is known, the left-hand sides can be evaluated using the numerical solution. Combining these two equations as follows

$$\frac{DE_2}{DE_1} = \frac{g_p h_2^p}{g_p h_1^p} = \left(\frac{h_2}{h_1}\right)^p \tag{4.9}$$

the observed accuracy is then

$$p = \frac{ln\left(\frac{DE_2}{DE_1}\right)}{ln(\frac{h_2}{h_1})} \tag{4.10}$$

Given an exact solution only two solutions are required to obtain the observed order of accuracy. The observed order of accuracy is effected by round-off and iterative convergence errors. Discretized forms of nonlinear equations can generally be solved to within machine round-off error. Iterative procedures are often terminated early to reduce computational effort, alternatively direct sparse methods, as already discussed, can be used to avoid errors associated with iterative methods.

4.2 Electrostatics Problems

The first example presented here is an electrostatics problem. Since the MOOSE is a prototype framework it is conceivable that a well developed PSE might be fully menu driven and suggest appropriate formulas and constants to the user in a problem context sensitive manner. Commercial tools with advanced user interfaces like FEMLAB do precisely this. However, as mentioned previously, the focus of this thesis is not on user interface development, but rather on a study of the PSE's essential components and the methods required for the precise modeling of motion. Individual conceptual elements, as identified in Chapter 3 will be highlighted and their relevance to the current problem mentioned. This first example is described with some additional details to give a clear idea of what level of interaction with the MOOSE is required, subsequent examples will be less exhaustive in the way they detail the solution process.

The details of the closed form representation were taken from [109].

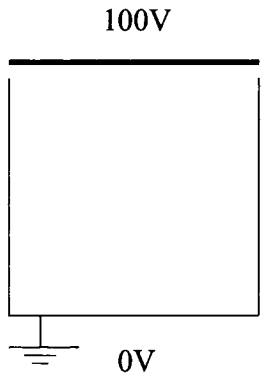


Figure 4.1: Electro-Statics Model

This model computes the electric field generated by a pair of wires bent into a square domain of dimension $L \times L$. From classical electrodynamics the electrical potential U(x) satisfies Poisson's PDE

$$\nabla^2 U(x) = -4\pi\rho(x) \tag{4.11}$$

Where $\rho(x)$ is the charge density at the spatial location. The above representation is the steady state case so there is no time dependence. In charge free regions where $\rho(x) = 0$ the scalar potential satisfies Laplace's equation:

$$\nabla^2 U(x) = 0 \tag{4.12}$$

In two dimensional rectangular coordinates it takes the form

$$\frac{\partial^2 U(x,y)}{\partial x^2} + \frac{\partial^2 U(x,y)}{\partial y^2} = 0 \tag{4.13}$$

4.2.1 Analytic Solution

To derive an analytical solution to Laplace's equation, using the method of separation of variables, first assume that the problem is the product of independent functions of X and Y

$$U(x,y) = X(x)Y(y) \tag{4.14}$$

Because $X(x)$ is a function of only x and $Y(y)$ of only y, the derivatives are ordinary instead of partial. Since $X(x)$ and $Y(y)$ are assumed to be independent, the only way this equation can be valid for all values of x and y is for each term to be equal to a constant

$$\frac{d^2X(x)/dx^2}{X(x)} = -\frac{d^2Y(y)/dy^2}{Y(y)} = k^2 \tag{4.15}$$

The choice of sign for the constant matches the boundary conditions and gives periodic behaviour in X, but not in Y. Solutions for X and Y are

$$X(x) = Asin(kx) + Bcos(kx) \tag{4.16}$$

$$Y(y) = Ce^{ky} + De^{-ky} \tag{4.17}$$

The $x = 0$ boundary condition can be met only if $B = 0$. The $x = L$ boundary conditions can be met only for values of k for which

$$k = L = n\pi \ , \ n = 1, 2, 3, ...$$

for each value of n there is a solution for X that is

$$X_n(x) = A_n sin\left(\frac{n\pi}{L}x\right) \tag{4.18}$$

For each value of k_n which satisfies the x boundary conditions, the y solution $Y(y)$ must satisfy the boundary conditions $U(x, y = 0) = 0$. This requires $D = -C$ and so

$$Y_n(y) = C\left(e^{k_n y} - e^{-k_n y}\right) \equiv 2C sinh\left(\frac{n\pi}{L}y\right) \tag{4.19}$$

In this case the principle of linear superposition holds and this means that the most general solution is the sum of the products $X_n(x)Y_y(y)$

$$U(x, y) = \sum_{n=1}^{\infty} E_n sin\left(\frac{n\pi}{L}x\right) sinh\left(\frac{n\pi}{L}y\right) \tag{4.20}$$

The E_n values are arbitrary constants and are fixed by requiring the solution to satisfy the remaining boundary condition at $y = L$. For this example the boundary condition is $U(x, L) = 100V$, so

$$\sum_{n=1}^{\infty} E_n sin\left(\frac{n\pi}{L}x\right) sinh\left(n\pi\right) = 100V \tag{4.21}$$

Due to symmentry even harmonics cancel. The potential for any point in the space is

$$U(x, y) = \sum_{n=1,3,5,\ldots}^{\infty} \frac{400}{n\pi} sin\left(\frac{n\pi x}{L}\right) \frac{sinh\left(n\pi y/L\right)}{sinh\left(n\pi\right)} \tag{4.22}$$

When evaluating the analytic term the $sinh()$ function may overflow for large values of n. Some of these overflows can be avoided by expressing the quotient of the two hyperbolic sine functions in terms of exponentials

$$\frac{sinh\,(n\pi y/L)}{sinh\,(n\pi)} = \frac{e^{n\pi(y/L-1)} - e^{-n\pi(y/L+1)}}{1 - e^{-2n\pi}} \tag{4.23}$$

4.2.2 Finite Difference Solution

To formulate this problem and its boundary conditions in the MOOSE is straightforward. The user follows the sequence of steps:

1. Create a data structure with 1 element U, to store the electrical potential.
2. Create 3 cell types
   (a) A charge free cell
   (b) A 100 Volt potential cell
   (c) A 0 Volt potential cell
3. Create equations for each cell type
   (a) Equations are equivalent except for the $-4\pi\rho(x)$ term
4. Draw the geometry for the problem
5. Create the solver program which initializes the problem, solves it, and plots it.

Creating the data structure to be solved for is very simple. The data structure editor is launched from the MOOSE's edit menu, a single entry "U" needs to be added to the new structure, and the structure needs to be saved with a simple name, like *estatics_pdef*.

Cells are created in a similar manner, the cell editor is launched from the MOOSE's edit menu, the 3 cells need to each have 2 fields defined, their data structure, and what equations to use.

The equation syntax for the MOOSE is quite simple for this problem. Through the user interface a text editor can be launched and the Laplacian equation can be written as:

```
PDEs:=[[LAPL(U)=0,U]];
```

The MOOSE adopts MAPLE syntax for this case, the variable *PDEs* is specified to the MOOSE framework as a list of pairs. Each pair in the list consists of a symbolic expression of one of the PDEs, in this case `LAPL(U)=0`, followed by the name of the variable to be solved for, in this case `U`. This expression can be though of as defining a row of the matrix, by specifying the variable the user is clarifying which symbol is to represent the diagonal in the matrix.

The 100 volt boundary conditions is expressed simply as:

```
PDEs:=[[U=100,U]];
```

This expression effectively solves a formula to generate a boundary condition. It similarly defines a list of pairs, the first entry in the pair is an equation, the second entry specifies the symbol associated with the matrix diagonal for the equation. The MOOSE framework does not make a special distinction between boundary conditions and PDEs. It is up to the model designer to ensure that a simulation domain is adequately specified.

The zero volt boundary condition is similar to the 100 volt boundary condition. Each equation should be associated with its respective cell type, this is managed through the MOOSE cell editor. Finally a solver program is needed. The basic solver program is summarized in pseudo code as Algorithm 1. All of the other details of matrix creation, and equation interpretation are handled by the MOOSE framework.

Notice in Algorithm 1 both the copy and the equation group must be specified. Recall from the discussion in the previous chapter that it is possible to specify multiple

Algorithm 1 Pseudo Code for Electrostatics Solver

```
// include MOOSE definitions

main()
{
     Initialize_Model(xdim=100,ydim=100, Copy=1, Equation_Group=1);
     solverdr_solve(Copy=1, Equation_Group=1);
     html_figure(title='E-Field',variable=U);
}
```

copies, and multiple model group members. For a steady state problem only a single copy is needed and only a single equation set is used. In the next transient example multiple copies and equation groups will be used.

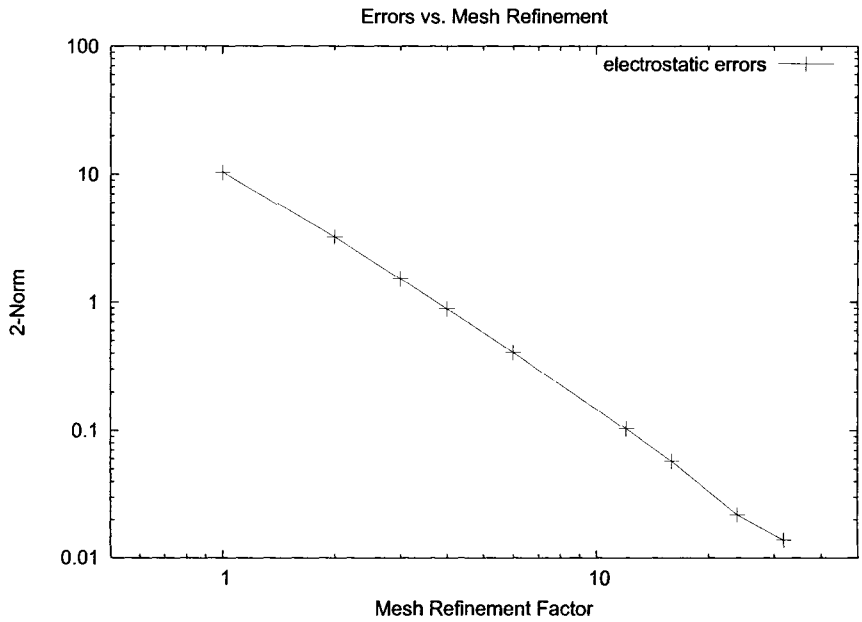


Figure 4.2: Errors Plotted Against Mesh Refinements

For the sake of comparison the 2-norm of the error is used, here defined as

$$\|e\|_2 = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} (exact_i - measured_i)^2} \tag{4.24}$$

By examining the error for various mesh resolutions it is possible to compute the observed order of the function as discussed in section 4.1.3. For this problem the point at which the 0V boundary condition meets with 100V boundary condition creates difficulties for the model. This illustrates the importance of using the 2-norm. If the infinity norm is used to compare errors, the error in the region of this localized discontinuity will dominate the problem. The 2-norm however provides a better measure of global error, and is thus a more representative way to compare the analytical and finite difference solutions. If the 2-norm is plotted against the mesh refinement factor then a straight line results as in Figure 4.2. As the mesh refinement increases the line better approaches the ideal, the observed accuracy is measured to be 2.01, which closely matches the theoretical expectation of 2.

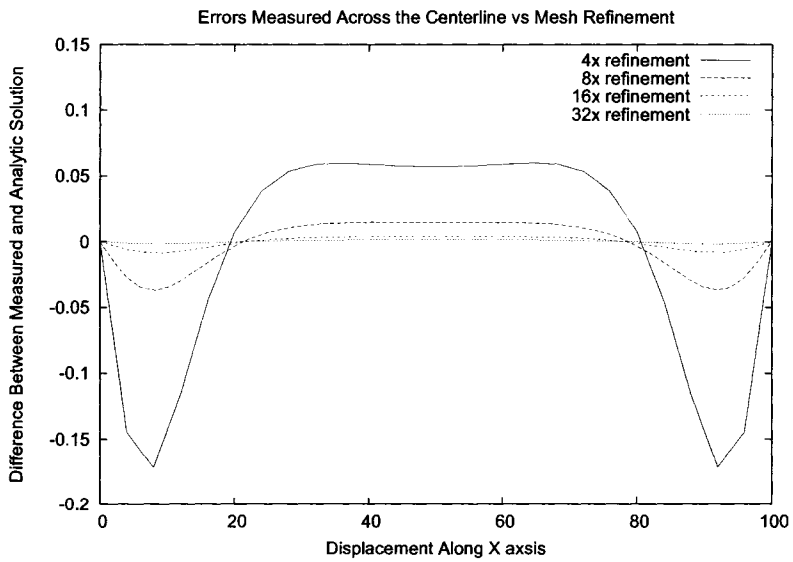


Figure 4.3: Errors Measured at Center of the Simulation Domain

To get a more qualitative sense of how the solution changes with increased mesh density Figure 4.3 shows the reduction of error across the middle of the simulation domain. It is interesting to notice that in this figure the errors tend to be at a maximum near the east and west edges of the simulated domain. In addition the errors are not uniform throughout the domain. This is most likely due to the already mentioned difficulties in simulating the exact point where the discontinuities in the boundary conditions meet.

4.3 Heat Flow in a Metal Bar

Heat diffusion as it evolves over time can be represented in terms of a parabolic PDE. Heat flows from regions of high temperature to those of low temperature. The analytical part of this presentation is taken from [34, 109]. The rate of heat flow through some material is proportional to the gradient of the temperature T within the material

$$H = -K \bigtriangledown T(x,t)$$

where K is the thermal conductivity of the material.

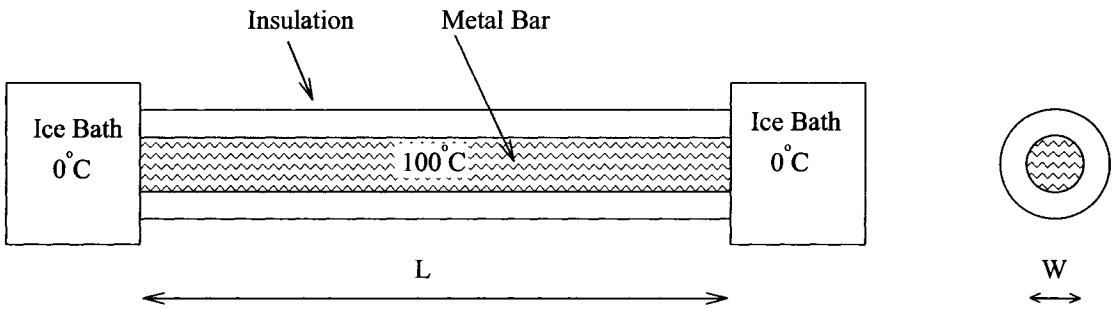


Figure 4.4: An Insulated Metallic Bar with Either End in an Ice Bath

The total amount of heat energy $Q(t)$ in the material at any one time is proportional to the integral of the temperature over the volume of the material

$$Q(t) = \int dx C\rho T(x,t) \tag{4.25}$$

where C is the specific heat and ρ the density of the material. Because energy is conserved, the rate of decrease of Q with time must equal the amount of heat flowing out of the material. When this energy balance is struck and the divergence theorem applied, the heat equation is the result

$$\frac{\partial T(x,t)}{\partial t} = \frac{K}{C\rho}\nabla^2 T(x,t) \tag{4.26}$$

assuming that the material has a constant density ρ. Equation 4.26 is a parabolic PDE with space and time as independent variables. The setup of this problem implies that there is no temperature variation in directions perpendicular to the bar, and so there is only one spatial coordinate to consider for this PDE. The one dimensional version is written

$$\frac{\partial T(x,t)}{\partial t} = \frac{K}{C\rho}\frac{\partial T(x,t)}{\partial x^2} \tag{4.27}$$

The initial temperature of the bar is given in addition to a pair of boundary conditions

$$T(x, t = 0) = 100$$

$$T(x = 0, t) = T(x = L, t) = 0$$

4.3.1 Analytic Solution

The analytic approach is similar to the one presented in the previous example and is based on the assumption that a solution exists in which the time and space dependencies occur as separate functions. The resulting pair of ODEs is

$$\frac{d^2X(x)}{dx^2} + \lambda^2 X(x) = 0 \tag{4.28}$$

$$\frac{d\Theta(t)}{dt} + \lambda^2 \frac{K}{C\rho}\Theta(t) = 0 \tag{4.29}$$

where λ is a constant to be determined. The boundary conditions at either end of the rod suggest that the solution to the spatially dependent function $X(x)$ is

$$X(x) = Asin(\lambda x) \tag{4.30}$$

The requirement that the temperature vanish at $x = L$ determines the possible values for the constant λ

$$sin(\lambda L) = 0 \Rightarrow \lambda = \lambda_n = \frac{2\pi}{L}, \ n = 1, 2, 3, ...$$

$$\Theta(t) = e^{-\lambda_n^2 t/C\rho} \tag{4.31}$$

In this case the principle of linear superposition holds. A solution using all the values of n can be written as

$$T(x,t) = \sum_{n=1}^{\infty} A_n sin\left(\lambda_n x\right) e^{-\lambda_n^2 t/C\rho} \tag{4.32}$$

where n can be any odd integer and A_n is an arbitrary constant. The Fourier expansion coefficients are determined by the initial condition that at time $t = 0$ the entire bar has a temperature of $T = 100$. Using T_0 as the initial temperature, and observing that even numbered harmonics cancel due to symmetry, the full solution is an infinite series

$$T(x,t) = \sum_{n=1.3,5,\ldots}^{\infty} \frac{4T_0}{n\pi} e^{-n^2\pi^2 Kt/(L^2 C\rho)} sin\left(\frac{n\pi x}{L}\right) \tag{4.33}$$

4.3.2 Finite Difference Solution

While the previous problem solved the finite difference solution on a closed two dimensional mesh this is not necessary to solve the parabolic problem numerically. Since there is no predefined limit to the time that one might wish to simulate and since the dependence of the solution flows in one direction only, the modeling domain can be represented by two one dimensional vectors.

In terms of the MOOSE framework this means that two copies of the simulation solution vector and matrix are needed. For this example a copy index and a distinct equation group number are needed. Each solution copy is associated with a single solution vector; the equation group determines what operation is applied to that vector. In the previous example this distinction was unimportant since there was only one solution vector, and one operation applied to that solution. The solution procedure for the MOOSE framework is similar to the previous example:

1. Create a data structure with 1 element T, to store the computed temperature
2. Create 2 cell types
   (a) A variable temperature cell

(b) A 0 degrees Celsius cell

3. Create equations for each cell type

4. Draw the geometry for the problem

5. Create the solver program which initializes the problem, solves it, and plots it.

For this problem the *sup[]* operator is used. This operator indicates that a superscript is being employed, where the index of the superscript refers to values derived from another time frame, in this case represented by a separate copy of the solution vector. The formulation in section 4.1.3 uses superscripts to indicate different time references, this is where the *sup[]* notation was derived from.

Algorithm 2 Transient PDEs

```
if eq_grp = 1
     then
     PDEs:=[ [T=sup[T,2]+h*K/(C*p)*LAPL(T), T] ];
elif eq_grp = 2
     then
     PDEs:=[ [T=sup[T,1]+h*K/(C*p)*LAPL(T), T] ];
end if;
```

The PDEs in Algorithm 2 and variable pairs follow the analytical specification of the problem. The equation `T=sup[T,2]+h*K/(C*p)*LAPL(T)`, should be read $T^{(1)} = T^{(2)} + h \cdot K/(C \cdot p) \cdot \nabla^2 T^{(1)}$. Notice how in this case specifying T as the variable to be solved for makes a difference, since there are other variables in the equation which are simply constants. The constants will appear symbolically in the matrix generator code, the user can define their values through global variable definitions. The variables which are not specifically identified by superscripts are inferred to apply to the current solution vector copy. This equation should be compared with equation 4.3 and equation 4.26.

For this example the solution mechanism always applies equation group 1 to vector copy 1, and equation group 2 to vector copy 2. The precise solution regime is not predetermined by the MOOSE framework, this solution mechanism is presented as a typical formulation. The solution methodology follows that described in Chapter 3, where a pair of vectors are constructed and a pair of equation are specified and used in alternating succession. Notice that the equations in group 1 refer by superscripts to values derived in copy 2. Similarly equations in group 2 refer to values derived from copy 1.

Recall that the equations specified by the user are not applied directly, but rather they are translated by the MOOSE framework into a matrix representing a system of equations which can be solved by a variety of linear solvers which are compatible with the framework. Also recall that part of the work that the MOOSE framework does is to seamlessly solve issues related to moving meshes as they may occur in a transient finite difference simulation. Notice that the above equations only refer abstractly to variables and vector copies, the MOOSE framework handles the details of translating these PDEs between the numerical space of the linear solver, and the representative space of the spatial model.

These equations are implicit in space and explicit in time, as compared with the equation 4.3 which is explicit in both space and time. Using a formulation which is implicit in space provides better stability properties at the cost of a more complex solution method, as was discussed in the previous chapters. By using a two vector implementation the memory consumption of the solver remains the same no matter how much time is simulated, and any number of time steps can be modeled.

The solver program is similar to the previous one, except that it must define an initial condition, and apply a sequence of steps to solve the problem. The user can change the step size h during the course of this execution using step size doubling

Algorithm 3 Transient solver For Heat Equation

```
Initialize_Model(xdim=1,ydim=0.1, Copy=1, Equation_Group=1);
Initialize_Model(xdim=1,ydim=0.1, Copy=2, Equation_Group=2);
write_all(Copy=1, Variable=T, 100); // set the initial temperature

t=0;
while(t < 100) {
     solverdr_solve(Copy=2, Equation_Group=2);
     solverdr_solve(Copy=1, Equation_Group=2);
     t=t+2*h;
}
```

as discussed in chapter 2, or any other step size estimation technique, although code for adjusting the step size is not presented in this example. The example code in algorithm 3 shows how the two separate model copies are initialized and solved for.

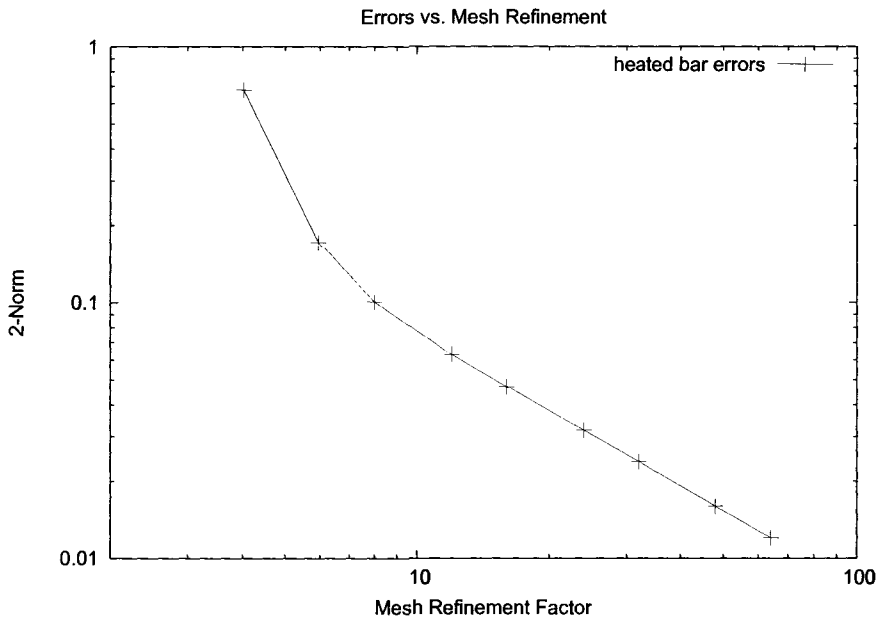


Figure 4.5: Errors versus Mesh Density

For this example a constant time step is used. The simulation is run for 1000 seconds and the constants K, p and C are chosen to reflect the physical constants for iron. Using the same techniques presented in the previous example the order of the spatial terms is determined to be 1.9. The plot which compares errors versus mesh refinement is illustrated in Figure 4.5. This plot is different in a few respects to the previous example. The slope of the curve is different since the mesh is one dimensional, hence the ratio of cells sizes is different. Also the problem is of a fundamentally different type, so the initial segments of these curves where the problem performs poorly angle in opposite directions.

4.4 Wave on a String

The wave equation is an example of a hyperbolic PDE. This thesis has not placed much emphasis on the study of hyperbolic PDEs, however the MOOSE framework is capable of handling them as this example will illustrate. Hyperbolic PDEs have their own special set of difficulties, and while the MOOSE framework supports the fundamental primitives necessary for their implementation modeling hyperbolic functions is not currently one of the frameworks' strengths. The analytical solution is partially derived from [109].

Consider a string of length l, tied down at both ends . The string has a constant density per unit length ρ, a constant tension τ, and is subject to neither friction nor gravitational forces. The vertical displacement of the string from its rest position is described by a function of two variables $y(x,t)$, where x is the horizontal location along the string and t the time. The string is only displaced in the vertical direction.

To derive a linear equation of motion it is assumed that the displacement and slope of the string are small. An infinitesimal section Δx of the string is isolated. From

Newton's equations the second law of motion indicates that the sum of the vertical forces on the string section must equal the mass times the vertical acceleration of the section

$$\sum F_y = \rho \triangle x \frac{\partial^2 y}{\partial t^2} \tag{4.34}$$

the forces are the components of the string's tension τ. The vertical components of the tension on each end of the segment change as the angle of the string changes, and those components are obtained by relating the slope of the string to $\frac{\partial y}{\partial x}$

$$\sum F_y = \tau \left[\left(\frac{\partial y}{\partial x} \right)_{x+\Delta x} - \left(\frac{\partial y}{\partial x} \right)_x \right] = \tau \frac{\partial^2 y}{\partial x^2} \tag{4.35}$$

$$\frac{\partial^2 y(x,t)}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 y(x,t)}{\partial t^2} \tag{4.36}$$

The propagation speed c is denoted by

$$c = \sqrt{\tau/\rho} \tag{4.37}$$

Since both ends of the string are tied down, the boundary conditions are that the displacements must vanish for all times at the end of the string. The initial condition at $t = 0$ is represented by the plucking of the right side of the string. The plucking of the string is modeled by the following function

$$y(x, t = 0) = \begin{cases} 1.25x/l & for\ x \leq 0.8l \\ 5.0(1 - x/l) & for\ x > 0.8l \end{cases} \tag{4.38}$$

Because the model uses a second order equation in time, a second initial condition is needed to determine the solution. The second initial condition is that the plucked

string is released from rest

$$\frac{\partial y}{\partial t}(x, t = 0) = 0$$

4.4.1 Analytic Solution

The analytic solution is obtained via separation of variables. As before the wave equation is assumed to be a product of a function of two functions, one in space, the other in time.

$$y(x, t) = X(x)T(t) \tag{4.39}$$

Solutions to the following two ODEs are needed,

$$\frac{d^2T(t)}{dt^2} + \omega^2 T(t) = 0 \tag{4.40}$$

$$\frac{d^2X(x)}{dt^2} + k^2 X(x) = 0 \tag{4.41}$$

Where $k = \frac{\omega}{c}$. The angular frequency ω and the wave vector k are determined by demanding that the solutions satisfy the boundary condition which specifies that the string is attached at both ends.

The corresponding solution for the time equation is

$$T_n(t) = C_n sin(\omega_n t) + D_n cos(\omega_n t) \tag{4.42}$$

$$\omega_n = n\omega_0$$

$$\omega_0 = ck_0 = \frac{2\pi c}{l}$$

The preceding solutions are the nth normal modes where by definition, each mode oscillates at a single frequency. The initial condition requires the C_n values to be zero. For a string with its ends fixed and initially at rest, there are solutions of the wave equation of the form

$$y(x,t) = \sum_{n=1}^{\infty} B_n sin(k_n x) cos(\omega_n t) \tag{4.43}$$

The Fourier coefficients B_n are determined by using the first initial conditions which describes how the wave is plucked. They are found to be

$$B_n = 12.5 \frac{sin(0.8\pi n)}{n^2\pi^2} \tag{4.44}$$

The final series is

$$y(x,t) = \sum_{n=1}^{\infty} 12.5 \frac{sin(0.8\pi n)}{n^2\pi^2} sin(\pi n x/l) cos(\sqrt{\tau\rho}\pi n t/l) \tag{4.45}$$

4.4.2 Finite Difference Solution

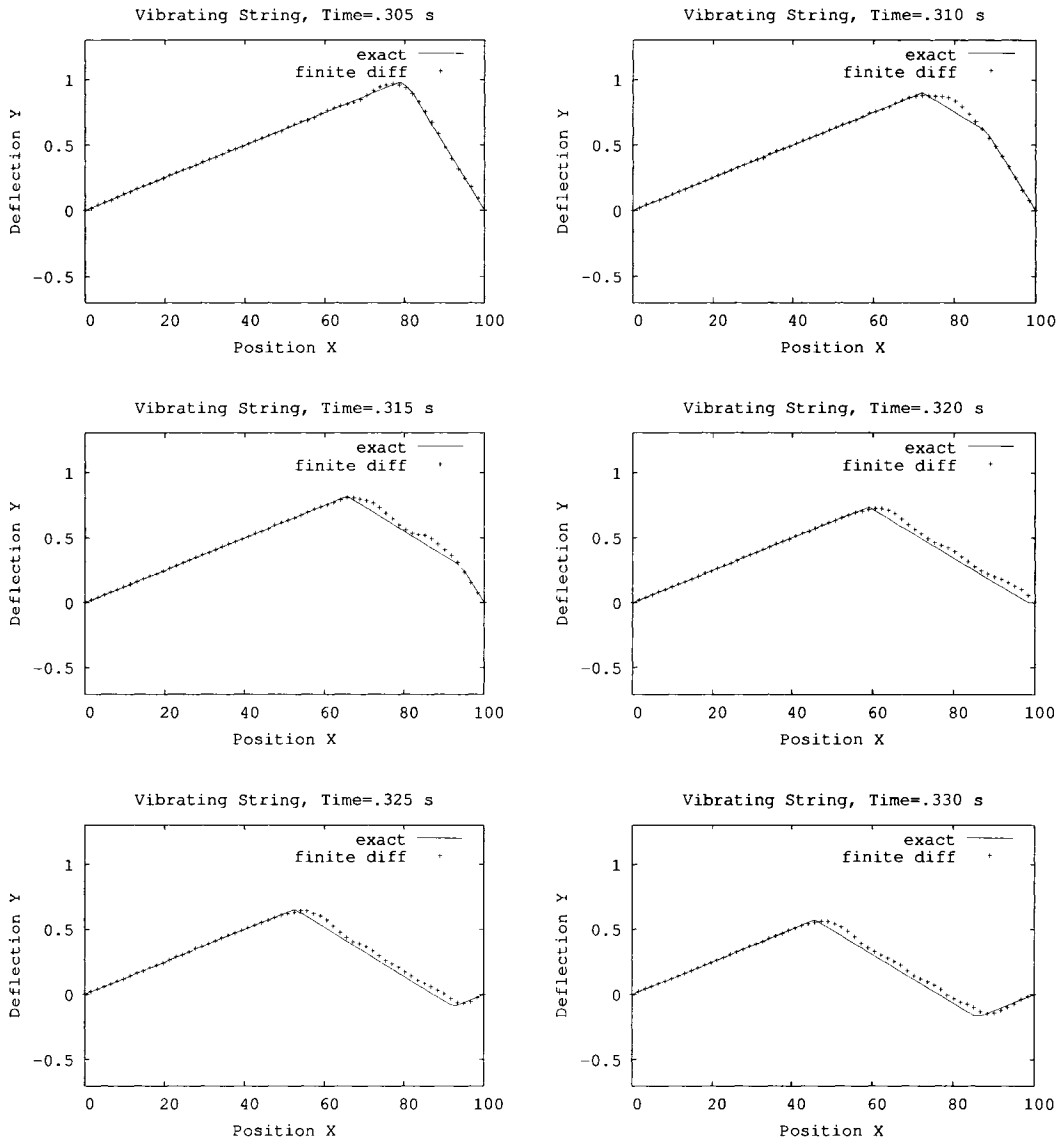


Figure 4.6: Solutions to the Wave on a String Problem

Hyperbolic problems tend to suffer from numerical errors in ways quite different from elliptic or parabolic problems. The vast literature on fluid dynamics and tech-

niques for solving problems related to the Navier-Stokes equations are a testament to this. Hyperbolic solutions can exhibit wave fronts and other sharp discontinuities which are difficult to model especially over many iterations.

Algorithm 4 Hyperbolic PDEs for the Wave on a String Problem

```
if eq_grp = 1
     then
     PDEs:=[[Y=2*sup[Y,3]-sup[Y,2]+h*h*Tau/p*LAPL(sup[Y,3]), Y] ];
elif eq_grp = 2
     then
     PDEs:=[[Y=2*sup[Y,1]-sup[Y,3]+h*h*Tau/p*LAPL(sup[Y,1]), Y] ];
elif eq_grp = 3
     then
     PDEs:=[ [Y=2*sup[Y,2]-sup[Y,1]+h*h*Tau/p*LAPL(sup[Y,2]), Y] ];
elif eq_grp = 4
     then
     PDEs:=[ [Y=sup[Y,1]+.5*h*h*Tau/p*LAPL(sup[Y,1]), Y] ];
end if;
```

The PDEs used are presented in their MOOSE notation form in algorithm 4. The implementation is second order in space and in time, a simple explicit formulation was used. The heat diffusion problem, discussed in the previous section, used two equation groups and two vector copies to model the transient phenomena of heat diffusion. For the wave equation three vector copies are used with four equation groups.

Three vector copies are needed to represent a centered finite difference formulation in time. Each one of the vector copies represents a different instance in time, and the set of three equation groups must be solved in a cyclic fashion similar to the solution strategy used in the preceding example.

The first three equation groups are similar to the two equation groups used to solve the heat equation, except that each group refers to variables in two other vector copies. The fourth equation group is a special case which allows the simulation to

be initialized. The fourth equation group assumes that the initial condition has been stored in the vector which represents the first simulation copy. It uses a discretization which is first order in time to compute the string position for the second vector copy. The first initial condition describes the shape of the string immediately after being plucked and provides a basic triangle wave form for the string. This initial condition was coded inside the main solver program, although in principle it could have been also represented by a fifth equation group. Using a separate equation group to initialize a model was discussed abstractly in the third chapter.

The simulation results are presented in Table 4.6 for six snapshots of the evolution of the simulation. For this simulation each time step was equivalent to 1/1000th of a second, the first figure in the table is computed after .305 seconds have elapsed and shows the vibrating string in the position that it was in when the simulation was started both for the analytical case, the solid line, as well as the finite difference model. The string retains a triangle shape throughout its vibration because the model is frictionless. Each subsequent figure shows the evolution of the model in .005 second increments. As this model progresses in time its shape will tend to deteriorate and it will diverge further and further from the correct solution. Even in the first 1/3rd of a second irregularities in the solution are beginning to appear, especially near the wave front, and along the trailing edge of the wave.

4.5 Verification of Patched Mesh Linking Rules

A critical problem in building the patched mesh matrix generation code was ensuring that the mesh linkages do not introduce excessive errors into the eigenvalue solution. For the purposes of verification a series of steady state models were developed and tested at various resolutions using a variety of test criteria. This section will present

two representative tests which illustrate the degree to which the conservation rules are able to correct errors, and under what circumstances. Error corrections achieved through conservation rules are highly problem dependent, although certain trends remain consistent across most tests. During the course of the development of the MOOSE libraries hundreds of test cases were studied in the attempt to establish a simple and meaningful rule-set. The rules described at the end of Chapter 3 will be illustrated in this section with two examples.

4.5.1 Geometric Conservation Rule Verification

The first example is based on the electrostatics problem presented earlier in this chapter. This problem is used to illustrate the geometric requirements of linked meshes by examining the errors induced by linking two meshes of different resolution.

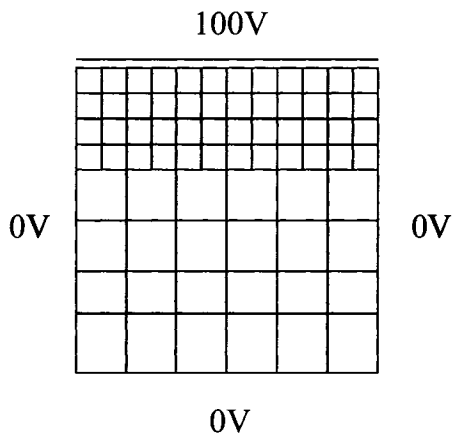


Figure 4.7: Partitioned Electrostatic Problem

Figure 4.7 shows a mesh construction which uses a doubly refined mesh near the top 1/3rd of the problem closest to the 100V potential and a less refined mesh for the rest of the problem. The intuition behind such a mesh partition is that the solution has a higher gradient in the top portion of the mesh, and hence requires more points

to accurately model its behaviour. Errors which result from the mesh connection strategy will be compared with the closed form solution. Errors for individual points are weighted by the cell area, so that errors in larger cells have a bigger impact than errors in smaller cells.

Top Mesh 18x6				
Bottom Mesh Dimensions	Total Points	Linear Errors	NonLinear Errors	Conservation Errors
18x12	324	1.53	1.53	1.53
15x10	258	10.2	10.4	1.67
13x9	225	22.9	19.0	1.67
10x7	178	53.9	41.7	2.07
9x6	162	66.1	55.7	2.58
6x5	136	116.9	107.9	4.72

Top Mesh 36x12				
Bottom Mesh Dimensions	Total Points	Linear Errors	NonLinear Errors	Conservation Errors
36x24	1296	.41	.41	.41
30x20	1032	5.13	4.99	.41
27x18	918	10.0	8.31	.42
21x14	726	24.5	18.7	.55
18x12	648	33.0	27.5	.74
14x9	558	54.6	48.8	1.10

Table 4.1: Mesh Connection Errors for Two Resolutions

Table 4.1 tabulates error measurements for two different starting resolutions for the electrostatics problem. The first column specifies the mesh dimensions for the bottom 2/3rds of the mesh. The top 1/3rd of the mesh remains constant in dimensions for both tests. The second column gives a count of the total number of points in the mesh. The next three columns tabulate measured errors for three different mesh connection strategies. Linear errors are those errors measured when only linear interpolation is used to connect meshes. NonLinear errors are the errors measured when only nonlinear

interpolation methods are used to connect the mesh. Recall from the previous chapter that nonlinear methods alone will only produce reasonable results when meshes are out of alignment, not when mesh cells on either side of a boundary are of different sizes. This test clearly illustrates this phenomena. The final column shows the measured error when the geometric conservation rule is used to link meshes. It should be clear that for both examples even when a large reduction in the number of points is used to model the simulation domain, the impact on the measured error for the conservation geometric rule is very moderate.

It is interesting to compare several cases. Take for example the 36x36 case where 648 points are used to compute a result. The error generated by this formulation without conservation using nonlinear interpolation is an order of magnitude worse than the error generated in the 18x18 case using 324 points with no special refinement or connection strategies. This case shows that haphazard mesh interconnections may not produce results which are any better than those which can be derived with standard regular meshes.

It is important to keep in mind that this example problem is constructed specifically to highlight a situation where nonlinear interpolation alone fails to provide satisfying results for a simple mesh interconnection strategy. For this problem conservation was used to correct errors which appeared not only between the top and bottom meshes, but also between the bottom mesh and the 0V boundary condition mesh, which was modeled at the same resolution as the top mesh. From the previous discussion on the electrostatics problem errors tend to be high along the left and right 0V boundaries. As the next set of test results indicates, geometric issues are not necessarily the primary concern for certain models since the user can arbitrarily control mesh depth. Other model details, in particular, moving material discontinuities, may be part of the problem definition and more difficult to compensate for.

4.5.2 Material Discontinuity Conservation Rule Verification

In situations where material discontinuities must be dealt with along mesh boundaries errors can arise when poor estimates for diffusion constants are used. As discussed in the implementation chapter, these errors can be avoided by selecting the direction for which the conservation rule is applied. For this example a two dimensional eigenvalue problem is chosen based on the neutron diffusion problem introduced in chapter 2. Although no closed form solutions exist for the two dimensional problem, a simplified one dimensional problem taken from [53] illustrates some concepts.

A simplified version of the transient neutron diffusion equation 2.7 which neglects the delayed precursor source terms and is expressed in only one spatial dimension and with only one energy group can be written as

$$\frac{1}{v}\frac{\partial \phi}{\partial t} - D\frac{\partial^2 \phi}{\partial x^2} + \Sigma_a \phi(x,t) = \nu\Sigma_f \phi(x,t) \tag{4.46}$$

As with the previous problems in this chapter, separation of variables is used

$$\phi(x,t) = \psi(x)T(t) \tag{4.47}$$

It is possible to rearrange equation 4.46 by substituting equation 4.47 to derive

$$\frac{1}{T}\frac{dT}{dt} = \frac{v}{\psi}\left[D\frac{\partial^2 \psi}{\partial x^2} + (\nu\Sigma_f - \Sigma_a)\psi(x)\right] = constant = -\lambda \tag{4.48}$$

The spatial component of equation 4.46 can be isolated and written as

$$D\frac{d^2\psi}{dx^2} + \left(\frac{\lambda}{v} + \nu\Sigma_f - \Sigma_a\right)\psi(x) = 0 \tag{4.49}$$

As a sample problem consider a one dimensional infinite slab reactor which has width a and fixed boundary conditions $\psi\left(\frac{a}{2}\right) = 0$ and $\psi\left(-\frac{a}{2}\right) = 0$. λ is still to be

determined. The eigenvalue problem

$$\frac{d^2\psi}{dx^2} + B_n^2\psi(x) = 0 \tag{4.50}$$

has symmetric solutions for

$$\psi_n = cos(B_n x) \tag{4.51}$$

$$B_n^2 = \left(\frac{n\pi}{a}\right)^2 \ , \ n = 1, 3, 5, ... \tag{4.52}$$

where λ_n is chosen to be

$$\lambda = v\Sigma_a + vDB_n^2 - v\nu\Sigma_f \equiv \lambda_n \ , \ n = 1, 3, 5, ... \tag{4.53}$$

The fundamental mode for the idealized slab reactor is a rough approximation of the neutron flux shape which more complex problems have. The shape of the flux profile for the slab reactor is characterized by the cosine

$$\phi(x) = cos\left(\frac{\pi x}{a}\right) \tag{4.54}$$

Characteristics of the eigenvalue steady state neutron diffusion solution involve searching for the lowest eigenvalue. The corresponding eigenvector is symmetric, all elements of the vector are the same sign, normally represented as positive. The maximum flux value occurs roughly in the center of the problem domain, the minimum flux value is normally zero and is normally located at the edge of the problem domain and can be represented by a fixed boundary condition.

A hypothetical rectangular core is modeled in two dimensions shown in figure 4.8.

The PDE which governs diffusion for this problem is

$$-\bigtriangledown \cdot D \bigtriangledown \phi + \Sigma_r = \frac{1}{k}\Sigma_f \phi \tag{4.55}$$

The cross section and reaction rate constants were chosen in an artificial way so that the eigenvalue solution would be exactly 1. One edge of the core is bounded by a mesh discontinuity. The mesh is divided in two sections, but the top and bottom mesh sections are of equal resolution. A wrap around geometry is used to connect the north and south edges of the mesh as well as the east and west edges of the mesh as illustrated in Figure 4.8.

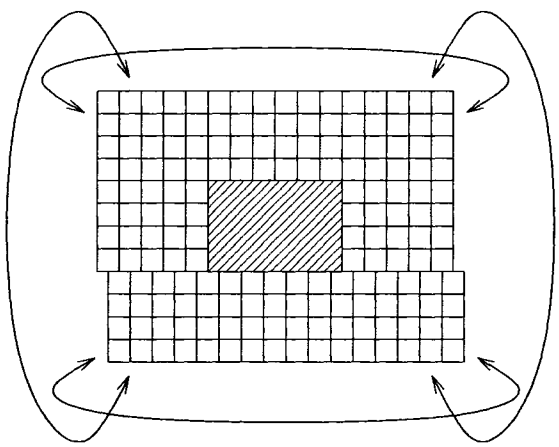


Figure 4.8: Partitioned Moving Mesh with Wrap Around Geometry

Wrap around geometry settings are convenient for some problems which involve moving meshes since they allow cells which leave one side of the simulation domain to re-enter on the opposite side. It is also possible to squash and extend intermediate mesh components to facilitate motion, however, wrap around geometries provide the simplest implementation for moving mesh components. For this example repositioning the mesh does not actually change the problem. While no closed form solution exists

for this problem it should be obvious that if the mesh interconnection strategy is ideal, shifting the lower mesh by any fraction should not change the fundamental eigenvalue. The eigenvalue computed when the cells in the top and bottom mesh are aligned is considered to be correct, and any deviation which is a consequence of shifting the mesh is considered to be an error.

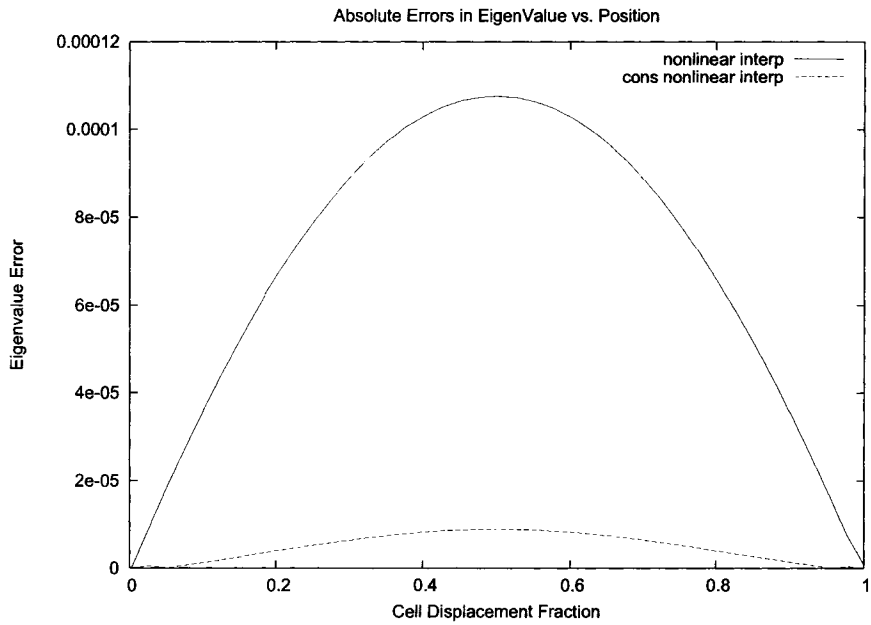


Figure 4.9: Motion Error

Figure 4.9 illustrates the deviation in computed eigenvalues for two mesh connection strategies. The first strategy uses nonlinear interpolation alone, the second strategy uses conservation rules to avoid estimating diffusion constants for cells which must handle material discontinuities. Nonlinear interpolation is used to connect all other cells. This result is quite interesting since at any given displacement for the lower mesh there will be no more than 4 estimated diffusion constants, yet by estimating flow through the use of conservation rules instead of the estimated constants an error reduction for this problem of a factor of twenty is possible.

4.6 Discussion

This chapter has presented a some basic theory behind verification strategies combined with a collection of problems which illustrate how the MOOSE framework was verified. Closed form solutions provide a rigorous benchmark to compare computed numerical solutions against. Using closed form solutions to verify a simulation model is somewhat limiting in the sense that only certain problems can be studied in this way.

This chapter should also clarify certain questions related to the MOOSE framework's usage. Some of the examples presented in Chapter 3 were described in a necessarily abstract terminology in order to capture the generality of the framework's capabilities. Comparing the discussion in Chapter 3 with the simplified concrete examples in this chapter should illustrate how the MOOSE framework handles various simulation types.

The verification of the geometric conservation rule and the material discontinuity conservation rule presented in the previous two sections represents some of the tests performed on the framework to verify its correct behavior. Any framework which attempts to implement patched moving meshes should be tested under at least similar circumstances. The scenarios presented in the preceding two sections are idealized and only apply in certain situations. When choices must be made between the material discontinuity conservation rules and the geometric conservation rules, it is often best to choose in favour of satisfying the material discontinuity rules. This choice is largely heuristic and to a certain extent will be determined by the exact problem formulation. Conflicts can arise such that it is not possible to satisfy either the material discontinuity rule or the geometric rule. The rules also depend on the specific geometry of the problem, the number of cells and the size of the cells.

The overall impact of using conservation rules in a realistic scenario is presented in the next chapter within the context of the rod insertion case study. As will be demonstrated, despite the potential for conflict in the rule-set, generally very good error reduction can be achieved.

Chapter 5

Simulation Studies

This chapter presents a sequence of studies examining a fuel assembly insertion experiment, similar to the fuelling incident that occurred at the McMaster Nuclear Reactor in January 1994. The January 1994 fuelling incident involved the insertion of a fuel assembly worth an estimated 24.8 mk[1] over an estimated 20 second period to a partially assembled core. The core had an initial k_{eff} of 0.983 and an initial power of 13mW. The point kinetics models used at the time concluded that the best estimate peak power was approximately 8.4 MW.

This chapter describes a sequence of related simulation application built with the MOOSE framework. The techniques provided by the MOOSE are leveraged in this chapter for the development of a simulation study, as well as for the verification of the MOOSE.

The goal of this chapter is to construct a reasonably accurate two dimensional approximation of the refueling incident that will execute within a reasonable period of time. The first section describes the calibration and simulation setup that was used. The second section will explore numerical stability and accuracy issues using various

[1]mk is a relative unit of measure corresponding to 1/1000th of k_{eff}, see [53] for a complete discussion of reactivity measurements

approximations. The third section will present a series of transient simulations using the best solutions provided in the previous sections.

The goal of these simulations is to generate a good solution in a short period of time. Since it is always possible to reduce the size of time steps and increase the number of cells used to model the scenario an effort has been made to examine the coarsest approximations that remain convincing.

5.1 Simulation Parameters

This section discusses the various simulation parameters which were used to configure the reactivity insertion model. Approximations, calibrations, and various simplifications to the general model are summarized.

5.1.1 Approximations

Because the goal of this study is to examine the effect of the use of moving grids on the transient neutron diffusion equation some latitude has been taken with a few of the classical simulation elements. The conclusions that are drawn regarding the MOOSE's methodology should apply equally well to a more rigorously configured simulation which makes fewer assumptions. The 1994 refueling incident caused several of the data recorders to go off their scales, so no actual measured data are available for comparing simulated peak power with the actual event.

The most significant approximation used in this study is that of examining a two dimensional view of the simulation scenario rather than a full three dimensional view. Since the principle point of reference is the zero dimensional point kinetics study the two dimensional study is presented with some confidence that it will provide more detail and some additional insights into the spatial components of the reactor core.

Several special core elements are not represented, including reflectors, the beryllium sources, and sample injection points. Burn-up of the core is treated in a very general way. Cross section constants were computed so that the burn-up of the core was approximately uniform at around 25%.

Cross Section Data

Although the simulation constants were extracted from the WIMS data base, the WIMS transport codes were not used to either collapse the constant groups or to generate constant mixtures. A simplified student WIMS data base was used. This 69 group data base was originally compiled by Jeremy Whitlock in 1992 [171]. A 69 group diffusion study for a simple core geometry was run to generate a detailed flux distribution for the MNR. This flux distribution was used to collapse the 69 group data base into smaller groupings of 12, 8, 5, 4, 3 and 2 energy levels. This flux spectrum and the location of the top boundaries of the 12 energy group divisions is plotted in Figure 5.1.

The energy group divisions are based on a technical document [46] written by Simon Day, and correspond roughly to energy group divisions used in MNR simulations today. Table 5.1 lists the 12 energy levels of the largest grouping, and shows how the smaller groups represent unions of the larger group divisions. Simulations run under the two group approximation corresponded with the other simulations best when the 821000 eV upper boundary was used for its thermal group. The note in the left column of the table refers to the discussion of the rationale of the selection of the energy boundary in Simon Day's technical document[2].

[2]Simon Day provided a great deal of assistance in the development of the simplified cross sections used in this thesis. Simon recommended against using WIMS, the transport theory based cross section collapsing tool used at the MNR, due to the amount of time that would have been required to understand it. Many thanks to him for his patience and hours answering questions on these issues and suggestions for developing simplified, but reasonable alternative data.

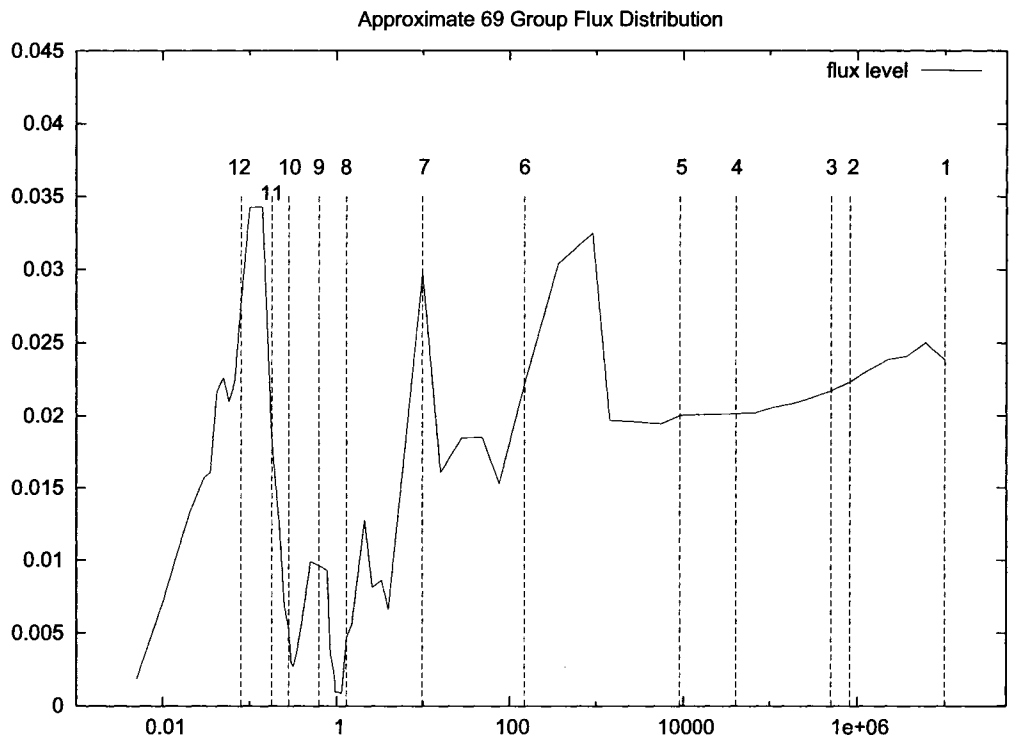


Figure 5.1: Flux Distribution used for Simplified Group Collapsing

Bin Label	eV	12Grp	8Grp	4Grp	2Grp	Note
1	1E+7	x	x	x	x	upper level of 1st group
2	821000	x	x	x	x	IAEA TECDOC233
3	500000	x				fission threshold of U238
4	41000	x				scattering cross sect. of H
5	9118	x	x	x		constant diffusion coef
6	148	x				
7	9.87	x	x			
8	1.3	x	x			handles PU240 resonance
9	0.625	x	x	x		thermal epithermal division
10	0.28	x				
11	0.18	x	x			Burnable Absorber
12	.08	x	x			

Table 5.1: Energy Groups

The generation of multi-group constants using a transport code like WIMS can be an extremely difficult and time consuming task, and so it was avoided. In addition to not using a transport code to collapse the constants simple weighted averages were used to mix the cell constants, and all materials were assumed to be at room temperature. Such techniques cannot take certain phenomena into account, like quantum resonance effects, or temperature related Doppler shifts which occur for some material mixtures and under certain operating conditions, and so the constants used for this study have limited validity.

The energy spectrum of delayed neutrons from thermal neutron induced fission of U235 is the poorest known of all input data in reactor calculations. Delayed neutrons are born at a lower energy than their prompt counter parts. Doroshenko [50] discusses the difficulties in measuring the data as well as techniques for approximating it analytically. For the experiments in this chapter it was roughly estimated that for the composite delayed spectrum 20% of all neutrons are born above the .821 MeV threshold, 30% are born between .821MeV and .5MeV, the remaining 50% of the delayed neutrons are produced between the .5 MeV and 41 KeV.

Despite the various approximations there is no reason to believe that the cross sectional data used in this chapter is inappropriate for comparing simulation techniques. Since this study is an experimental one which examines new numerical methods for modeling motion, focus was placed on the relative accuracy of each model, rather than on the precise correspondence of the model with absolute measurements.

5.1.2 Calibration

To compensate for errors introduced by the simplified cell collapsing techniques and the approximate two dimensional interpretations, models were designed so that each

principle feature could be adjusted. Steady state models were used to checkpoint various rod positions and the core simulation was calibrated by adjusting certain constants. The checkpoints include

1. Shim inserted 13%, fuel inserted 0%, $k_{eff} = .983$
2. Shim inserted 13%, fuel inserted 100% $k_{eff} = 1.009 \ +/- \ .0015$
3. Shim inserted 100%, fuel inserted 100% $k_{eff} = 0.9195$

The simulation is most sensitive to the maximum estimate of k_{eff} . While error estimates in the low values for k_{eff} are ignored in this study, an error of about 10% in the estimate of the worth of the fuel assembly is taken into account, which is reflected in the second calibration point. These errors are applicable in the calibration of the transient model and are discussed in the next section.

Steady State Calibration

The steady state simulation is calibrated in three different ways. The worth of the shim rods is only approximately specified for this problem and is cited as between 75mk and 100mk in the MNR safety report, depending on fuel loading patterns, and fuel burn up. For these experiments 88mk was used as the insertion worth. The precise composition of the shim rods was adjusted to alter their worth. In the two dimensional representation of the core two shim rods are used to represent the 5 shim rods and one regulating rod that are present in the actual reactor. The cross sectional constants which represent the shim rod composition of 80% Ag, 15% In, 5% Cd are averaged with a set of constants which replace Al for the materials which would normally absorb neutrons. The initial position of the shim rod is set at about 13% insertion, or 8 cm.

The worth of the inserted fuel assembly is calibrated by adjusting the burn up of the stationary fuel assemblies in the core. This technique was chosen rather than directly adjusting the burn up of the inserted assembly to keep the experiments as uniform as possible. For each test case the inserted assembly, and the assemblies directly to its left and its right are set at 25% burn up. Since the focus of many of the tests investigates the details of the interaction between the moving fuel assembly and its neighbours it is important that the cross sections which represent these components remain fixed throughout all the tests. Calibrating the core by adjusting the burn-up of the stationary assemblies reduced the impact of calibration on the comparison of tests.

Fine calibration of the initial steady state model was done using a floating point constant $fcal$ which was multiplied by the fission spectrum term χ_g to adjust the overall reactivity of the fuel. The steady state version of the neutron diffusion equation including the calibration term is written as

$$-\bigtriangledown \cdot D_g \bigtriangledown \phi_g + \Sigma_{Rg}\phi_g - \sum_{g'=1,g'\neq g}^{G} \Sigma_{Sg'g}\phi_{g'} = \frac{fcal_g}{k} \cdot \chi_g \sum_{g'=1}^{G} \nu\Sigma_{fg'}\phi_{g'} \qquad (5.1)$$

Calibrating a model using these various end point conditions is not very time consuming. The eigenvalue steady state problem for this model runs on a modest PC in under 60 seconds for the 5 group model so it is relatively easy to execute several variations of the problem during the model calibration.

The calibration algorithm uses a simple iterative solver which examines the differences between computed eigenvalues for various rod and assembly positions and adjusts the calibration parameters accordingly. First the algorithm adjusts the shim rod composition until it has achieved a certain degree of precision. The model alternately removes and inserts the shim rods checking the difference in the computed

eigenvalues for the 13% inserted position and the 100% inserted position.

In the second phase the algorithm adjusts the core burn-up to set the reactivity of the inserted fuel assembly. For this estimate the shim rods are withdrawn to the 13% insertion position so the fuel assembly worth can be estimated. The calibration function cycles back and forth between these two phases, adjusting the shim worth and the inserted fuel assembly worth until both have achieved the required degree of accuracy. The calibration algorithm allows for a 1/2 mk error (0.05%) in the shim worth but only a .02 mk error (0.002%) in the fuel worth.

Table 5.2 summarizes the core burn-ups that were used to achieve equality between the various models as well as the *fcal* parameter. These figures indicate that the cell collapsing techniques used to generated the various energy group constants lead to some differences between models derived at various energy groups. Comparison between results taken from simulations performed at different energy groups which differ by an amount on the order of 10% will be understood to be the result of errors induced by the collapsing procedures and subsequent recalibration.

Total Energy Divisions	Core Burn-up	$fcal_g$
g=2	33.97%	0.94403
g=4	27.28%	0.93391
g=8	28.27%	0.94217
g=12	26.75%	0.94951

Table 5.2: Core Burn-up and *fcal* Adjustments

Transient Calibration

The transient problem uses the same basic calibration points as the steady state problem, although since the transient equations are somewhat more complicated, in particular due to the inclusion of the delayed precursors and a non-zero background

radiation source term, it requires some extra adjustments. A sub-critical reactor core behaves like an amplifier with an amplification factor of $1/(1-k)$ so the additional neutrons included as part of the sub-critical source tend to change the behaviour of the core when it is close to criticality. In addition the small fraction of delayed precursors required for the control of a critical core do not correspond exactly with the prompt neutrons modeled in the steady state simulation.

A sub-critical constant flux level is added to the simulation across all energy groups to model background radiation. The reactor depends on sub-critical neutrons produced by the spontaneous fission of fuel byproducts as a neutron source for starting the reactor. Mathematically this is represented by the inclusion of a constant factor in the transient version of the neutron diffusion equation. The transient version of the neutron diffusion equation can be solved for the case where its first derivative is zero and both the delayed precursor concentrations as well as the flux levels are solved for so that they balance the low power source neutrons. This provides the initial condition for the transient calculation.

Transient neutron diffusion equations including precursor terms and constant source

$$\frac{1}{v_g}\frac{\partial \phi_g}{\partial t} = \bigtriangledown \cdot D_g \bigtriangledown \phi_g - \Sigma_{Rg}\phi_g + \sum_{g'=1,g'\neq g}^{G} \Sigma_{Sg'g}\phi_{g'} + fcal \cdot (1-\beta)\chi_g \sum_{g'=1}^{G} \nu\Sigma_{fg'}\phi_{g'} + \chi_g^D \sum_{i=1}^{6} \lambda_i C_i + S_g \tag{5.2}$$

$$\frac{\partial C_i}{\partial t} = \lambda_i + fcal \cdot \beta_i \sum_{g'=1}^{G} \nu\Sigma_{fg'}\phi_i \tag{5.3}$$

Steady state neutron diffusion equations

$$0 = \bigtriangledown \cdot D_g \bigtriangledown \phi_g - \Sigma_{Rg}\phi_g + \sum_{g'=1,g'\neq g}^{G} \Sigma_{Sg'g}\phi_{g'} + fcal \cdot (1-\beta)\chi_g \sum_{g'=1}^{G} \nu\Sigma_{fg'}\phi_{g'} + \chi_g^D \sum_{i=1}^{6} \lambda_i C_i + S_g \tag{5.4}$$

$$0 = \lambda_i + fcal \cdot \beta_i \sum_{g'=1}^{G} \nu\Sigma_{fg'}\phi_i \tag{5.5}$$

For an arbitrary low power source the ion chamber can be adjusted so that it produces a desired reading for the steady state sub-critical case. This is not the same as the eigenvalue steady state problem since precursor densities are included in this computation and the eigenvalue problem makes no assumptions about a constant source, it rather only examines the reactor's multiplication rate. The overall problem calibration is not very sensitive to the initial power, as will be discussed later.

The most important calibration point for the transient calculation is the adjustment of the reactor period, or rate of change for the case where the fuel assembly is fully inserted. Relying on the reactivity calibrations performed for the steady state case gives a first order approximation of the correct calibration for the transient case. When the fuel assembly is inserted completely, it is estimated that ρ[3] is between .0075 and .012. The precise amount of excess reactivity in the core for the case where the fuel assembly is fully inserted is quite difficult to compute, and is one of the key unknowns in the simulation.

The inhour equation is derived from a point kinetics model and expresses the relationship between the various decay constants which occur as part of the delayed precursor model and core reactivity, or rate of change of power. A discussion of this equation and its associated constants goes somewhat outside the scope of this chapter, the interested reader will find a complete presentation in [53]. This equation can be used to derive a relationship between reactor period and excess reactivity. This relationship is plotted in Figure 5.2. It can be seen that the reactor period varies quite

[3]The symbol ρ signifies reactivity, defined as $\rho = (k_{eff} - 1)/k_{eff}$. The estimate for the range of ρ is based conversations with Simon Day and notes from Wm. J. Garland's original estimates of reactor period for this event. A broad range was chosen to capture the most likely extremes for this particular event.

rapidly if $\rho < .01$, but that for values of $\rho > .012$, the period changes less rapidly.

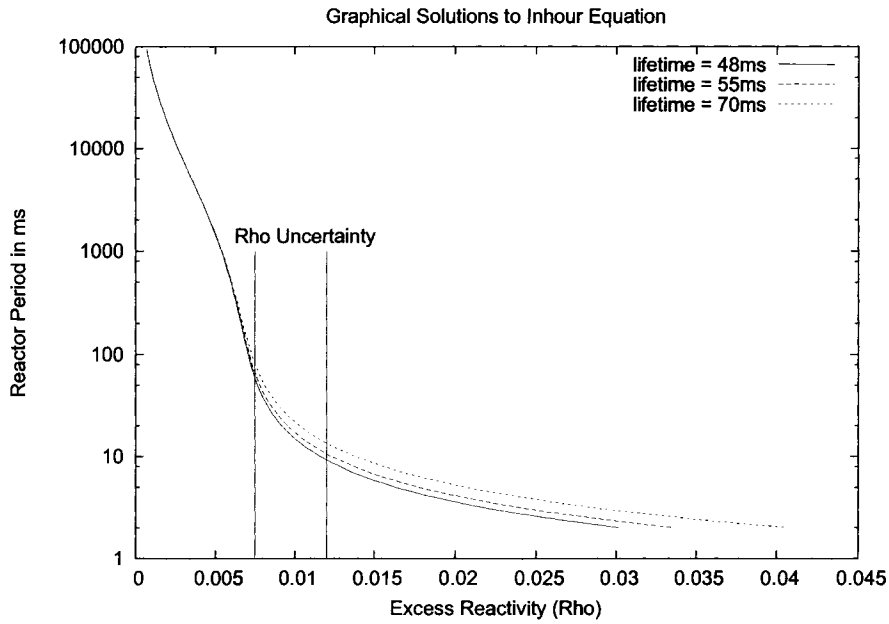


Figure 5.2: Inhour Equation Solution

For short periods the reactivity depends heavily on the neutron lifetime within the reactor. Neutron lifetime is difficult to measure and varies depending on the reactor type. Several estimates of average neutron lifetime were used, an estimate in the range of 48ms to 55ms was recommended by Simon Day. Reactor period is only dependent on neutron lifetime for the case where the period is very short. This adds to the uncertainty in the calibration of the model.

The main purpose of examining the plot in Figure 5.2 is to formulate an estimate of the range of reactor periods which can reasonably be associated with the supercritical core. The transient algorithm is run and *fcal* is adjusted until the reactor period matches the estimate taken from the inhour equation. The control rods are fully withdrawn in this case and the model is allowed to undergo an uncontrolled excursion for the purposes of accurately estimating the period by measuring the increase in

reactor power for each time step. The transient model is executed for a long enough period that the delayed precursors have a sufficient amount of time to stabilize.

The difference between the values computed for *fcal* in the steady state case and the transient case is marginal, but important. Usually a shift of less than 0.05% in k_{eff} is required to correct the transient model. This correction represents the necessary modifications to the equations which are brought into play by the set of delayed precursor constants.

5.1.3 Numerical Simulation Parameters

In any simulation study the goal is typically to achieve an acceptable level of precision with a minimal amount of effort. Effort in this case can be quantified as either the amount of time required for a given simulation to execute, or can be measured as the difficulty of the implementation of the simulation. Some errors may be tolerated, others may not be. Simulation parameters which can be adjusted that have an impact on the execution speed of the model as well as an impact on the precision of the model are:

- The geometric mesh density
- The number of energy groups modeled
- The order of the time integration
- The step size used for time integration

Higher order approximations to spatial derivatives are not implemented by the MOOSE for a variety of reasons, partly because they complicate the inter-mesh connection strategies. The MOOSE uses second order estimates of spatial derivatives.

Simulation times increase dramatically with denser meshes, so only a few different mesh densities are studied. Some solutions are presented with regional mesh refinements. In particular regional refinement is helpful near material discontinuities, and near leading and trailing regions of motion.

The time integration problem is inherently stiff due to the broad range of time constants that must be modeled. The stiffness of the problem suggests that a higher order integration method may be necessary. To minimize stability problems associated with the CFL condition an implicit third order multi-value time integration method is used.

A variety of step sizes were experimented with. Choosing a small step size for the time integration routine provides better precision at the cost of taking more steps. This relationship is investigated in more detail later in this chapter.

In the following sections each of these parameters will be adjusted in the attempt to realize the most precise simulation setup for the problem at hand.

5.1.4 Physical Simulation Parameters

While certain parameters of the fuel insertion problem were measurable, other parameters are not well known. At the McMaster Nuclear reactor fuel assemblies are inserted by hand. An operator stands on the bridge which is suspended above the core and uses a long hook to insert fuel assemblies. The insertion time in the incident report is specified as 20 seconds, however since this is not a mechanically controlled process it may be subject to a certain amount of variation.

The initial flux of the reactor core is extremely low, and is not actually measurable. In principle the sub-critical power of the core can be measured by comparing the difference of the water inlet and outlet temperatures although if the core has been

shut down for a sufficiently long time this difference may be too small to measure with any accuracy. Sub-critical core heat is produced by a variety of processes, and will not necessarily be the result of nuclear processes which produce neutrons. The number of fissioning neutrons present in the core can be estimated by examining the density of certain spontaneously fissioning fuel by-products, in particular Pu240 is relevant. Spontaneous fission rates for two main fission byproducts are

- Pu240 1.5e+3 n/(gram*s)
- U238 .018e+3 n/(grams*s)

A 30% burned up HEU fuel assembly will have a ratio of 238U to 240Pu of approximately 500 to 1 [90], or roughly .016 grams of Pu240 / assembly. Other sub-critical neutron sources also exist including neutrons which result from the collision of gamma rays and heavy water, although in the MNR the proportion of heavy water in the coolant is very small. The MNR also has a beryllium assembly which can be activated with a gamma source to produce neutrons. The sub-critical neutron source is difficult to quantify with any degree of precision given the variety of processes involved and the difficulty in measuring them empirically. Its impact on the transient simulation will be investigated later in this chapter.

The McMaster nuclear reactor has a variety of demands placed on it to maximize flux at various regions within the core for users of radiation sites and beam ports. Consequently the fuel loading patterns are adjusted regularly to try to meet the needs of various researchers. The fuel loading pattern is therefore a complex history problem, and no attempt has been made to represent partial assembly burn-up in this study other than an overall core burn-up of around 25%.

The ion chamber which signals the high power trip incurs a delay between the measurement of high power and the physical release of the shut down rods of about

25ms. The shut down rods are released by cutting off the current to the electric magnets which hold them in place above the core. In addition to delays incurred by the the ion chamber's control circuitry some time is required for the residual magnetism in the coils which support the magnets to dissipate. Since in this experiment the core is in a super-critical state with a very short period at the instant that the core reaches its maximum power even a small variation in the precise value of this delay may have a large impact on the maximum power achieved by the the core.

The power which the ion chamber measures is not precisely proportional to the maximum power of the core during the course of the excursion. Given the presence of the delayed precursors and the various effects of multiple energy groups as part of the transient multi-dimensional study the shape of the flux profile at sub-critical power will not be the same as the shape of the flux profile at maximum power. This suggest that the position of the Ion chamber as modeled in the two dimensional study may have some impact on the accuracy of the study as well.

The ion chamber itself is modeled rather simply. For discussion of radiation detection instruments see [74, 104]. Without delving too deeply into the physics of how such sensors work the assumption is made that the ion chamber generates a DC signal proportionate to the number of ion pairs generated at any given instant from collision with high energy particles. The generation of a single ion pair is understood to be the result of a colliding particle loosing about 30-35 eV. For purposes of simulation an estimate of the total energy of all radioactive particles in a cell is computed by taking an average across each flux group with an average energy level of at least 30 eV, weighted by the average energy of that group.

To satisfy safety regulations the shim rods must be fully inserted within 500ms. This corresponds to about 1/2 the acceleration due to gravity, the reduction in rate is caused by the water in the core.

5.1.5 The Simulation Geometry

The simulation was built on a grid with dimensions 41 x 40 cells, the reactor core used 19x15 cells, the remaining cells representing the moderator. The fuel within the core occupied a physical space of approximately 60 cm in height, 45 cm in length, and 56 cm in width. For the two-dimensional representation a 60 cm height was used with a 48 cm width. This gave a cell dimension of 2.5 cm wide by 4 cm long.

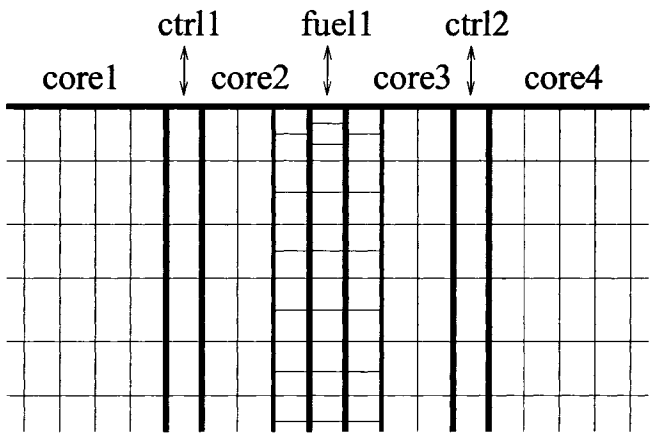


Figure 5.3: Refined Mesh Showing Top of Core

Figure 5.3 shows a portion of the geometry which focuses on the top of the core and labels each of the main regions. The shim rods were positioned so that when *fuel1* was fully inserted they divided the core into 3 roughly equal segments, of width 5 cells for *core1*, 7 cells for *core2* + *fuel1* + *core3*, and 5 cells for *core4*.

The model was run at several different resolutions. In the default resolution the core was represented by 15x19 cells within a simulation region of 41x40 cells. Higher resolutions which were tested include 82x80 cells, 164x160 cells, 246x240 cells and 328x320 cells for the most refined tests.

For the conservation tests several mesh refinements were applied. The regions *core*2 and *core*3 were further subdivided so that the cells immediately to the left and

right of *fuel1* could be refined in the Y direction as shown in Figure 5.3. All the cells in *fuel1* were similarly refined, with the added feature that the tips of *fuel1* were divided into 3 parts instead of just two. The coarsest mesh dimensions are 41x40 cells a total of 1640 geometric positions used by the volume weighted tests. Under the refined mesh strategy they are 38x40+3x80+4 for a total of 1764 cells in the coarsest geometry, an increase of less than 10%.

5.2 Steady State Simulation Results

The steady state simulations presented in this section focus on examining the reactivity changes in the core which result from small adjustments in the position of the fuel rod. These simulation studies effectively compute global rate of change of reactivity in the core which is characterized by the inverse of the first eigenvalue. These simulation studies compute the first derivative of the power curve or the instantaneous neutron multiplication rate of the core which does not consider the delayed precursors and which would result from the tested fuel assembly insertion level.

Several issues are addressed in this section. Although by default the simulation is calibrated at several extreme points there will still be several measurable variations which occur in between those points, it is the object of this section to use these variations to argue for the relative merits of various simulation methods. A two group simulation is used to conduct this part of the study. 4 different mesh refinements are compared, each is twice as dense as its predecessor.

The experiments in the following sections were designed to evaluate the merits of the conservation methods. Results computed with moving conservative meshes are compared with motion modeled using a simple volume weighted technique. Volume weighted methods are often used as a reference point due to their simplicity of im-

plementation. The volume weighted method approximates intermediate rod positions by using cells with averaged diffusion values. This methodology usually demonstrates unsatisfactory results for large mesh spacing, however the method is simple, it makes sense in an intuitive way and it does not require any advanced mesh techniques.

5.2.1 Geometric Refinement Study

The mesh densities, including both the core and the surrounding moderator are

- 41x40
- 82x80
- 164x160
- 328x320

The simulation runs were done using two energy group divisions. For this portion of the study reactivity was only measured for fully aligned cell positions for each of the meshes in question. The most demanding geometry at 328x320 cells consumed close to the total amount of memory on the available hardware (about 1 Gigabyte) so this is the last refinement that was attempted and it was only computed for the reference case where assembly positions are aligned with the mesh. The reactivity curve computed at 328x320 cells is plotted in Figure 5.4. For this plot reactivity increases as the insertion distance approaches zero. The rest of this chapter will use the convention that negative insertion distances mean that the rod, either fuel or control, is withdrawn. A reactivity greater than one indicates that the reactor power is increasing, a reactivity of less than one indicates that the reactor power is decreasing.

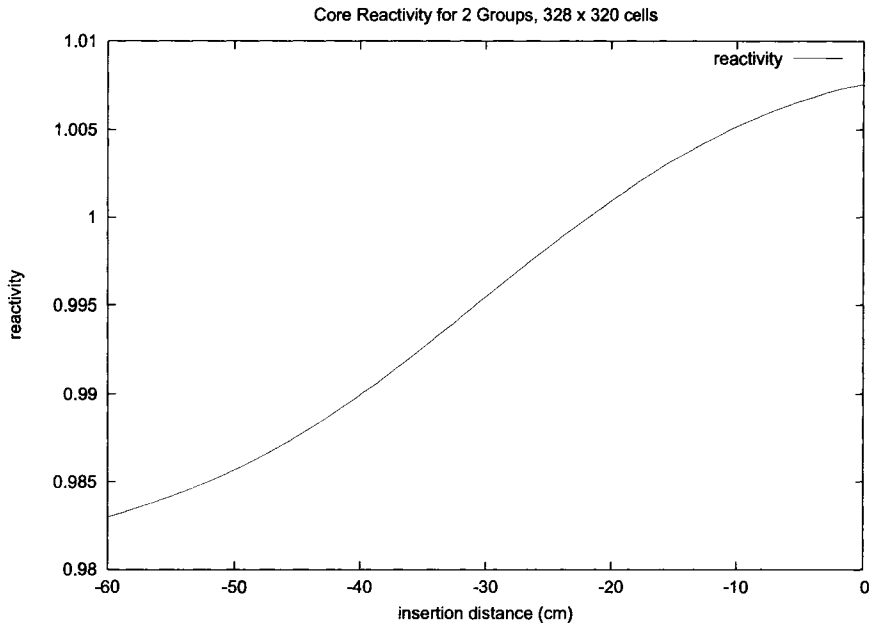


Figure 5.4: Reactivity vs. Insertion Distance

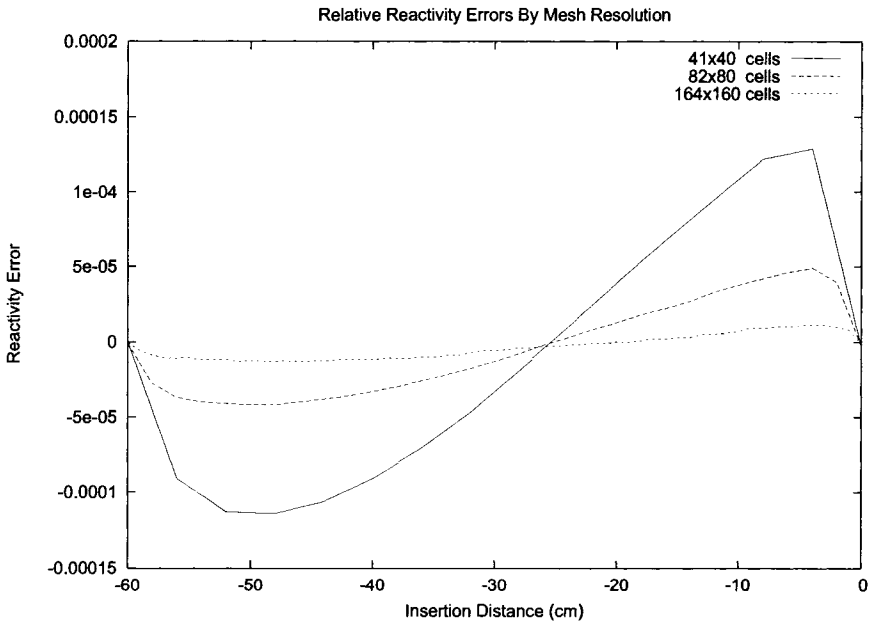


Figure 5.5: Relative Reactivity Errors

Since the reactivity curves for each mesh differ by only very small amounts, rather than plotting the curves themselves, the difference between the curves are plotted. Figure 5.5 shows the difference between reactivity measurements computed at three different resolutions subtracted from the curve computed at the highest resolution. This plot gives an indication of what errors can be attributed to the mesh, and what errors may arise from other sources.

The next sequence of tests compares two techniques for approximating motion of the fuel assembly labeled *fuel1* in Figure 5.3. These tests measure the sequence of instantaneous reactivities for a continuous sequence of fuel assembly positions. The first technique, labeled as *conservation* in the graphs, uses the refined mesh illustrated in Figure 5.3, and is based on the methodology described in Chapter 3. The second technique, labeled *volume weighted* in the graphs, uses a standard Cartesian mesh the simple method of averaging cell constants to approximate fuel assembly positions which cannot be accurately represented on the mesh. Reactivity curves for the coarsest mesh are plotted in Figure 5.6.

Error estimates were computed by taking the difference between the computed solution at arbitrary positions with an interpolated solution, where only mesh aligned calculations were used, and a fourth order interpolation function was used to estimate an ideal solution. This error estimation method makes the assumption that the first derivative of the reactivity curve should be continuous. Any smooth interpolation of trusted points should provide a good estimate of the reactivity curve, although this is not the same as an exact solution since it changes depending on the mesh resolution. The differences for ideal solutions of various resolutions are illustrated in Figure 5.5. Unusual cusping in the curve or sharp irregular changes in the curve's direction are assumed to be the results of numerical errors rather than physical artifacts.

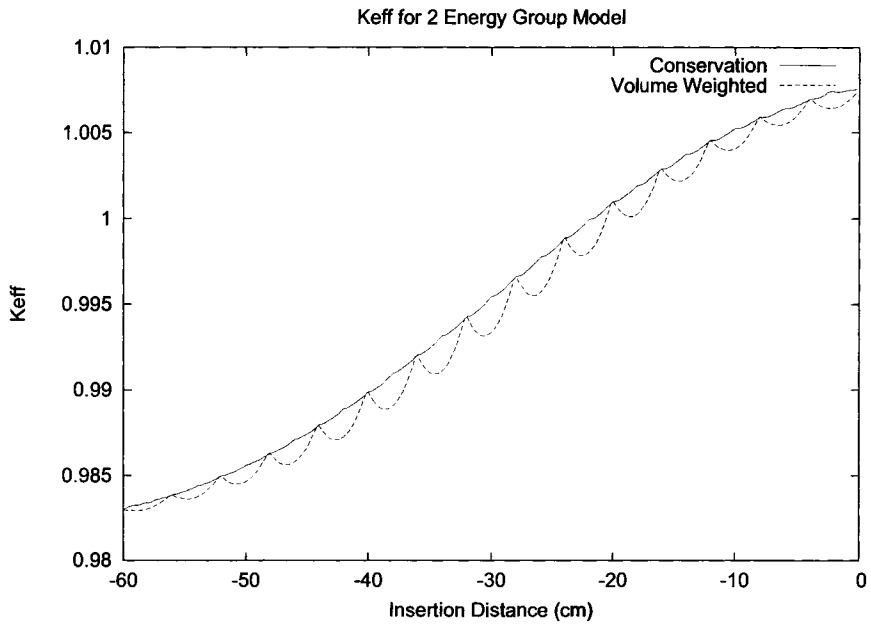


Figure 5.6: Reactivity Cusps

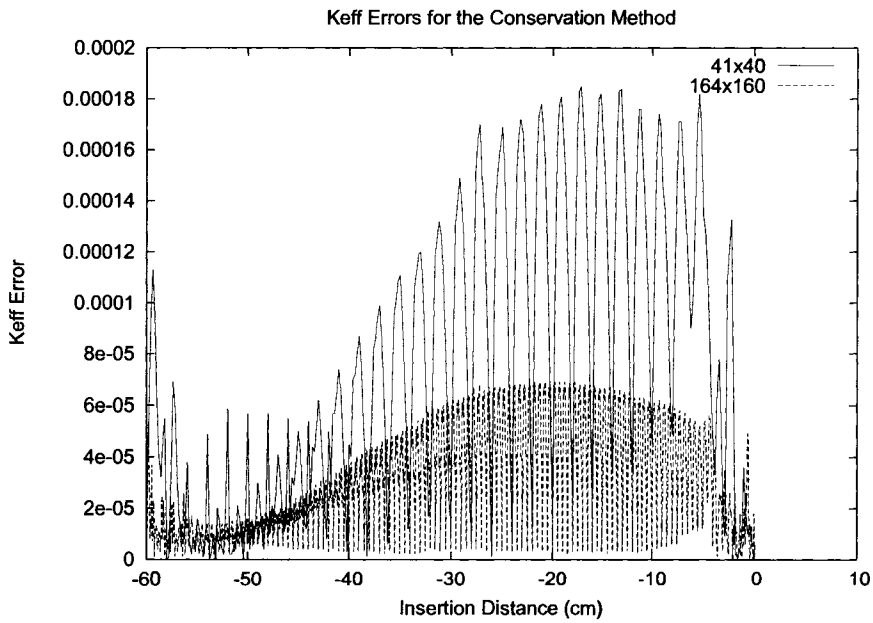


Figure 5.7: Errors for the Conservation Method

The relative reactivity errors for 2 resolutions under the conservation methodology are plotted in Figure 5.7. These errors are calculated as the absolute difference between the ideal reactivity solution and the computed solution. Each cusp represents the motion of the fuel assembly from one cell boundary to the next, so there are correspondingly 4 times as many cusps in the 164x160 resolution plot. Errors tend to be largest at the beginning of the insertion and at the end of the insertion. When the tip of the fuel assembly is closely aligned with either edge of the core this creates a decision conflict for the conservation algorithm since it must deal with material discontinuities on either side of a mesh boundary. As would be expected, with reduced cell size a reduction in error is also observed.

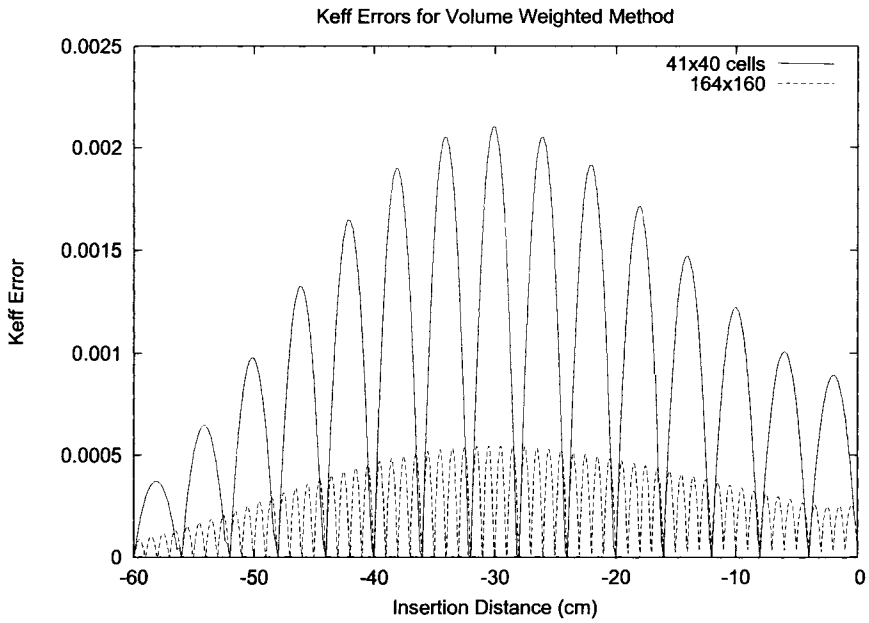


Figure 5.8: Errors for the Volume Weighted Method

The relative reactivity errors for 2 resolutions under the standard cell constant mixture scheme are plotted in Figure 5.8. These errors are computed in the same way as those presented in Figure 5.7. This curve has some features that are similar to those

illustrated in Figure 5.7. As in the conservation plots, each cusp represents the motion of the fuel assembly from one cell boundary to the next, so they are correspondingly 4 times as many cusps in the 164x160 resolution plot. The conservation geometry uses additional localized refinements so few and more regular cusps are evident in the remixed plots.

Both the volume weighted method and the conservation method have difficulty estimating reactivity in the middle of the core. This can be explained by observing that the rate of change of reactivity, as illustrated in Figure 5.4 is greatest when the fuel assembly is 1/2 way inserted. The relative error plots for the volume weighted method tend to be simpler curves with smoother shapes than those generated by the conservation method. This is due to the more obvious implementation of the volume weighted method. Each spike and unusual deviation in the conservation errors usually indicates the application of one of the various rules involved in the methodology.

Errors were measured using a 2 norm computed as

$$\sqrt{\sum_{i}^{max} Abs\left(Ideal_i - Measured_i\right)^2 \cdot \left(x_i - x_{i-1}\right)} \tag{5.6}$$

The results are summarized in Table 5.3. In each case the conservation methods show a significant reduction in error over the volume weighted strategy. Errors for each mesh, as measured with the two norm are reduced by a factor of 10 when volume weighted techniques are compared against conservative moving meshes. While an error reduction of about 1/4 would normally be expected for a mesh doubling study the idealized solution in this case is not the same as an exact solution, and so the principles described in the previous chapter regarding observed orders of accuracy do not apply.

Mesh Density	Computed Points	Conservation Error	Volume Weighted Error
41x40	300	.0557	.80
82x80	600	.030	.44
164x160	1200	.017	.22

Table 5.3: 2 Norm Error Summaries

5.2.2 Energy Group Study

While geometric refinement is understood to have a clear impact on precision it is not as clear whether a more refined energy group structure would also lead to differences or unusual peculiarities for either the conservation methods or the remix methods.

The same experiment which compares the integrated error difference for the conservation reactivity estimation method with the volume weighted method was performed for various energy group divisions. As discussed in the section on approximations the group collapsing procedure does not produce ideal results. A 5%-10% difference in core burn-up was required to calibrate the models which is also reflected in the measurement of 2 norm errors for various energy division. The errors for the 12 energy group model are plotted in Figure 5.9. This plot shows the difference between the errors generated by the volume weighted method versus the errors generated by the conservation method. These errors are typical across all energy groups. The conservation model has twice as many humps as the remix model because it uses a localized geometric refinement although the total number of variables across the mesh is almost the same.

A variety of tests were performed. Increasing the energy group division did not seem to have any impact on the size of the cusps for either the conservation method or the volume weighted method. Simulation models for 2, 4, 8 and 12 energy groups all demonstrated similar error patterns.

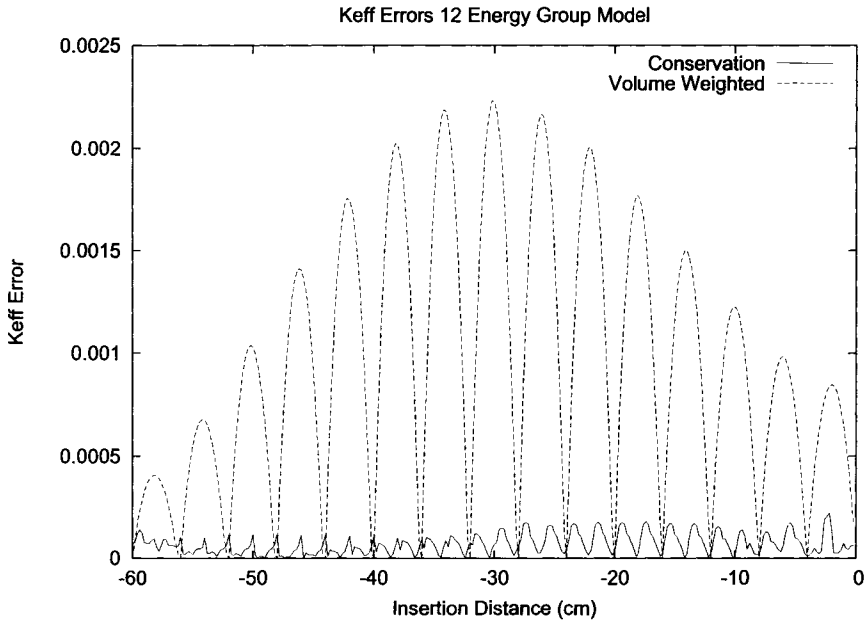


Figure 5.9: 12 Group Reactivity Errors

5.2.3 Performance Study

Strictly speaking the execution speed of the MOOSE frame work is determined by the implementation of its solvers, and as such is not actually at issue for this thesis, some summary performance data for parallel solution times is presented.

Jose Roman, one of the main SLEPc authors who provided assistance in fine tuning the eigensolver, reported the following parallel execution times for the 164x160 mesh and the 320x328 problem under the 8 group case in Table 5.4. For his tests, the following computing platform was used: a cluster of 20 nodes with dual Pentium 2 Ghz Xeon processors with 1 Gbyte of memory per node, interconnected with a high-speed SCI network with 2-D torus topology. A detailed version of these performance results, as well as a summary of the first part of this chapter will appear in an article published in the Annals of Nuclear Engineering [73] sometime this year[4].

[4]Accpeted for publication June 6th, 2007.

Mesh Dim	Row Dim	Matrix Non-Zeros	proc=1 secs	proc=2 secs	proc=4 secs	proc=8 secs	proc=16 secs
164x160	260,064	2,929,008	93.99	57.51	30.79	24.67	15.91
328x320	971,296	11,230,176	502.61	373.56	160.68	85.15	–

Table 5.4: Parallel Execution Times in Seconds for 8 Group Problem

These tests mainly illustrate the parallel scalability of the eigensolver library to 8 and 16 nodes. A full investigation into the performance of numerical solvers and the details behind a parallel solution strategy goes beyond the scope of this thesis. This information is reproduced here only to give a rough indication of SLEPc's capacities.

5.3 Transient Simulation Tests

The steady state simulation tests have a variety of calibration points which allow their values to be fixed at certain extremes. Confidence in the results derived in this section is based on the confidence that can be derived from the steady state simulations. The transient simulation includes several additional terms, not relevant to the steady state model, which represent the delayed precursors. As discussed in the section on calibration, the inhour equation was used to double check the transient model and ensure that the simulated period correctly corresponded with the model's excess reactivity.

This section assumes several default simulation parameters which remain fixed, unless otherwise noted

- uncontrolled minimum reactor period = 25ms
- maximum power trip = 2500 KW, as measured at ion chamber location
- maximum power is taken across entire core, not necessarily same as ion chamber

- subcritical power = 100 mW
- fuel assembly insertion speed = 3cm/sec
- shutdown rods drop at 1/2 acceleration due to gravity
- 50 ms delay between maximum power measured and fuel assembly release
- 4 energy groups
- ion chamber power is measured from the fastest energy group
- power step size ratio = 0.925
- conservative mesh

One of the major difficulties in presenting any transient results which attempt to reproduce the 1994 reactivity insertion incident is that the precise reactivity for the fully inserted fuel assembly is not known. While the previous section discussed what may seem to be very small differences in eigenvalues, this section will make it clear what impact such small differences can have.

This chapter uses a generous breadth in estimating the maximum reactivity that the MNR core could have reached. As a preliminary study the maximum excess reactivity is used as a calibration point for the calculation of the maximum power, as discussed on the section on calibration. The suggested uncertainty in the estimation of the fuel assembly worth of +/- 1.5 mk translates into an uncertainty in the maximum power of about +/- 20%. Given the results in Table 5.5, all models in this section will use an excess reactivity of 9 mk as their calibration point, and will assume an error of +/- 30%. Narrowing this estimate is outside the scope of this thesis and is more properly studied as a physics problem.

Excess Reactivity ρ (mk)	7.5	8.5	9	9.5	10.5
Uncontrolled Minimum Period (ms)	50	30	25	21	15
Maximum Simulated Power (MW)	5.7	7.6	8.7	9.8	12.1

Table 5.5: Maximum Reactivity vs. Maximum Power

Previous studies which estimate the transient behaviour of the core under similar circumstance were computed at a lower resolution using point kinetics models than the estimates computed in this chapter. On the one hand this makes the results presented in this section entirely novel, because no similar results for the MNR have been computed to the same degree of resolution. On the other hand, the results presented in this section cannot be clearly validated against any existing measurements or other comparable simulations.

One of the most commonly cited failures of simulation studies is that they often do not include sufficient evidence of their validity or accuracy. It is believed that the material presented in this chapter which analyzes errors for the steady state case when combined with the more general discussion in the previous chapter on verification provide a sufficient degree of credibility to establish the conclusions drawn at the end of the chapter. The broad error range that this chapter assumes must also be taken into account. It is important to point out that the purpose in studying the reactivity excursion incident is primarily to calculate the order of magnitude of the power maximum. Differences between results of +/- 10% are not important, rather what is at stake is whether the instantaneous power is 2 times, 5 times, or 50 times the acceptable limit for the core. Given that the reactor core's instantaneous power recorder exceeded its scale of 6MW for a brief, but significant period of time, these rough guesses are within the range of possibility.

Figure 5.10 shows the difference between two reactivity curves computed with the MOOSE, one using conservation techniques the other using constants weighted by

volume to represent tip motion. The volume weighted plot shows many unnatural spikes and dips which are clearly the result of numerical artifacts and have no bearing on the actual simulation. The maximum power derived using both methods is similar, although the time at which the peaks occur is different. The maximum power is largely determined by the segment of the curve which immediately follows the 2.5MW point, and so in some respects this metric forgives the volume weighted method for its earlier mistakes. The dropping of the shut down rods brings a rather abrupt halt to the excursion.

Time (s)	8	12	13	14	15	16	17
Volume Weighted Power (W)	.27	.53	.73	1.2	3.9	24	520
Conservative Power (W)	.28	.59	1.0	2.2	7.8	120	29000

Table 5.6: Selected Power Levels Prior to Control Rod Drop

Table 5.6 compares the computed power levels shortly before the control rods drop. At 8 seconds the two methods approximately agree, differing by less than 5% in their estimate of the power level. At 17 seconds the two methods differ in their estimate of maximum power by a factor of 50. This sudden change illustrates the impact that an error of 1 or 2 mk can have on a reactivity calculation and is the principle argument in favour of using the conservative mesh strategy.

The remainder of this chapter will focus on simulation results derived using the MOOSE's conservation methodology on a coarse 41x40 mesh. The volume weighted strategy will not be investigated any further.

One of the main concerns addressed by the original tech report which discussed the refueling incident was whether or not the reactor cladding actually melted. For the cladding to melt the core power needed to reach a peek high enough for long enough to raise the temperature of the fuel cladding to a value above the melting

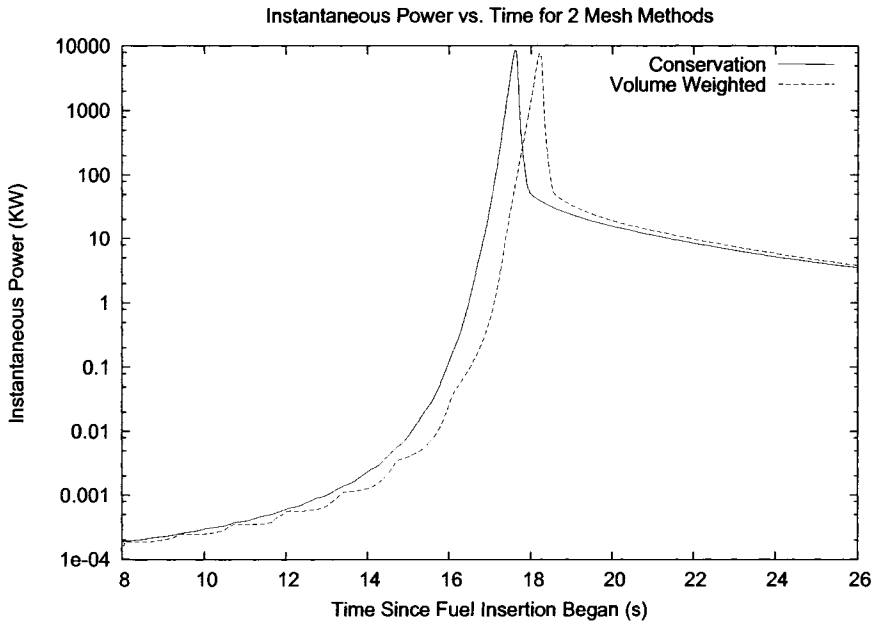


Figure 5.10: Differences Between Conservation and Volume Weighted Methods

point for aluminum, which is around 650 degrees Celsius. The tests performed in this section investigate the maximum instantaneous power reached by the reactor at two locations. The first location is that measured by the ion chamber. The second location is roughly near the center of the core and is a measure of the maximum power reached. Previous point kinetic simulations were not able to make this kind of spatial distinction in the location of various power measurements in the MNR core. This investigation will not attempt to model the temperature of the fuel plates.

While more in depth models are within the capability of the MOOSE's framework, the scope of this thesis cannot include all possible avenues of investigation. The analysis of the reactivity power as an instantaneous function of the time dependent neutron diffusion equation combined with the delayed precursors presents a sufficiently rich avenue for investigation, and this section will limit its investigation to the study of a handful of parameters which influence the core's power peak.

Parameters which will effect the model setup include the step size ratio, the ion chamber delay, and the fuel assembly insertion time.

5.3.1 Time Based Integration Method Selection

At the beginning of any given transient run the exact point at which the power peak occurs cannot be known, finding this point is rather the purpose of the simulation. The integration algorithm must support a variable step size to capture the rapidly changing behaviour which occurs at the instant that the power reaches its maximum. While multi-step predictor corrector algorithms were experimented with, the inability of these methods to easily adjust their step size made them unusable. The transient neutron diffusion problem must represent neutrons which travel at a wide variety of velocities. As discussed in chapter 2 this results in an extremely stiff problem. Second order trapezoid methods were experimented with, and although they were reasonably accurate and simple to program they often failed to remain stable throughout the duration of the simulated run. Modeling the wide variety of neutron speeds while still being able to take steps large enough to allow the simulation to progress at a reasonable rate requires the use of multi-value methods. Multi-value methods are very accurate, allow their step size to be changed during the course of execution, and for some variations also have very good stability properties . All of the transient plots in this chapter were computed with a 3rd order multi-value integration method.

5.3.2 Step Size

The choice of step size is related to the choice of integration methods. More precise methods can often use larger step sizes, although the integration step size may change as the simulation progresses.

In the region of the fuel reactivity peak the neutron diffusion power function behaves very much like an exponential curve. The rate at which the curve increases is approximated by the ratio of consecutive power measurements taken at the ion chamber. The transient algorithm chooses its step size by attempting to keep this ratio constant. At each iteration of the algorithm a step is tentatively taken. If the ratio of the new power level to the old power level is less than the accepted threshold, the new calculation is kept, and the step size is increased. If the new step results in a power ratio which is outside the accepted range the step is then rejected, the step size is reduced, and a new step is calculated. Many steps in sequence may be rejected while the algorithm searches for an appropriately small step to take.

To avoid changing step size at every single stage in the algorithm a small range is maintained between the lower and upper acceptable step ratios. Additional step size constraints are imposed when the reactor power gets very close to 2500KW. If the simulation measures a core power greater than 2500KW it checks the size of the previous measurement. If the previous measurement was less than 2490KW the current step is rejected and the step size is reduced. This ensures that no matter how large a step the algorithm takes it will choose a time index appropriately close to the actual instant that the core goes overpower, usually within about +/- 5KW of 2500KW.

A simple metric, step size ratio, measured the change in the total power. The step size ratio was measured as the

$$\frac{min(Power_{old}, Power_{new})}{max(Power_{old}, Power_{new})}. \tag{5.7}$$

A sequence of two group transient simulations were run which used a variety of ratios between the step sizes ranging from 0.5 to 0.95. An ideal step size should be large

enough to minimize execution time, but small enough to preserve a reasonable amount of accuracy. The measured power maximum for each step size is reported in Table 5.7. The power excursion for two step sizes is plotted in Figure 5.11 on a logarithmic scale for the period immediately before and after the peak power was reached. The 0.70 step is plotted with individual points to give an indication of how the step size was adjusted.

Step Size Ratio	.5	.7	.80	.85	.875	.90	.925	.94	.95
Power Peak (MW)	8.90	9.13	8.80	8.60	8.77	8.69	8.66	8.61	8.57
Steps taken	140	195	268	337	398	483	628	754	924

Table 5.7: Transient Step Size Selection

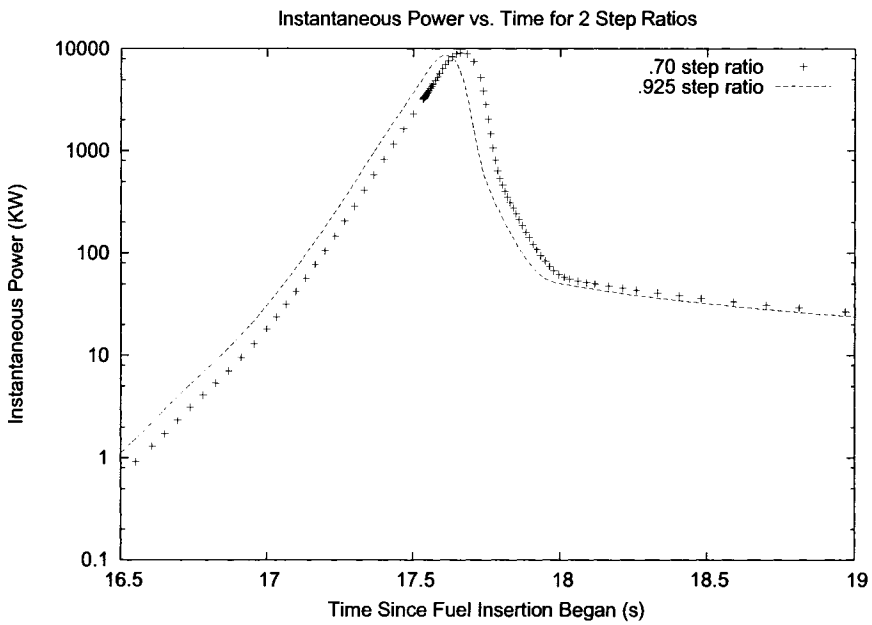


Figure 5.11: Transient Reactivity Excursions

Neither curve plotted in Figure 5.11 is perfectly smooth. The small perturbations in the plot are easily attributed to the approximate nature of the reactivity estimations

between grid points. The plot generated based on the .70 step ratio separates the points sufficiently that it provides a sense of how the algorithm adjusts step size. By inspecting this plot around time index 17.5 seconds the point at which the step size is reduced to capture the instant that the power goes above 2500KW can be observed.

Changing the step size has some impact on precisely which points are used during the excursion. One would expect that by increasing the number of steps taken by the algorithm the power peak might shift in a consistent way either with an increasing trend or with a decreasing trend. The data collected in Table 5.7 tends to indicate that using a step size ratio of 0.925 gives a result accurate within 1% of the estimate based on the .95 ratio which uses 50% more points. Based on this data the remaining tests in this chapter will use a step size ratio of 0.925.

5.3.3 Number of Energy Groups

Using the step size estimated in the previous section three separate simulation runs using various energy group divisions measured the peak power for the core. The 2 group model was not used for transient tests because for very short reactor periods it exhibited behaviour quite different from the other three models. In addition to the peak power the measurement recorded at the ion chamber as well as the time of the peak are also recorded. Increasing the number of energy group divisions used by the problem tends to change the shape that the flux takes. All the ion chamber measurements are taken from the first group in each case. The first group measures the neutron flux with energies that range from 8.21e+5 eV to 1e+7 eV and is the same for each simulation. Although no appreciable differences were noticed between the behaviour of simulations with different number of energy divisions for the steady state case, for the transient case the flux shape of the highest group is slightly different

for each model. The 4 group or 8 group models are both good candidates for more precise measurement studies given their reasonable running times. The 12 group model executes quite slowly relative to the others and while it may present information with the highest accuracy the long execution times required make it desirable to limit further study to the 4 and 8 group models.

Energy Groups	4	8	12
Peak Power (Kw)	8660	8660	8650
Ion Chamber Peak (Kw)	6600	7080	7200
Time of Peak	17.6	16.8	17.0

Table 5.8: Peak Power by Energy Division

5.3.4 Ion Chamber Delay

The time delay between when the particles in the ion chamber are activated by high levels of radiation and the time when the shim rods are released is reported to be 25ms in the MNR safety report. Given that the reactor period is so short at the instant that this occurs there is some concern that if this delay were very much longer that the reactor might reach a much higher peak power. This value is difficult to measure and it could in fact be smaller or larger than the reported value. The results in Table 5.9 show that for small variations around the estimated value of 25ms no more than a 10% shift in the peak power takes place. Very long ion chamber delays, on the order of several times the reactor's minimum period are required before the Peak Power changes in a drastic way. 50 ms is a relatively conservative estimate of the ion chamber delay. It is likely closer to 25 ms.

The model presented in this chapter is able to compute the difference in power measured at the location of the ion chamber and the absolute maximum of the core. In steady state these two values agree, but during the excursion the shape of the

core changes slightly and the flux at the center of the core increases at a faster rate than the flux at the edge of the core. This phenomena is marginal, the difference in the shape of the core only accounts for the interior of the core being about 25%-30% higher in power than the power measured at the ion chamber.

Delay (ms)	15	20	25	30	40	50	60	75	100	150
Peak (MW)	5.9	6.2	6.4	6.8	7.6	8.7	9.3	11.0	14.8	27.3

Table 5.9: Peak Power vs. Ion Chamber Delay

5.3.5 Sub-critical Power

As already discussed the sub-critical flux is difficult to estimate and impossible to measure, especially if it is assumed that the core has not been operational for a week or more. Since the transient neutron diffusion problem exhibits exponential behaviours it is reasonable to ask what impact the initial point has on the performance of the problem. Given the already established parameters a collection of 4 group simulation models were run each using a different sub-critical power in the range of 1 watt and 1e-6 watts. Reductions in the starting power effectively allows the fuel assembly to be inserted further into the core before the 2.5 MW trip point is reached. Extra insertion distance means that some additional reactivity has been added by the time that the core is at the 2.5 MW point. The simulation is somewhat insensitive to this factor, it takes a reduction of a factor of 10^6 in the initial core power to increase the maximum core power by a factor of 2.

Sub-critical Power (W)	1	1e-1	1e-2	1e-3	1e-4	1e-5	1e-6
Peak Power (MW)	8.2	8.6	10.6	12.5	12.7	14.4	17.7
Time Peak Occurs	17.37	17.61	17.82	18.01	18.18	18.35	18.51

Table 5.10: Peak Power vs. Sub-critical Power

5.3.6 Rod Insertion Speed

Fuel assemblies are inserted by an operator who stands on the bridge over the core at the MNR. The process is not automated in any way but is left to the discretion and experience of the operator. The original estimate of a 20 second insertion time is only a best guess which represents the typical actions of an operator. A range of insertion speeds from 10cm/s (assuming that the assembly was accidentally dropped at some point) to .5 cm/s is used . The differences between these trials are listed in Table 5.11. Since the actual insertion is done by hand the speed will not necessarily be uniform. The insertion speed of 3 cm/s, corresponding with a total insertion time of 20 seconds, is suggested as likely the fastest rate of insertion that would have occurred. The simulation is relatively sensitive to insertion speeds, reducing the speed of the fuel assembly insertion tends to reduce the maximum power achieved.

Velocity (cm/s)	10	7.5	5	3	2	1	.5
Peak (MW)	37.5	30.5	15.2	8.7	6.7	4.5	3.7
Time (s)	5.98	7.7	11.0	17.7	25.6	48.98	93.7

Table 5.11: Peak Power vs. Fuel Insertion Speed

5.4 Comparison with Garland's Report

The original technical note [69] which analyzed the instantaneous power level of the MNR was written by Wm. J. Garland in 1997. Garland's report describes a zero dimensional point kinetics model based on the same parameters described in this chapter. He concluded that the best estimate of peak power was approximately 8.4 MW. In another internal document he reports the results of a sensitivity study and adjusts a variety of parameters similar to the parameters examined in this chapter

and concludes that 9.8MW is a reasonable maximum power. He used a linear ramp to model the reactivity insertion. Garland's report also estimates the highest temperatures reached by the core. The transient simulation studies in this chapter have restricted their focus to the instantaneous power level reached by the core and have not attempted to estimate a maximum temperature.

This chapter has drawn approximately the same conclusion as Garland's report, that the power excursion that the MNR core underwent was well within safety tolerance. The difference in the various peak powers computed in this chapter are not significantly different from the peak computed in Garland's report.

The experiments undertaken in this chapter show that the shape of the core flux only has a moderate impact on the simulation results. There is only a 25% difference between the power measured at the ion chamber and the power measured at the center of the reactor core. This spatial difference is due to a small change in shape of the power profile during the course of the reactivity excursion. This spatial effect is not accounted for in Garland's point kinetics model although it is not significant enough to have a large impact on the maximum power estimate. Garland's report used a linear ramp to model reactivity versus insertion distance. This chapter used a full two dimensional simulation to compute reactivity, a definite S shaped curve was presented in the first section. While this change in shape has some impact on the results, it is not a large enough to clearly account for a major difference.

The most important issue with respect to estimating the maximum height of the instantaneous power during the excursion appears to be tied to the precise amount of excess reactivity that was inserted into the core. Other questions of importance include the estimation of the initial power level, and the ion chamber delay. The impact of these secondary parameters are directly linked to the estimate of excess reactivity inserted into the core.

Issues not considered in the original technical report but relevant to the development of this model include the mesh density used, the energy group structure used, and the intermediate cell approximation method used. Each had a definitely measurable impact on the precision and credibility of the result derived. In addition this study examined certain physical quantities which were considered to be unknowns, including the fuel assembly insertion time, the sub-critical power. The final estimated peak power of about between 8 MW and 12 MW, depending on various parameters, roughly agrees with the figure computed in the 1997 technical report and other estimates reported by Garland.

Interestingly the comparison of the differently shaped reactivity versus insertion curves for the volume weighted case and the conservation case showed that the most important part of this curve in determining the maximum power is its shape in the .5 second period immediately after the core power exceeds 2.5 MW. A simulation which estimates roughly the correct period for this segment of the reactivity curve should be in close agreement with these results. Because of the nature of the problem, whereby the reactivity insertion begins at a time determined when the power reaches 2.5 MW, simulators which use incorrect reactivity histories prior to this time are forgiven for their mistakes.

The specific values computed in Table 5.6 show that slight errors in reactivity curves can result in power errors of enormous magnitude, especially when those errors are allowed to accumulate over even moderately short periods of time. Conservative mesh techniques allow these errors to be avoided while still permitting reasonable execution times; this is the principle argument for their use.

Chapter 6

Conclusions

The MOOSE framework is intended to be a generic modeling tool for simulating moving components on structured grids. As a tool, the MOOSE provides a rich environment which allows a wide variety of problem configurations for the investigation of eigenvalue steady state models and transient models, supporting various design configurations and model layouts. Finite difference and finite volume techniques are straightforward enough that automating the translation of fundamental partial differential operators is possible. The structured meshes used by the MOOSE provide advantages in terms of low memory consumption and analytical simplicity.

The MOOSE framework has also been used to develop moving mesh interconnection strategies appropriate for studying reactor transients. This chapter suggests areas which can take advantage of the framework as well as some general conclusions regarding the advantages which can be achieved using the MOOSE framework. The contributions of this thesis, as summarized in the introduction, are reproduced here.

1) A clearly defined methodology for the linking of meshes

Chapter 3 summarizes the implementation of three rules which facilitate the linking of patched meshes. The rules apply when i) meshes are misaligned, ii) meshes use different densities and iii) when material discontinuities occur along mesh boundaries. These rules were justified through experiments and arguments based on first principles.

2) Detailed error analysis which address two major questions:

2 a) The extent to which using coarse meshes with special motion techniques can improve upon performance

While a full comparison with a nodal code fell outside the scope of this investigation, results computed in Chapter 4 and Chapter 5 showed significant reductions in errors which resulted from calculations performed on coarse mixed density meshes, over similar calculations on meshes which were several times more refined. Chapter 5 quantified some error reductions as approximately a full order of magnitude or more for a given problem.

2 b) Whether interpolation is sufficient to connect meshes

Only in a few special cases should conservation of flow at mesh boundaries be ignored. Nonlinear interpolation alone is not an appropriate mesh connection technique, especially if meshes are linked in a patchwork, or non-overlapping pattern, or if material discontinuities appear near mesh boundaries.

3) Detailed re-examination of the estimated power peak reported in the 1997 MNR technical report

Several experiments originally done with zero dimensional point kinetics models were repeated in Chapter 5 with a multi-dimensional moving mesh model. Although it was suggested in the past that the MNR has a small enough reactor core that

a spatial treatment would not reveal any specific new details about the reactivity incident, this was not demonstrated until now. The report's conclusions were verified to within the uncertainty of the problem parameters.

4) A prototype implementation of the MOOSE framework clearly identifying a variety of design issues

Moving meshes have not been used in reactor physics in part because of the perception that they are difficult to program, difficult to implement, and tend to be inflexible. The flexibility of an implementation can be built into the solution process using tools like computer algebra and code generation as presented in Chapter 3. Such tools, illustrated by the MOOSE's design, can build a bridge between a user configurable model and high performance solution techniques.

5) The first highly developed nuclear application based on the Krylov-Schur method implemented within the SLEPc project

Previous publications by the SLEPc authors [82] used a simplified one and a half speed steady state reactor model. The steady state results discussed in Chapter 5 used 8 and 12 group fully developed fission model combined with a moving mesh and are more sophisticated than the previously reported nuclear models built with this tool. SLEPc is perhaps the only well developed high performance parallel public domain sparse eigenvalue solver available today.

6.1 Future Work

The following sections present unexplored reactivity models, moving mesh topics which go beyond reactor physics, and finally a short summary of logical extensions to the MOOSE framework are summarized.

6.1.1 MNR Models

Future work possible with the existing MOOSE framework may include the analysis of additional detailed simulations of the McMaster Nuclear Reactor. A fundamental correction to the model would be the development of more rigorous cross section data. While the data that was used was correctable through fine tuning of model parameters, a production simulation used to predict new behaviours rather than analyze past scenarios, would require better foundations.

Because the MOOSE framework is able to move an entire sub-mesh relative to the main geometry a sequence of studies which examined the insertion of fuel rods which exhibited non-uniform axial burn-up patterns is possible. Most transient studies assume a uniform axial burn-up. It would be possible to re-conduct the experiments from Chapter 5 using cross sectional data which better modeled burn-up pattern of the rods which typically occurs.

Simulations which tested the sensitivity of the two dimensional model to the position and exact behaviour of the ion chamber are also important. Additional scoping studies which further calibrated the modeled instruments with the actual instruments of the MNR would have been instructive.

The MOOSE is fully capable of modeling both the advective flow of fluid through the core simultaneously with modeling neutrons produced within the core. The instantaneous power is of interest to this thesis mainly in that it served as a challenging

case study which exercised the various capabilities of the MOOSE framework. Modeling heat build up is of interest for the safety analysis of the core since the question of whether the fuel was actually damaged during the reactivity excursion is determined by examining whether the melting point of the aluminum fuel cladding was exceeded.

6.1.2 MEMS: Further Avenues of Investigation

Figure 6.1: Mite and Gear Chain

The case study presented in Chapter 5 focused on reactor control rod motion, however, this is only one possible major application to which the MOOSE framework can be applied. MEMS devices are one example of a modeling domain which incorporates motion. Other examples outside reactor core design include, combustion engine design, robotics design and aerospace design to name only a few. Rather than attempt to discuss a wide range of unrelated examples, this section provides a collection of problem designs which incorporate motion for which there is currently no accepted

PSE tool, and for which the MOOSE framework would be appropriate in its current state of development.

Microelectromechanical Systems (MEMS)[1] are devices which range in size from a micrometer to a millimeter and diverge from standard silicon manufacturing by incorporating moving components. Simple devices such as oscillating capacitors, fluid valves, magnetic springs, optical switches, relays, fracture and motion sensors can be constructed on scales smaller than one millimeter. The design of a MEMS device is subject to computing the simple motion of a component within either an electric or electromagnetic field, and the analysis of the thermal and or mechanical properties of that device. Future work in the area of PSE tools which focus on motion will find MEMS to be a fruitful area for problem definitions.

Examples of MEMS Devices

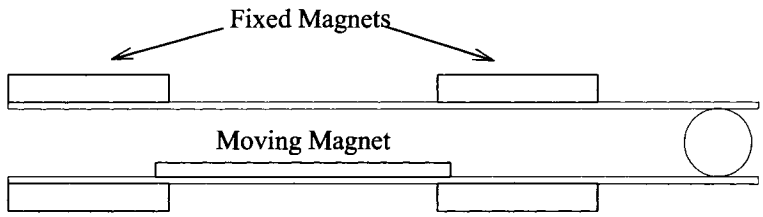


Figure 6.2: Micro Actuator Construction

Rostaing [139] describes a micro actuator which uses permanent magnets to hold a switch in place once it has changed position in a high displacement design (in excess of 100 micrometers), shown in Figure 6.2. A moving magnet is maintained in one of two stable positions by integrated permanent magnets. A pulsed current in the conductors acts on the moving magnet displacing it from one stable position to the

[1]Figure 6.1 Courtesy of Sandia National Laboratories, SUMMiT Technologies, www.mems.sandia.gov

other. During movement, the mobile magnet is neither guided nor retained by any mechanical element. Its central position in its stable states is ensured by the forces from the fixed magnets, angular stability during motion is ensured by inertia.

Interactions between magnets and currents are among the most efficient at small scales. In this application the actuator is driven by current pulses of up to 5 Amps in a conductor of dimensions 20 x 10 micrometers in its thinnest sections. This corresponds to current densities of about 25000 Amps/mm^2 and is possible due to the peculiarities of the scale of the device.

Kawano [100] reports on numerical techniques used to model a two dimensional MEMS variable capacitor with accelerated motion effects. The acceleration of the capacitor is derived under the equilibrium between the mechanical elastic force of the spring and the electrical potential as illustrated in Figure 6.3.

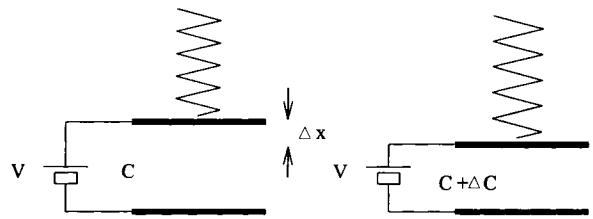


Figure 6.3: Variable MEMS Capacitor

Simulating such a device requires a moving model which can accommodate mechanical, electrical, and electromagnetic phenomena. Not only are the capacitor plates very small, and hence their mass is very low (less than 10^{-10}Kg), but also the system must be able to model very high frequencies.

Other MEMS scale devices include Micromirrors [178] which are often used for implementing optical cross-connects. Micro-valves [91] can be fabricated using hydrogel material. Hydrogel material has the unique characteristic of responding to external stimulus by changing in volume. A micromechanical sensor array of laterally mov-

ing mass-spring systems is described by [145]. The device is fabricated by SCREAM technology and is intended for low cost applications in wear state recognition. Diaphragm membranes are used in a variety of applications in acoustics and are also found in other pressure sensitive devices (such as disposable blood pressure sensors) are described by [54, 114].

6.1.3 Framework Extensions

The majority of the simulations presented in this thesis were either elliptic or parabolic in nature. Hyperbolic problems require a few additional operators (cross product for example) for general implementation which were not developed for the MOOSE framework. In principle these operators could be added as extensions to the already existing matrix generation functions.

Many models require three dimensional representations. This work was not attempted as part of this thesis because it was believed that two dimensional models would be sufficiently able to demonstration the basic conservation principles.

Allowing mesh shear and rotation transformations on moving mesh blocks would provide other important extensions which would increase the scope of geometric models that the MOOSE framework could address. Rotation and shear functions further complicate cell linking and were not necessary for the target applications.

To be generally usable a PSE requires a detailed graphical user interface. While a simplified user interface was developed for the framework, a fully graphical implementation which included context sensitive hints system, parallel debugging tools, graphical mouse driven layout tools, and integrated output representation, is an important and necessary extension.

The error analysis presented in this thesis is based entirely on experimental mea-

surements. Tests described in Chapter 4 and Chapter 5 suggested that the errors induced by the resolution of the mesh were no greater than those which resulted from the conservation mesh linking rules. Analytical work conducted by various authors on linked meshes, discussed in Chapter 2, developed error bounds for mesh linking strategies based on hyperbolic equations for both interpolation connection schemes and for conservation connection schemes. Developing expressions for the error which is induced by the mesh interconnection strategies presented in this thesis would be a productive extension to the work.

6.2 Performance Discussion

The most refined mesh presented in Chapter 5 of 320x328 cells simulated a total of over 200000 variables for the 2 group fuel rod insertion model. On a modern desktop computer the memory allocation procedures used by the direct solver as part of the eigenvalue problem consumed over 1 gigabyte of physical memory to solve this problem. The more memory intensive direct solvers were favoured for their better execution performance and higher precision, especially under the steady state tests. The transient tests did not benefit as much from the precision provided by the direct methods and so the GMRES solver was favoured for some of the transient tests.

A direct dense matrix routine, like those implemented in LAPACK, would require the storage of the matrix as an array of 200000 rows by 200000 columns to represent the 320x328 mesh for the 2 group case. A double precision floating point number consumes 8 bytes of space, so such a matrix is stored in a dense format would consume 320 gigabytes of memory. While sparse solvers present their difficulties, for certain problems, they are capable of circumventing these memory requirement by assuming that unless otherwise specified a matrix entry is zero.

While classical relaxation methods can be implemented without translating the problem into a matrix, if implemented in a naive fashion their performance scales with the cube of the size of the problem. This poor performance alone would make them unusable as tools for this thesis. Previous simulation work [71] developed for the MNR used a simple hand coded iterative solver based on the power method which performed quite poorly both in terms of precision and speed when compared with SLEPc's performance. Earlier versions of this work [72] also suffered from poor precision prior to the incorporation of SLEPc. Plots like those presented in Chapter 5 require the computation of thousands of individual eigenvalues and solver performance, while not the focus of this thesis, will determine whether such computations are possible or not.

The MOOSE's error correcting algorithms yield a difference in precision of a full order of magnitude over naive volume weighted methods. This difference means roughly that the calculation performed on the 41x40 mesh using the conservation method is more precise than the calculation performed using the volume weighted method on a mesh of 164x160 cells. Depending on the eigenvalue solver method selected and the architecture used, at best, the eigenvalue method will scale at about the ratio of $n \cdot \log(n)$ where n is the size of the problem. In the performance studies presented in chapter 5, when timed on a wall clock, the 164x160 mesh calculation ran between 50 and 100 times more slowly than the 41x40 calculation depending on the precise problem setup and computer used.

This simple comparison should compellingly suggest that in the case of motion studies being able to use a coarse grid will dramatically reduce execution times without introducing unmanageable errors.

6.3 Final Conclusion

Although this thesis has focused discussing performance in terms of reducing measurable errors, it must also be pointed out that the reduction in implementation effort for each model built under a framework like the MOOSE is significant. Quantitatively measuring the human effort required to implement one type of model versus another is very difficult and falls squarely within the realm of learning theory, psychology and human computer interfaces. Reducing the implementation effort required by scientists and engineers is just as important as reducing the computational effort required by a machine.

As discussed in the previous section, a one order of magnitude difference in precision between two methods translates into a two orders of difference in magnitude of execution time when the attempt is made to improve the precision of the poorer method by refining the mesh. While the conservation mesh methods are more complex to implement than their more obvious brute force counterparts, the savings in terms of costs of equipment, and the time required to wait for a given solution to be derived may be well worth the effort of implementation.

Increasing global problem resolution to reduce errors should be a last resort. Methods like the grid conservation techniques described in this thesis open up an entirely new vista of computational data. While moving grid techniques have received little attention in nuclear forums, the results presented in this thesis suggest that there is a wealth of high precision information available that could improve the state of the art in nuclear engineering, especially when special problems which take component motion are to be addressed.

While framework development may be expensive, and for any modeling system a certain learning curve is necessary, ultimately tools which follow the ideals presented

by the advocates of PSEs will likely dominate the future simulation landscape. Tools like the MOOSE framework, which can potentially be applied to a wide variety of problem domains, will in the future allow researchers who are interested in modeling new engineering mechanisms to focus their efforts on their particular domains of interest rather than on the details of numerical modeling.

Bibliography

[1] B. Afeyan, K. Won, and A. Kanaev. Modeling and design of optical semiconductor devices using FEMLAB, wavelets and adaptivity. In *Proceedings of the IEEE/LEOS 3rd International Conference on Numerical Simulation of Semiconductor Optoelectronic Devices*, pages 63–67. IEEE Press, October 2003.

[2] Lance J. Agee. RETRAN overview. *Nuclear Technology*, 70:7–16, July 1985.

[3] Agostinelli. Geant4 — a simulation toolkit. *Nuclear Instruments and Methods in Physics Research*, A(506):250–303, 2003.

[4] E.H.K. Akaho and B.T. Maakuu. Simulation of reactivity transients in a miniature neutron source reactor. *Nuclear Engineering and Design*, 213:31–42, 2002.

[5] Robert L. Akers, Elaine Kant, Curtis J. Randall, Stanley Steinberg, and Robert L. Young. SciNapse: A problem solving environment for partial differential equations. *IEEE Computational Science and Engineering*, 4:32–42, 1997.

[6] Robert L. Akers, Elaine Kant, Curtis J. Randall, Stanly Steinberg, and Rober L. Young. *SciNapse: A Problem Solving Environment for Partial Differential Equations*, pages 109–121. In Houstis et al. [89], 2000.

[7] A.V. Alexeyev, O.A. Zvenigorodskaya, R.M. Shagaliev, and T.A. Taiwo. Performance assessment of KORAT-3D on the ANL IBM-SP computer. *Transactions of the America Nuclear Society*, 80:297–298, 1999.

[8] G. Allen, T. Goodale, and E. Seidel. The cactus computational collaboratory: Enabling technologies for relativistic astrophysics, and a toolkit for solving PDEs by communities in science and engineering. In *Frontiers of Massively Parallel Computation*, pages 36–41. IEEE Press, 1999.

[9] Gabrielle Allen, Tome Goodale, G. Lanfermann, T. Radke, E. Seidel, W. Benger, H. Hege, A. Merzky, J. Masso, and J. Shalf. Solving Einstein's equations on supercomputers. *IEEE Computer*, pages 52–58, December 1999.

[10] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer Methods in Applied Mechanics and Engineering*, 184(2–4):501–520, April 2000.

[11] K. Appert, R. Gruber, S. Merazzi, T. M. Tran, and S. Wuthrich. Problem-solving environment for parallel computers. *Nuclear Instruments and Methods in Physics Research A*, 389:51–55, 1997.

[12] L. Archambault, L. Beaulieu, J.F. Carrier, F. Castrovillari, S. Chauvie, F. Foppiano, G. Ghiso, S. Guatelli, S. Incerti, E. Lamanna, S. Larsson, M.C. Lopes, L. Peralta, M.G Pia, P. Rodrigues, V.H. Tremblay, and A. Trindade. Overview of geant4 applications in medical physics. In *IEEE Nuclear Science Symposium Conference Record*, volume 3, pages 1743–1745, October 2003.

[13] Dorian Arnold, Henrib Casanova, and Jack Dongarra. Inovations of the NetSolve grid computing system. *Concurrency and Computation: Practice and Experience*, 14:1457–1479, December 2002.

[14] Joachim K. Axmann. Parallel adaptive evolutionary algorithms for pressurized water reactor reload pattern optimizations. *Nuclear Technology*, 119:276–291, 1997.

[15] Yousry Y. Azmy. Multiprocessing for neutron diffusion and deterministic transport methods. *Progress in Nuclear Energy*, 31(3):317–368, 1997.

[16] Aurora Badulescu and Robert Lyon. Classroom simulators: User friendly education with nuclear reactor simulators. *IAEA Bulletin*, 43(1):25–28, 2001.

[17] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors. *Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide.* SIAM, Philadelphia, PA, 2000.

[18] S. Bakshi, D. Deshmukh, and R. V. Ravikrishna. Multidimensional modeling of flow through piston-controlled ports using a multi-block, moving mesh algorithm. *Journal of Mechanical Engineering Science*, 218:207–222, 2004.

[19] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matt Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.3.2, Argonne National Laboratory, 2006.

[20] B.R. Bandini, D.L. Aumiller, and E. T. Tomlinson. An algorithm to correct for control rod cusping in the NESTLE multidimensional neutronics module of RELAP5-3D. In *International RELAP5-3D Users Seminar.* Montana, 2003.

[21] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition.* SIAM, Philadelphia, PA, 1994.

[22] Don Batory. The road to utopia: A future for generative programming. In *Domain Specific Program Generation, LNCS 3016*, pages 1–18, 2004.

[23] Lukasz Beca. Building collaborative problem-solving environments as shared places. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences.* IEEE, 2001.

[24] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84, 1989.

[25] Marsha J. Berger. On conservation at grid interfaces. *SIAM Journal of Numerical Analysis*, 24(4):967–985, October 1987.

[26] Marsha J. Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.

[27] Yansunori Bessho, Yuichiro Yoshimoto, Osamu Yokomizo, Tyutaro Yamashita, Masumi Ishikawa, and Akio Toba. Development of a three-dimensional core dynamics analysis program for commercial boiling water reactors. *Nuclear Technology*, 117:281–292, March 1997.

[28] S. S. Bhattacharyya, R. Leupers, and P. Marwedel. Software synthesis and code generation for signal processing systems. *IEEE Transactions on Circuits and Systems*, 47(9), September 2000.

[29] Aart J. C. Bik, Peter J. H. Brinkhaus, Peter M. W. Knijnenburg, and Harry A. G. Wijshoff. The automatic generation of sparse primitives. *ACM Transactions on Mathematical Software*, 24(2):190–225, 1998.

[30] Joseph D. Blahovec, Lester A. Bowers, John W. Luginsland, Gerald E. Sasser, and John J. Watrous. 3-d ICEPIC simulations of the relativistic klystron oscillator. *IEEE Transactions on Plasma Science*, 28(3), June 2000.

[31] Ruxandra Bondarescu, Gabrielle Allen, Gregory Daues, Ian Kelley, Michael Russell, Edward Seidel, John Shalf, and Malcolm Tobias. The astrophysics simulation collaboratory portal: a framework for effective distributed research. *Future Generation Computer Systems*, 21(2):259–270, 2005.

[32] David L. Brown, William D. Henshaw, and Dan Quinlan. Overture: Object-oriented tools for applications with complex geometry. In *Computing in Object-Oriented Parallel Environments: Third International Symposium, ISCOPE 99*, pages 96–107, San Francisco, CA, 1999. Springer.

[33] Kristopher R. Buschelman, William Gropp, Lois C. McInnes, and Barry F. Smith. PETSc and Overture: Lessons learned developing an interface between components. In *Proceedings of the IFIP TC2/WG2.5 Working Conference on the Architecture of Scientific Software*, pages 57–68, Deventer, The Netherlands, The Netherlands, 2001. Kluwer, B.V.

[34] H.S. Carslaw and J. C. Jaeger. *Conduction of Heat in Solids.* Oxford, 1959.

[35] Lee L. Carter and Randy A. Schwartz. Visual creation of lattice geometries for MCNP criticality calculations. *Transactions of the American Nuclear Society*, 77:223, 1997.

[36] Henri Casanova and Jack Dongarra. Using agent-based software for scientific computing in the NetSolve system. *Parallel Computing*, 24(12–13):1777–1790, 1998.

[37] Arun Chauhan and Ken Kennedy. Optimizing strategies for telescoping languages: procedure strength reduction and procedure vectorization. In *ICS '01: Proceedings of the 15th international conference on Supercomputing*, pages 92–101, New York, NY, USA, 2001. ACM Press.

[38] G. Chesshire and W. D. Henshaw. Composite overlapping meshes for the solution of partial differential equations. *Journal of Computational Physics*, 90:1–64, 1990.

[39] G. Chesshire and W. D. Henshaw. A scheme for conservative interpolation on overlapping grids. *SIAM Journal of Scientific Computing*, 15:819–845, July 1994.

[40] C.H. Cho, J.H. Choi, J.Y. Huh, and S. Y. Lee. CANDU NPP reactor regulating system modeling using CATHENA. *Annals of Nuclear Energy*, 23(11):909–918, 1996.

[41] Melvyn Ciment. Stable difference schemes with uneven mesh spacings. *Mathematics of Computation*, 25(114):219–227, April 1971.

[42] Comsol. *FEMLAB.* Comsol, 2001.

[43] Krzysztof Czarnecki and Ulrich W. Eisenecker. Components and generative programming (invited paper). In *ESEC/FSE-7: Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT*

international symposium on Foundations of software engineering, pages 2–19, London, UK, 1999. Springer-Verlag.

[44] Krzysztof Czarnecki, K. Osterbye, and M. Volter. Generative programming. In *Object-Oriented Technology. ECOOP 2002 Workshop, LNCS 2548*, pages 15–29, 2002.

[45] P. D'Ambra, M. Danelutto, D. di Serafino, and M. Lapegna. Advanced environments for parallel and distributed applications: a view of current status. *Parallel Computing*, 28:1637–1662, 2002.

[46] Simon Day. Energy group structure for modelling. Technical Report MNR-TR 2004-07-09, McMaster University Nuclear Reactor, 2004.

[47] N DeBardeleben, W Ligon, S. Pandit, and D. Stanzione. Coven - a framework for high performance problem solving environments. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing HPDC-11*, pages 291–298, 2002.

[48] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 20(3):720–755, 1999.

[49] Jack Dongarra, Shirley Moore, and Anne Trefethen. Numerical libraries and tools for scalable parallel cluster computing. *The International Journal of High Performance Computing Applications*, 15(2):175–180, Summer 2001.

[50] A.Yu. Doroshenko, V.M. Piksaikin, and M.Z. Tarasko. The energy spectrum of delayed neutrons from thermal neutron induced fission of 235u and its analytical approximation. Technical Report UDC 539.173.84, State Scientific Center of the Russian Federation Institute for Physics and Power Engineering, 1999.

[51] Tzvetan Drashansky, Elias N. Houstis, Naren Ramakrishnan, and John R. Rice. Networked agents for scientific computing. *Communications of the ACM*, 42(3):48–ff., 1999.

[52] Paul F. Dubois. Designing scientific components. *IEEE Computational Science and Engineering*, pages 84–90, September 2002.

[53] James Duderstadt and Louis Hamilton. *Nuclear Reactor Analysis*. John Wiley and Sons, 1976.

[54] William P. Eaton, Fernando Bitsie, James H. Smith, and David W. Plummer. A new analytical solution for diaphragm deflection and its application to a surface-micromachined pressure sensor. In *International Conference on Modeling and Simulation of Microsystems*, 1999.

[55] David W. Eccles and Paul T. Groth. Wolves, football, and ambient computing: facilitating collaboration in problem solving systems through the study of human and animal groups. In *NordiCHI '04: Proceedings of the third Nordic conference on Human-computer interaction*, pages 269–275, New York, NY, USA, 2004. ACM Press.

[56] A.H. El-Hag, Jayaram S.H., and E.A. Cherney. Influence of shed parameters on the aging performance of silicone rubber insulators in salt-fog. *IEEE Transactions on Dielectrics and Electrical Insulation*, 10, 2003.

[57] A.M. El-Messiry. ETRR-2 control rod withdrawal accident. *Annals of Nuclear Energy*, 27:745–755, 2000.

[58] Thilo Ernst, Tom Rother, Franz Schreier, Jochen Wauer, and Wolfgang Balzer. DLR's virtuallab: Scientific software just a mouse click away. *IEEE Computing in Science and Engineering*, pages 70–79, January 2003.

[59] Martin Erwig and Zhe Fu. Software reuse for scientific computing through program generation. *ACM Transactions on Software Engineering Methodology*, 14(2):168–198, 2005.

[60] M.C. Espirito-Santo, P. Goncalves, M. Pimenta, P. Rodrigues, B. Tome, and A. Trindade. Applications of geant4 in astroparticle experiments. *IEEE Transactions on Nuclear Science*, 51:1373–1377, August 2004.

[61] Anne Feltus and Kristofer Labowski. Analyses of ATWSs using the PVM-coupled RETRAN thermal-hydraulics and STAR three-dimensional kinetics codes. *Transactions of the American Nuclear Society*, 76:291–292, 1997.

[62] Dieter Fensel and Enrico Motta. Structured development of problem solving methods. *IEEE Transactions on Knowledge and Data Engineering*, 13(6), 2001.

[63] Bernd Fischer and Johann Schumann. Autobayes: a system for generating data analysis programs from statistical models. *Journal of Functional Programming*, 13(3):483–508, 2003.

[64] G. Fox, D. Gannon, and M. Thomas. A summary of grid computing environments. *Concurrency and Computation: Practice and Experience*, 14(13-15), 2002.

[65] Geoffrey Fox. E-science meets computational science and information technology. *IEEE Computing in Science and Engineering*, pages 84–85, July 2002.

[66] E. Gallopoulos, E. Houstis, and J.R. Rice. Computer as thinker/doer: problem-solving environments for computational science. *IEEE Computational Science and Engineering*, 1:11–23, Summer 1994.

[67] D. Gannon, R. Bramley, T. Stuckey, J. Villacis, J. Balasubramanian, E. Akman, F. Breg, and S. Diwan. Developing component architectures for distributed scientific problem solving. *IEEE Computational Science and Engineering*, 5:50–63, April 1998.

[68] David R. Gardner, Joseph P. Castro, Paul N. Demmie, Mark A. Gonzales, Gary L. Hennigan, Michael F. Young, and Sudip S. Dosanjh. Developing a flexible system-mnodeling environment for engineers. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, pages 1549–1558. IEEE Press, 2002.

[69] Wm. J. Garland. CATHENA simulation of the January 1994 fuelling incident. Technical Report MNR-TR 97-01, McMaster University Nuclear Reactor, 1997.

[70] C. William Gear. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice-Hall, 1971.

[71] D. Gilbert, Wm. J. Garland, and T. Ha. MNRSIM: An interactive visual model which links thermal hydraulics, neutron production and other phenomean. In *Proceedings of the 6th International Conference on Simulation Methods in Nuclear Engineering.* Canadian Nuclear Society, October 2004.

[72] D. Gilbert, Wm. J. Garland, and W.F.S. Poehlman. Application of a combined eulerian-lagrangian updating method to the rod cusping problem. In *Proceedings of the 6th International Conference on Simulation Methods in Nuclear Engineering.* Canadian Nuclear Society, October 2004.

[73] D. Gilbert, J.E. Roman, Wm.J. Garland, and W.F.S. Poehlman. Simulating control rod and fuel assembly motion using moving meshes. *Annals of Nuclear Engineering*, to appear, 2007.

[74] O. Glockler. Testing the dynamics of shutdown systems instrumentation in reactor trip measurments. *Progress in Nuclear Energy*, 43:91–96, 2003.

[75] A. Goel, C. Phanouriou, F. Kamke, C. J. Ribbens, C. A. Shaffer, and L. T. Watson. WBCSIM: A prototype problem solving environment for wood-based composites simulations. *Engineering with Computers*, 15:198–210, 1999.

[76] B.N. Hanna. Cathena: A thermalhydraulic code for candu analysis. *Nuclear Engineering and Design*, 180:113–131, 1998.

[77] Michael Heath. *Scientific Computing: An Introductory Survey.* McGraw-Hill, 2002.

[78] T. C. Henderson, P. A. McMurty, P. J. Smith, G. A. Voth, C. A. Wight, and D. W. Pershing. Simulating accidental fires and explosions. *Computing in Science and Engineering*, pages 64–76, March 2000.

[79] Vicent Hernadez, Jose E. Roman, Andres Tomas, and Vicent Vidal. SLEPc users manual: Scalable library for eigenvalue problem compuations. Technical Report DSIC-II/24/02, Universidad Politecnica De Valencia, 2006.

[80] V. Hernandez and J. E. Roman. High-quality computational tools for linear-algebra problems in FEM electromagnetic simulation. *IEEE Antennas and Propagation Magazine*, 46(6):110–119, December 2004.

[81] V. Hernandez, J. E. Roman, A. M. Vidal, and V. Vidal. Calculation of lambda modes of a nuclear reactor: A parallel implementation using the implicitly restarted Arnoldi method. *Lecture Notes in Computer Science*, 1573:43–57, 1999.

[82] V. Hernandez, J. E. Roman, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Transactions on Mathematical Software*, 31(3):351–362, September 2005.

[83] J. Herrington. *Code Generation in Action.* Manning, 2003.

[84] E. N. Houstis, J. R. Rice, S. Weerawarana, A. C. Catlin, and P. Papachiou. PELLPACK: A Problem-Solving Environment for PDE-Based Applications on Multicomputer Platforms. *ACM Transactions on Mathematical Software*, 24(1):3–73, March 1998.

[85] E. N. Houstis, J. R. Rice, S. Werrawarana, A. C. Catlin, and P. Papachiou. PELLPACK: A problem-Solving Environment for PDE-Based Applications on Multicomputer Platforms. *ACM Transactions on Mathematical Software*, 24(1):30–73, March 1998.

[86] Elias N. Houstis, Ann Christine Catlin, Nitesh Dhanjani, John R. Rice, Naren Ramakrishnan, and Vassilios S. Verykios. Mypythia: a recommendation portal for scientific software and services. *Concurrency and Computation: Practice and Experience*, 14(13-15):1481–1505, 2002.

[87] Elias N. Houstis, Anne C. Catlin, and John R. Rice. PYTHIA-II: A Knowledge Database system for Managing Performance Data and Recommending Scientific Software. *ACM Transactions on Mathematical Software*, 26:227–253, June 2000.

[88] Elias N. Houstis and John R. Rice. Future problem solving environments for computational science. *Mathematics and Computers in Simulation*, 54:243–257, 2000.

[89] Elias N. Houstis, John R. Rice, Efstratios Gallopoulos, and Randall Bramley, editors. *Enabling Technologies for Computational Science.* Kluwer Academic Publishers, 2000.

[90] IAEA. Research reactor core conversion from the use of highly enriched uranium to the use of lowe enriched uranium fuels guidebook. Technical Report IAEA-TECDOC-233, International Atomic Energy Agency, Vienna, 1980.

[91] M.W.A. Ibrahim, J.R. Saunders, and A. Moussa Walied. Hydrogel microvalve device modeling and simulation. In *Proceedings of the 2005 International Conference on MEMS, NANO and Smart Systems.* IEEE, 2005.

[92] S. Jan, J. Collot, M.L. Gallin-Martel, P. Martin, F. Mayet, and E. Tournefier. GePEToS: a Geant4 monte carlo simulation package for positron emission tomography. *IEEE Transactions on Nuclear Science*, 52:102–106, Febuary 2005.

[93] J. J. Jeong, K.S. Ha, B.D. Chung, and W.J. Lee. Development of a multidimensional thermal-hydraulic system code, mars 1.3.1. *Annals of Nuclear Energy*, 26:1611–1642, 1999.

[94] T. Jevremovic, J. Vujic, and K. Tsuda. Anemona- a neutron transport code for general geometry reactor assemblies based on the method of characteristics and r-function solid modeler. *Annals of Nuclear Energy*, 28:125–152, 2001.

[95] X. Jiao, M. T. Campbell, and M. T. Heath. Roccom: An object-oriented, data-centric software integration framework for multiphysics simulations. In *Proceedings of the 17th annual international conference on Supercomputing*, pages 358–368. ACM Press, 2003.

[96] Han-Sem Joo. *Resolution of the Control Rod Cusping Problem for Nodal Methods.* PhD thesis, Dept. of Nuclear Engineering, MIT, 1984.

[97] George Chin Jr., Ruby Leung, Karen Schuchardt, and Debbie Gracio. New paradigms in problem solving environments for scientific computing. In *Proceedings of the 7th International Conference on Intelligent User Interfaces*, pages 39–46. ACM Press, 2002.

[98] Dale k. Pace. Modeling and simulation verification and validation challenges. *Johns Hopkins Applied Technical Digest*, 25(2), 2004.

[99] K. Kao and M. Liou. Advance in overset grid schemes: From chimera to DRAGON grids. *AIAA Journal*, 33, October 1995.

[100] K. Kawano, S. Shahrani, T. Mori, M. Kuroda, and M.M Tentzeris. Dynamic and electrical analysis of mems capacitor with accelerated motion effects. In *Wireless Communications and Applied Computational Electromagnetics*, pages 759–762. IEEE/ACES, 2005.

[101] K. Kennedy, B. Broom, and K. Cooper. Telescoping languages: A strategy for automatic generation of scientific problem-solving systems from annotated libraries. *Journal of Parallel and Distributed Computing*, 61:1803–1826, 2001.

[102] Liaquat Ali Khan, Nasir Ahmad, M.S. Zafar, and Ayaz Ahmad. Reactor physics calculations and their experimental validation for conversion and upgrading of a typical swimming pool type research reactor. *Annals of Nuclear Energy*, 27:873–885, 2000.

[103] Do Sam Kim and Nam Zin Cho. Kinetics calculation under space-dependent feedback in analytic function expansion nodal method via solution decomposition and galerkin scheme. *Nuclear Science and Engineering*, 140:267–284, 2002.

[104] Glenn F. Knoll. *Radiation Detection and Measurment.* John Wiley and Sons, 1989.

[105] V.V. Komarov and V.V. Yakovlev. Simulation of components of microwave heating applications by femlab, microwavelab and quickwave-3d. In *Proceedings of the 36th Microwave Power Symposium.* Sna Francisco, 2001.

[106] Steven E. Koonin. *Computational Physics.* Benjamin/Cummings Publishing Company, Inc, 1986.

[107] Y.M. Kwon, Y.b. Lee, W.P. Chang, and D. Hahn. A computer code for evaluating inherent safety of an advanced lmfbr. *Transactions of the American Nuclear Society*, 85:213–214, 2001.

[108] M. Kyaw, R. Caplan, D. Mogdans, and L. Leff. A compiler for generating the global stiffness matrix for symbolically defined regular finite element analysis grids. *Computers and Structures*, 76:461–469, 2000.

[109] Rubin H. Landau. *Computational Physics: Problem Solving with Computers.* John Wiley and Sons, Inc., 1997.

[110] R. D. Lawrence. Progress in nodal methods for the solution of the neutron diffusion and transport equations. *Progress in Nuclear Energy*, 17(3):271–301, 1986.

[111] F. Lei, R.R. Truscott, C.S. Dyer, B. Quaghebeur, D. Heynderickx, R. Nieminen, H. Evans, and E. Daly. MULASSIS: a Geant4-based multi-layered shielding simulation tool. *IEEE Transactions on Nuclear Science*, 49:2788–2793, December 2002.

[112] F. Leszcynski, D. Aldamada, and A. Trkov. Wims-d library update. Technical report, International Atomic Energy Agency, Vienna, 2006.

[113] B. J. Lewis, W. T. Thompson, F. Akbari, D. M. Thompson, C. Thurgood, and J. Higgs. Thermodynamic and kinetic modeling of fuel oxidation behavior in operating defective fuel. *Journal of Nuclear Materials*, 328:180–196, 2004.

[114] Vanco Litovski, Miona Andrejevic, and Mark Zwolinski. Ann based modleing, testing and diagnosis of mems. In *7th Seminar on Neural Network Applictions in Electrical Engineering*. IEEE, 2004.

[115] Shahani Markus, Sanjiva Weerawarana, Elias N. Houstis, and John R. Rice. Scientific computing via the web: The net pellpack pse server. *IEEE Computational Science and Engineering*, pages 43–51, July 1997.

[116] C. Wayne Mastin. Implicit finite difference methods on composite grids. *Journal of Computational and Applied Mathematics*, 20:317–323, 1987.

[117] Alain Monier. *Application of the Collocation Technique to the Spatial Discretization of the Generalized Quasistatic Method for Nuclear Reactors*. PhD thesis, Dept. of Nuclear Engineering, Ecole Polytechnique Montreal, 1991.

[118] T. Sinh Nguyen. *A Multigrid Method Applied to Reactor Kinetics*. PhD thesis, Department of Engineering Physics, McMaster University, 2006.

[119] Beat Niderost, Marco van de Giessen, Wim Lourens, and Jon Krom. The webumbrella web-based access to distributed plasma-physics measurement t data. *IEEE Transactions on Nuclear Science*, 49:1579–1583, june 2002.

[120] B. Nilsson and J. Eborn. An object-oriented model database for thermal power plants. In *European simulation conference*, September 1995.

[121] Suely Oliveira and Yuanhua Deng. Preconditioned krylov subspace methods for transport equations. *Progress in Nuclear Energy*, 33(1/2):155–174, 1998.

[122] Symour V. Parter. On the overlapping grid method for elliptic boundary value problems. *SIAM Journal of Numerical Analysis*, 36(3):819–852, 1999.

[123] M. Patrick and R. Voight. Advancing simulation science and engineering at disciplinary interfaces. *Computing in Science and Engineering*, pages 14–16, March 2000.

[124] R. Christina Penland, Yousry Y. Azmy, and Paul J. Turinsky. Error analysis of the nodal expansion method for solving the neutron diffusion equation. *Nuclear Science and Engineering*, 125:284–299, 1997.

[125] Rod Perala. An introduction to PEPSE ('parallel electromagnetics problem solving environment'): and considerations for parallelization of a finite differenced time-domain solver. *International Journal of Numerical Modeling: Electronic Networks, Devices and Fields*, 8:187–203, 1995.

[126] Robert E. Peterkin and John W. Luginsland. A virtual prototyping environment for directed-energy concepts. *IEEE Computing in Science and Engineering*, pages 42–49, March 2002.

[127] Craig E. Peterson, John g. Shatford, Ardesar Irani, Nicholas g. Trikouros, Antonia F. Dias, and Lance J. Agee. Maint-steam-line-break analysis of tmi-1 using retran-3d. *Nuclear Technology*, 128:233–243, November 1999.

[128] S. J. Plimpton, Bruce Hendrickson, and J. R. Stewart. A parallel rendezvous algorithm for interpolation between multiple grids. *Journal of Parallel and Distributed Computing*, 64:266–276, 2004.

[129] David Potter. *Computational Physics.* John Wiley & Sons, 1973.

[130] A. Quist. *Application of Mathematical Optimization Techniques to Nuclear Reactor Reload Pattern Design.* PhD thesis, Dept. of Mathematics, Delft, 2000.

[131] M. Rai. A conservative treatment of zonal boundaries for Euler equation calculations. *Journal of Computational Physics*, 62:472–503, 1986.

[132] Man Mohan Rai. A relaxation approach to patched-grid calculations with the Euler equations. *Journal of Computational Physics*, 66:99–131, 1986.

[133] Matjaz Ravnik, Tomaz Zagar, and Andreja Persic. Fuel element burnup determination in mixed triga core using reactor calculations. *Nuclear Technology*, 128:35–45, October 1999.

[134] John R. Rice. From scientific software libraries to problem-solving environments. *IEEE Computational Science and Engineering*, pages 44–53, Fall 1996.

[135] John R. Rice. A perspective on computational science in the 21st century. *IEEE Computational Science and Engineering*, pages 14–16, March 1999.

[136] John R. Rice. *Future Challenges for Scientific Simulation*, pages 6–17. In Houstis et al. [89], 2000.

[137] Patrick J. Roache. *Verification and Validation in Computational Science and Engineering.* hermosa, 1998.

[138] R. Rosner, A. Calder, J. Dursi, B. Fryxell, D. Lamb, J. Niemeyer, K. Olson, P. Ricker, F. Timmes, J. Truran, H. Tufo, Y. Young, and M. Zingale. Flash code: Studying astrophysical thermonuclear flashes. *IEEE Computing in Science and Engineering*, pages 33–41, March 2000.

[139] H. Rostaing, J. Stepanek, O. Cugat, C. Dieppedale, and J. Delamare. Magnetic, out-of-plane, totally integrated bistable micro actuator. In *Proceedings of the 13th International Conference on Solid-State Sensors, Actuators and Microsystems*, pages 1366–1370. IEEE, 2005.

[140] Christopher J. Roy. Review of code and solution verification procedures for computational simulation. *Jornal of Computational Physics*, 205(1):131–156, 2005.

[141] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, PA, second edition edition, 2003.

[142] G. Santin, D. Strul, D. Lazaro, L. Simon, M. Krieguer, M.V. Martins, V. V. Breton, and C. C. Morel. Gate: a Geant4-based simulation platform for pet and spect integrating movement and time management. *IEEE Transactions on Nuclear Science*, 50:1516–1521, October 2003.

[143] R. Scheichl. Parallel solvers for the transient multi-group neutron diffusion equations. *International Journal for Numerical Methods in Engineering*, 47:1751–1771, 2000.

[144] Max Schlee and Jean Vanderdonckt. Generative programming of graphical user interfaces. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 403–406, New York, NY, USA, 2004. ACM Press.

[145] Michael Schlegel, Goran Herrmann, and Dietmar Muller. Application of the multi architecture modelling design method to system level mems simulation. In *Design, Test, Integration and Packaging of MEMS/MOEMS*. IEEE, 2003.

[146] Randy A. Schwartz and Lee L. Carter. Visual editor to create and display MCNP input files. *Transactions of the American Nuclear Society*, 77:311–312, 1997.

[147] K.A. Smith, A.J. Baratta, and G.E. Robinson. Coupled RELAP5 and CONTAIN accident analysis using PVM. *Nuclear Safety*, 36(1):94–108, January-June 1995.

[148] John E. So, Thomas J. Downar, Raghunandan Janardhan, and Howard Jay Siegel. Mapping conjugate gradient algorithms for neutron diffusion applications onto SIMD, MIMD, and mixed-mode machines. *International Journal of Parallel Programming*, 26(2):183–207, April 1998.

[149] V. Sreedharan and D. Zhang. Finite element modeling of cellular responses of gap junction connected osteocytes under extremely low-frequency electromagnetic fields. In *Proceedings of the 2003 IEEE 29th Annual Bioengineering Conference*, pages 160–161. IEEE Press, March 2003.

[150] Srinath Srinivasa. Connotations of problem solving. *SIGSOFT Software Engineering Notes*, 26(6):80–82, 2001.

[151] Daniel C. Stanzione. *Problem Solving Environment Infrastructure for High Performance Computer Systems.* PhD thesis, Dept. of Electrical Engineering, Clemson University, December 2000.

[152] Goran Starius. Composite mesh difference methods for elliptic boundary value problems. *Numerische Mathematic*, 28:243–258, 1977.

[153] G. W. Stewart. A Krylov–Schur algorithm for large eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 23(3):601–614, 2001.

[154] T. M. Sutton and B. N. Aviles. Diffusion theory methods for spatial kinetics calculations. *Progress in Nuclear Energy*, 30(2):119–182, 1996.

[155] H. S. Tang and T. Zhou. On nonconservative algorithms for grid interfaces. *SIAM Journal of Numerical Analysis*, 37(1):173–193, 1999.

[156] Masahiro Tatsumi and Akio Yamamoto. Object-oriented three-dimensional fine-mesh transport calculation on parallel/distributed environments for advanced reactor core analyses. *Nuclear Science and Engineering*, 141:190–217, 2002.

[157] Joe F. Thompson. Composite grid generation code for general 3-d regions- the EAGLE code. *AIAA*, 26(3), March 1988.

[158] Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill, editors. *Handbook of Grid Generation.* CRC Press, 1999.

[159] Joe F. Thompson, Z.U.A. Warsi, and C. Wayne Mastin. *Numerical Grid Generation.* Elseview Science Publishing Co., Inc., 1985.

[160] N. K. Todorova and K. Ivanov. Investigation of spatial coupling aspects for coupled code application in PWR safety analysis. *Annals of Nuclear Energy*, 30:189–209, 2003.

[161] Paul van der Mark, Lex Wolters, and Gerard Cats. Using semi-lagrangian formulations with automatic code generation for environmental modeling. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 229–234, New York, NY, USA, 2004. ACM Press.

[162] Robert van Engelen. *Ctadel: A Generator of Efficient Codes.* PhD thesis, Department of Computer Science, Leiden University, October 1998.

[163] A. W. M. van Schijndel. Modeling and solving building physics problems with femlab. *Building and Environment*, 38:319–327, February 2003.

[164] R. vanEngelen and L. Wolters. Tomorrow's weather forecast: Automatic code generation for atmospheric modeling. *IEEE Computational Science and Engineering*, pages 22–31, July 1997.

[165] Marc Vass, John M. Carroll, and Clifford A. Shaffer. Supporting creativity in problem solving environments. In *C&C '02: Proceedings of the 4th conference on Creativity & cognition*, pages 31–37, New York, NY, USA, 2002. ACM Press.

[166] G. Verdú, R. Miró, D. Ginestar, and V. Vidal. The implicit restarted Arnoldi method, an efficient alternative to solve the neutron diffusion equation. *Annals of Nuclear Energy*, 26(7):579–593, 1999.

[167] Franz J. Vesely. *Computational Physics: An Introduction.* Plenum Press, 1994.

[168] Rich Vuduc and James Demmel. Code generators for automatic tuning of numerical kernels: Experiences with FFTW. In *SAIG*, pages 190–211, 2000.

[169] Z. J. Wang. A fully conservative interface algorithm for overlapped grids. *Journal of Computational Physics*, 122:96–106, 1995.

[170] James S. Warsa, Todd A. Wareing, Jim E. Morel, John M. McGhee, and Richard B. Lehoucq. Krylov subspace iterations for deterministic k-eigenvalue calculations. *Nuclear Science and Engineering*, 147:26–42, 2004.

[171] J Whitlock. Maclib69.dat: Simplified 69-group neutron cross-section library. Technical Report Version 1.0, McMAster University Nuclear Reactor, 1992.

[172] Andrew M. Wissink, Richard D. Hornung, Scott R. Kohn, Steve S. Smith, and Noah Elliott. Large scale parallel structured amr calculations using the samrai framework. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, pages 6–6, New York, NY, USA, 2001. ACM Press.

[173] Samuel Shaw Ming Wong. *Computational Methods in Physics and Engineering.* World Scientific Publishing Co., 1997.

[174] Poting Wu and Elias N. Houstis. EPPOD: A Problem Solving Environment for Parallel Electronic Prototyping of Physical Object Design. *Journal of Parallel and Distributed Computing*, 42:157–172, 1997.

[175] Y. F. Xie, G. L. Browning, and G. Chesshire. A two-dimensional composite grid numerical model based on the reduced system for oceanography. *SIAM Journal of Scientific computing*, 17(5):1122–1134, September 1996.

[176] Jianguo Xin, Katia Pinchedez, and Joseph E. Flaherty. Implementation of hierarchical bases in FEMLAB for simplicial elements. *ACM Trans. Math. Softw.*, 31(2):187–200, 2005.

[177] Akio Yamamoto. A simple and efficient control rod cusping model for three-dimensional pi n-by-pin core calculations. *Nuclear Technology*, 145:11–17, January 2004.

[178] N. Yazdi, H. Sane, T.D. Kudrle, and C.H. Mastrangelo. Robust sliding-mode control of electrostatic torsional micromirrors beyond the pull-in limit. In *Proceedings of the 12th International conference on Solid-State Sensors, Actuators and Microsystems*, pages 1450–1453. IEEE, 2003.

[179] M. Yeung and G. Jiang. Development of an efficient three-dimensional reactor core model for simulator application. *Nuclear Technology*, 97:352–361, 1992.

[180] H. Yoshida and K. Minamimoto. Secure graphical user interface for geant4. In *IEEE Nuclear Science Symposium Conference Record*, volume 3, pages 1614–1616. IEEE, October 2003.

[181] Leonid Yu. Zaslavsky. An adaptive algebraic multigrid for multi-group neutron diffusion reactor core calculations. *Applied Mathematics and Computation*, 53:13–26, 1993.

[182] H. Zhang and E. E. Lewis. An adaptive approach to variational nodal diffusion problems. *Nuclear Science and Engineering*, 137:14–22, 2001.

[183] Hui Zhang and E. E. Lewis. Spatial adaptivity applied to the variational nodal Pn equations. *Nuclear Science and Engineering*, 142:57–663, 2002.

Appendix 1

Glossary of Symbols and Terms

ARPACK	eigensolver library
BLAS	linear algebra library
CATHENA	modeling tool for CANDU reactors
CG	Conjugate Gradient, iterative solution for symmetric systems
CFL	Courant Friedrichs Lewy, a stability condition
FEMLAB	commercial problem solving environment
GMRES	iterative solution method for nonsymmetric systems
HEU	highly enriched uranium
Jacobi	simple iterative solution method based on average of neighbors
Krylov-Schur	eigenvalue solution technique
Krylov Subspace	Subspace spanned by the vectors $b, bA^1, bA^2, ..., bA^{n-1}$
MNR	McMaster Nuclear Reactor
MeV	Mega electron Volt, measure of energy
PDEs	partial differential equations
PETSc	general linear solver
PSE	problem solving environment
SLEPc	eigenvalue solver
WIMS	tool for evaluating cross sections
U235, U238	Uranium

Cartesian Grid	square grid
direct solver	linear solution methodology, using fixed number of steps
eigenvalue	the scalar solution λ, to the matrix vector problem $Ax = \lambda x$
explicit method	mathematical formulation, can be solved in one step
implicit method	mathematical formulation based on simultaneous definitions
iterative method	linear solution method based on a sequence of steps
mk	milli-k, 1/1000 of k, reactor multiplication rate
nodal method	nonlinear solution strategy used in nuclear engineering
nonconformal	type of mesh, with nonaligned vertices
patched mesh	non overlapping multi-block mesh
sparse matrix	a matrix with the property that 99% or more of entries are zero

A	coefficient matrix
B	2nd coefficient matrix, often singular in neutron diffusion equation
C	delayed precursor concentration, in neutron diffusion equation
C	heat capacity
c	wave velocity in a medium
D	diffusion constant
h	step size
k	constant
k_{eff}	reactor multiplication rate
S	source term
T	temperature
t	time
U	general field
v	neutron speed

β	relative yield of delayed precursor group
λ	in neutron diffusion equation, decay rate of precursor group
λ	in matrix equation, an eigenvalue
ρ	reactivity, $\rho = (k_{eff} - 1)/k_{eff}$
ρ	density
Σ_a	absorption cross section
Σ_s	scattering cross section
Σ_f	fission cross section
Σ_R	removal cross section
σ	shift value
ϕ, Φ	neutron flux
ν	neutrons produced in a single fission
ω	angular frequency
χ	fission spectrum
χ^D	delayed fission spectrum

Appendix 2

Fundamental Numerical Algorithms

Several fundamental algorithms discussed in the body of the thesis are presented in this appendix.

Conjugate Gradient

The conjugate gradient algorithm is an example of a typical spectral method, the following presentation is taken from [21]. The conjugate gradient method is the oldest and best known of the non-stationary methods. The method proceeds by generating vector sequences of iterates, residuals corresponding to the iterates, and search directions used in updating the iterates and residuals. Although the length of these sequences can become large, only a small number of vectors needs to be kept in memory. In every iteration of the method two inner products are performed in order to compute update scalars that are defined to make the sequences satisfy certain orthogonality conditions.

The iterates x^i are updated in each iteration by a multiple α^i of the search vector p^i

$$x^i = x^{i-1} + \alpha_i p^i$$

Correspondingly the residuals $r^i = b - Ax^i$ are updated as

$$r^i = r^{i-1} - \alpha_i q^i$$

where

$$q^i = Ap^i$$

The choice $\alpha = \alpha_i = r^{(i-1)T} r^{(i-1)} / p^{iT} A p^i$ minimizes $r^{iT} A^{-1} r^i$ over all possible choice for α. The search directions are updated using the residuals

$$p^i = r^i + \beta_{i-1} p^{i-1}$$

where the choice of $\beta_i = r^i / r^{(i-1)T} r^{i-1}$ ensures that p^i and Ap^{i-1} are orthogonal.

Algorithm 5 Pseudo Code for Conjugate Gradient

```
Compute r(0)=b-Ax(0) for some initial guess x(0)
for i=1, 2, ...
    p(i-1)=r(i-1)*r(i-1)
    if i=1
        p(1)=r(0)
    else
        B(i-1)=p(i-1)/p(i-2)
        p(i)=r(i-1) + B(i-1)p(i-1)
    endif
    q(i)=A*p(i)
    a(i)=p(i-1)/p(i) * q(i)
    x(i)=x(i-1)+a(i)*p(i)
    r(i)=r(i-1)-a(i)*q(i)
    check for convergence; continue if necessary
end
```

The Multi-Grid Algorithm

The linear multi-grid method [181, 118] can be an extremely fast solution technique, although its implementation diverges quite radically from the previously described iterative methods because it requires multiple problem representations.

The multi-grid algorithm is a divide-and-conquer technique for solving elliptic PDEs. The algorithm obtains an initial solution for an $n \times n$ grid by using an $\frac{n}{2} \times \frac{n}{2}$ grid as an approximation, taking every other grid point from the n-by-n grid. The coarser $\frac{n}{2} \times \frac{n}{2}$ grid is in turn approximated by an $\frac{n}{4} \times \frac{n}{4}$ grid, and so on recursively. The work done on a particular grid eliminates the error in half of the frequency components not eliminated on other grids. The work performed on a coarse grid makes the overall solution smoother, which is equivalent to getting rid of the high frequency error.

The problem is specified by the grid size i, the coefficient matrix is T^i, and the right hand side is b^i. Let P^i denote the problem of solving a discretized elliptic problem on a $(2^i+1) \times (2^i+1)$ grid, with $(2^i-1)^2$ unknowns. A sequence of related problems is generated $P^m, P^{m-1}, P^{m-2}, ...P^1$ on coarser and coarser grids, where the solution to P^{i-1} is a good approximation to the solution of P^i.

Let b^i be the right-hand-side of the linear system P^i. Let x^i denote an approximate solution to P^i. The restriction operator R^i takes a pair (b^i, x^i) and maps it to (b^{i-1}, x^{i-1}) , which is a simpler problem on the next coarser grid, with starting guess x^i:

$$(b^{i-1}, x^{i-1}) = R^i(b^i, x^i)$$

The restriction operator for simple problems can be computed as a weighted average of each grid point value with its nearest neighbors. The interpolation operator

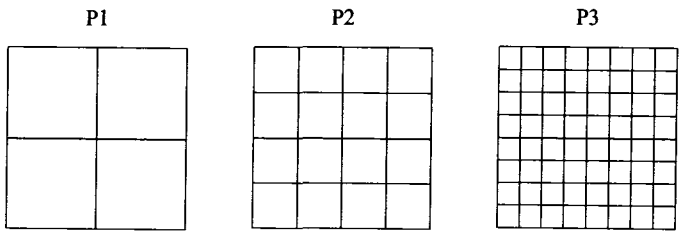


Figure 6.4: Sequence of Meshes used by Multi-Grid

Algorithm 6 Multi-grid V cycle

```
function MGV( b(i), x(i) )                  ... return an improved solution
                                            ... x(i) to P(i)
      if i = 1                              ... only one unknown
           compute the exact solution x(1) of P(1)
           return ( b(1), x(1) )
      else
           x(i) = S(i)( b(i), x(i) )        ... improve the solution
           ( b(i), d(i) ) = In(i-1)( MGV( R(i)( b(i), x(i) ) ) )
                                            ... solve recursively
           x(i) = x(i) - d(i)               ... correct fine grid solution
           x(i) = S(i)( b(i), x(i) )        ... improve the solution some more
           return ( b(i), x(i) )
      endif
```

In^{i-1} takes an approximate solution x^{i-1} and converts it to an approximation x^i for the problem P^i on the next finer grid:

$$(b^i, x^i) = In^{i-1}(b^{i-1}, x^{i-1})$$

The solution operator S^i take a problem P^i and approximate solution x^i, and computes an improved x^i.

$$x^i = S^i(b^i, x^i)$$

The improvements damp the high frequency components of the error. The basic multiplied V cycle can be summarized in Algorithm 6[2].

Multiple representations of a physical model of various resolutions are used by the method to accelerate convergence of the model. The solution process cycles between solving a high resolution version of the model and a low resolution of the model.

[2]Algorithm is reproduced from on-line course notes provided by Jim Demmel
http://www.cs.berkeley.edu/~demmel/cs267-1995/lecture25/lecture25.html

The low resolution representation of the problem accelerates the communication of information across the mesh.

Multi-grid methods have excellent performance characteristics. The spectral techniques presented in Chapter 2 will find a solution in at best in $O(n \cdot \log(n))$ iterations, in contrast the multi-grid method can solve a problem in $O(n)$ iterations. Multi-grid methods also adapt well to parallel implementations. For some problems constructing multiple similar representations may not be trivial (the In^i and R^i operators described in the algorithm), this is the principle difficulty associated with the multi-grid method.

Lanczos Algorithm

A summary of the Lanczos algorithm is presented here for the standard eigenvalue problem

$$Ax = \lambda x$$

where A is symmetric and real. The algorithm starts with a properly chosen starting vector v and builds up an orthogonal basis V_j of the Krylov subspace,

$$K^j(A, v) = span\{v, VA, A^2v, ..., A^{j-1}v\}$$

one column at a time. In each step just one matrix-vector multiplication

$$y = Ax$$

is needed. In the new orthogonal basis V_j the operator A is represented by a real symmetric tridiagonal matrix,

$$T_j = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{j-1} \\ & & \beta_{j-1} & \alpha_j \end{bmatrix}$$

which is also built up one row and column at a time, using the basic recursion,

$$AV_j = V_j T_i + re_j^*$$

with $V_j^* r = 0$. At any step j an eigensolution for T_j can be computed as

$$T_j s_i^{(j)} = s_i^{(j)} \theta_i^{(j)}$$

where the superscript (j) is used to indicate that these quantities change for each

Algorithm 7 Lanczos Method

```
start with r=v
B(0)=||r||2
for j=1,2,... until convergence
     v(j)=r/B(j-1)
     r=r-v(j-1)/B(j-1)
     A(j)=v(j)r
     r=r-v(j)*a(j)
     reorthogonalzie if necessary
     B(j)=||r||2
     compute approximate eigvenvalues T(j)=S*t(j)*S
     test bounds for convergence
end for
compute approximate eigenvectors X=V(j)S
```

iteration j. The Ritz value $\theta_i^{(j)}$and its Ritz vector,

$$x_i^{(j)} = V_j s_i^{(j)}$$

will be a good approximation to an eigenpair of A if the residual has a small norm. The Ritz pair is computed as

$$r_i^{(j)} = Ax_i^{(j)} - x_i^{(j)}\theta_i^{(j)} = AV_j s_i^{(j)} - V_j s_i^{(j)}\theta_i^{(j)} = (AV_j - V_j T_j)s_i^{(j)} = v_{j+1}\beta_j s_{j,i}^{(j)}$$

This norm satisfies

$$\left\|r_i^{(j)}\right\|_2 = \left|\beta_j s_{i,j}^{(j)}\right| = \beta_{j,i}$$

the algorithm needs to monitor the subdiagonal elements β_j of T and the last elements $s_{i,j}^{(j)}$ of its eigenvectors to generate an estimate of the norm of the residual. As soon as this estimate is small, the Ritz value $\theta_i^{(j)}$ can be flagged as converged to the eigenvalue λ_i.

Appendix 3

Example MOOSE PDEs

This appendix presents the MOOSE configuration equations which were used to specify the sparse matrices for the case study presented in Chapter 5. These equations capture the details of the neutron diffusion equation, and the multi-value integration method. Appendix 3 gives a partial listing of the source code which is generated by the MOOSE based on this configuration file for the 2 energy group case.

```
#########################################################
# Physical constants which define delayed precursors #
#########################################################
# Bet[i] taken from p64 D&H, Sum(B[i]=.007) total delayed precursors
Bet:=[.000266, .001491, .001316, .002849, .000896, .000182];
# Lam[i], taken from p64 D&H, Lam=1/T
Lam:=[.0183, .0458, .167, .448, 2.02, 4.49];
# V[i], average group velocity, defined as 13,891 * sqrt(eV) m/s,
# This vector is in cm/s, so the # are really big.
# oi is the old index, it changes depending on our mode.

###############################
# Delayed precursor equations #
###############################
Delayed:=[
    C[i]=sup[C[i],oi] + .5 * G_h *
   (-Lam[i]*C[i] +      fcal[G]*sum(Bet[i]*nu_Sigma_f[j]*T[j],j=1..G)
    -Lam[i]*sup[C[i],oi] +
               fcal[G]*sum(Bet[i]*nu_Sigma_f[j]*sup[T[j],oi],j=1..G)
             ), C[i] ];

#Delayed precursor calculation for steady state case
Delayed_ss:=[
    0=-Lam[i]*C[i] +fcal[G]*sum(Bet[i]*nu_Sigma_f[j]*T[j],j=1..G)
    , C[i] ];
```

```
#######################################################################
# Source Terms and Removal Terms for the Neutron Diffusion Equation #
#######################################################################
# According to D&H delayed neutrons appear with a different distribution
# than prompt neutrons. pChi approximates this
Source_new:=(.993)*Chi[i]*fcal[G]*sum(nu_Sigma_f[j]*T[j],j=1..G)
                + pChi[i]*sum(Lam[j]*C[j],j=1..6) ;
Source_old:=(.993)*Chi[i]*fcal[G]*sum(nu_Sigma_f[j]*sup[T[j],oi],j=1..G)
                + pChi[i]*sum(Lam[j]*sup[C[j],oi],j=1..6) ;
Removal_new:=LAPL(1/(3*Sigma_tr[i])*T[i]) - T[i]*Sigma_r[i]
                + sum(T[j]*Sigma_s[j][i],j=1..G) - T[i]*Sigma_s[i][i];
Removal_old:=LAPL(1/(3*Sigma_tr[i])*sup[T[i],oi]) - sup[T[i],oi]*Sigma_r[i]
                      sup[T[i],oi]*Sigma_s[i][i];



###############################################
# Third order Multi-Value Integration Method #
###############################################
s1:=(G_h/G_ho);
s2:=(G_h*G_h)/(G_ho*G_ho);

Th[i]:=sup[T[i],oi]+s1*sup[T1[i],oi]+  s2*sup[T2[i],oi];
Th1[i]:=             s1*sup[T1[i],oi]+2*s2*sup[T2[i],oi];
Th2[i]:=                                s2*sup[T2[i],oi];

alpha:= G_h * (V[i] *
               ( Removal_new + G_subcrit + Source_new )
                             -Th1[i]/G_h ) ;

# third order method, Stiffly stable.
Flux:= [T[i]=Th[i]  + alpha * (2./3.), T[i]],
      [T1[i]=Th1[i] + alpha  ,         T1[i]],
      [T2[i]=Th2[i] + alpha * (1./3.), T2[i]];
# steady state including precursors
Flux_ss:=[ 0 = V[i] *
        ( Removal_new + G_subcrit + Source_new ), T[i] ],
         [T1[i]=0,T1[i]],
         [T2[i]=0,T2[i]];
```

```
#######################################
# PDE Specification by Equation Group #
#######################################
if eq_grp = 1
then
    # initialization, set all variables = 0
    initseq:=[T[i]=0,T[i]],[T1[i]=0,T1[i]],[T2[i]=0,T2[i]];
    PDEs:=[seq(initseq,i=1..G),
           seq([C[i]=0,C[i]],i=1..6)];
elif eq_grp = 2
then
    # Solve even step of integration
    oi:=3;
    PDEs:=[seq(Flux,i=1..G),seq(Delayed,i=1..6)];
elif eq_grp = 3
then
    # Solve odd step of integration
    oi:=2;
    PDEs:=[seq(Flux,i=1..G),seq(Delayed,i=1..6)];
elif eq_grp = 4
then
    # Solve steady state precondition for integration
    PDEs:=[seq(Flux_ss,i=1..G),seq(Delayed_ss,i=1..6)];
end if;
```

Appendix 4

Example Generated Code

```
/*-------------------------------------------------------------
 * This function is an example of the code generated by the MOOSE
 * for the construction of sparse matrices. Reproduced here is
 * the first 4300 lines (roughly 20%) of one of the functions which
 * generates a sparse matrix for the two group transient case
 * for the multi-value integration method.
 *
 * The generated code for this case builds the matrix entries
 * necessary for the interior of a rectangular mesh, as well
 * as the code for the east and west borders for one step of
 * the transient algorithm. Conservation rules are embedded.
 *
 * All of the code in this file is automatically generated.
 * For the sake of this appendix many long comments have been
 * clipped. The comments embedded in the function describe
 * some of the partial symbolic variables which the matrix
 * generator is using to generate the code. Matrix generator
 * function names often precede sections of code which they
 * generate.
 *
 * While this function is quite long and tedious to examine by
 * hand, it presents no difficulties for gcc to compile, or for
 * the matrix generator to create based on the input configuration
 * files.
 *
 *
 *
 *
 * Calling arguments are as follows:
 *
 * cell_struct_%s_%s       cell structure for given constant/property
 * dimx, dimy              dimension of the map, 1..dimx,1..dimy
 *                        defines the major variables, a ring exists
 *                        around these also.
 * key                     the type of cell being defined by the
 *                        caller. Certain keys match certain EQstrs.
 * Mid, bid                Matrix id, and b vector id.
 * idx                     variable space id
 * mode                    another selector for choosing equations
 * X_st                    starting point in X vector
 * dx, dy                  delta X and delta Y for this map
 * posx, posy              starting position of grid[1][1]
 * mask                    ??
 *
 * Purpose of function:
 *
 * Generates a portion of the matrix for a given key and a given
 * geometric subregion.
 *
 * Notice the name of the function is given as:
 *
 * matrix_build_<physics_name>_<const_name>_<property_name>
 *
```

```
 * Any given basic map is partly defined by what constant structure
 * it uses and what property structure it uses.
 *
 * A basic map may also use multiple physics regiems within the map,
 * in particular for defining border conditions which differ from the
 * default border conditions.
 *
 * Remember also that the mode which is used to initialize a map
 * can also effect which physical regimes are used.
 *---------------------------------------------------------------
 */
// File created by Maple, calling parameters built as
// EQFILE:="default_mp"; PRPFILE:="default_pdef";
// CONFILE:="gr2_wcell_cdef"; MODES:=[2, 3, 4, 1];
// to debug run maple in project dir, execute the following:
// read "../../bin/eq_pp"; tracelast;
#include "simlib.h"
#define P_SIZE 12               // Property Size
int
 matrix_build_default_mp_gr2_wcell_cdef_default_pdef
    (cell_struct_gr2_wcell_cdef_default_pdef *** grid, int dimx, int dimy,
     int key, int Mid, int bid, int idx, int mode, int X_st, float dx, float dy,
     float posx, float posy, int mask) {
    int ROW_POS, COL_POS, ROWLEN;
    int x, y, i, xyindex;
    int ierr;
    double VALUE;
    double weight[16];
    int col[16];
    int cons_method;
    double con, cons, fx, fy;
    double reject_fr;
    ROWLEN = dimx + 2;
    int PHIs[MAX_R];
    double PHIsx[MAX_R];
    double PHIsy[MAX_R];
    int PHI[MAX_R];
    double fr[MAX_R];
    double unfr;
    cell_struct_gr2_wcell_cdef_default_pdef *cps[MAX_R], *cp[MAX_R];
    void *ocp;
    PHI[0] = -1;
    PHIs[0] = -1;
    fr[0] = 0;
    cp[0] = NULL;
    cps[0] = NULL;
    unfr = 0;
    ocp = NULL;
    double SUP[1000], t[1000];  // temporary storage for optimization
    fx = 0;
    fy = 0;
    con = 0;
    cons = 0;
    VALUE = 0;
    COL_POS = 0;
    ierr = 0;
    i = 0;
    xyindex = 0;
    con = 0;
    SUP[0] = 0;
    t[0] = 0;
    if (grid != NULL) {

        if (mode == 2) {
```

```
/*do the interior setup (easy)*/
            for (y = 1; y <= dimy; y++)
                for (x = 1; x <= dimx; x++) {
                    if (grid[x][y]->cell_id == key) {
//*******************Call made to gen:-mat_block()
// *************** call made to pre_supsrc
                        SUP[1] =
                            rs_rd_pt(3, rs_prop_T, posx + dx * (x + -.5),
                                     posy + dy * (y + -.5), 1, 1);
                        SUP[2] =
                            rs_rd_pt(3, rs_prop_T1, posx + dx * (x + -.5),
                                     posy + dy * (y + -.5), 1, 1);
                        SUP[3] =
                            rs_rd_pt(3, rs_prop_T2, posx + dx * (x + -.5),
                                     posy + dy * (y + -.5), 1, 1);
                        SUP[4] =
                            rs_rd_pt(3, rs_prop_T, posx + dx * (x + -.5),
                                     posy + dy * (y + -.5), 2, 1);
                        SUP[5] =
                            rs_rd_pt(3, rs_prop_T1, posx + dx * (x + -.5),
                                     posy + dy * (y + -.5), 2, 1);
                        SUP[6] =
                            rs_rd_pt(3, rs_prop_T2, posx + dx * (x + -.5),
                                     posy + dy * (y + -.5), 2, 1);
                        SUP[7] =
                            rs_rd_pt(3, rs_prop_C, posx + dx * (x + -.5),
                                     posy + dy * (y + -.5), 1, 1);
                        SUP[8] =
                            rs_rd_pt(3, rs_prop_C, posx + dx * (x + -.5),
                                     posy + dy * (y + -.5), 2, 1);
                        SUP[9] =
                            rs_rd_pt(3, rs_prop_C, posx + dx * (x + -.5),
                                     posy + dy * (y + -.5), 3, 1);
                        SUP[10] =
                            rs_rd_pt(3, rs_prop_C, posx + dx * (x + -.5),
                                     posy + dy * (y + -.5), 4, 1);
                        SUP[11] =
                            rs_rd_pt(3, rs_prop_C, posx + dx * (x + -.5),
                                     posy + dy * (y + -.5), 5, 1);
                        SUP[12] =
                            rs_rd_pt(3, rs_prop_C, posx + dx * (x + -.5),
                                     posy + dy * (y + -.5), 6, 1);
//PDE [T[1] = SUP[1]+G_h/G_ho*SUP[2]+G_h^2/G_ho^2*SUP[3]+.6666666667*G_h*(V[1]*(1/3*(T[e][1]/(1
//PDE_vars [T[1], T1[1], T2[1], T[2], T1[2], T2[2], C[1], C[2], C[3], C[4], C[5], C[6]]
/*****************/
// var[1]=1
// cof[1]=-.6666666667*G_h*(V[1]*G_subcrit-1.*(G_h/G_ho*SUP[2]+2.*G_h^2/G_ho^2*SUP[3])/G_h)-1.*
// var[2]=C[6]
// cof[2]=-2.993333333*G_h*V[1]*pChi[1]
// var[3]=T[2]
// cof[3]=-.6666666667*G_h*V[1]*(.993*grid[x+0][y+0]->c->Chi[0]*fcal[2]*grid[x+0][y+0]->c->nu_S
// var[4]=C[5]
// cof[4]=-1.346666667*G_h*V[1]*pChi[1]
// var[5]=C[1]
// cof[5]=-.1220000000e-1*G_h*V[1]*pChi[1]
// var[6]=C[2]
// cof[6]=-.3053333333e-1*G_h*V[1]*pChi[1]
// var[7]=C[3]
// cof[7]=-.1113333333*G_h*V[1]*pChi[1]
// var[8]=C[4]
// cof[8]=-.2986666667*G_h*V[1]*pChi[1]
// var[9]=T[e][1]
// cof[9]=-.2222222222*G_h*V[1]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[0]+.5000000000*grid[x+
```

```
// var[10]=T[w][1]
// cof[10]=-.2222222222*G_h*V[1]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[0]+.5000000000*grid[x
// var[11]=T[n][1]
// cof[11]=-.2222222222*G_h*V[1]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[0]+.5000000000*grid[x
// var[12]=T[s][1]
// cof[12]=-.2222222222*G_h*V[1]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[0]+.5000000000*grid[x
// diag=T[1]
// diagmult=1-.6666666667*G_h*V[1]*(-1/3/(1/2*'grid[x+0][y+0]->c->Sigma_tr[0]'+1/2*'grid[x+1][y
/******************/
// Row for EQ T[1] = SUP[1]+G_h/G_ho*SUP[2]+G_h^2/G_ho^2*SUP[3]+.6666666667*G_h*(V[1]*(1/3*(T[e
// Variables represented by this insertion:
//[1, C[6], T[2], C[5], C[1], C[2], C[3], C[4], T[e][1], T[w][1], T[n][1], T[s][1]]
// The diagonal element is set to:
// 1-.6666666667*G_h*V[1]*(-1/3/(1/2*'grid[x+0][y+0]->c->Sigma_tr[0]'+1/2*'grid[x+1][y+0]->c->S
                        ROW_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
// Zero mat structure  [C[6], T[2], T1[2], T2[2]]
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 2 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 7 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        t[4] = 1. / (dx * dx);
                        t[13] = 1. / (dy * dy);
                        VALUE =
                            1 -
                            .6666666667 * G_h * V[1] * (1. /
                                                        (grid[x + 0][y + 0]->
                                                         c->Sigma_tr[0] +
                                                         grid[x + 1][y +
                                                                     0]->c->
                                                         Sigma_tr[0]) * t[4] *
                                                        -2. * 1. / 3. +
                                                        1. /
                                                        (grid[x + 0][y + 0]->
                                                         c->Sigma_tr[0] +
                                                         grid[x + -1][y +
                                                                      0]->c->
                                                         Sigma_tr[0]) * t[4] *
                                                        -2. * 1. / 3. +
                                                        1. /
                                                        (grid[x + 0][y + 0]->
                                                         c->Sigma_tr[0] +
                                                         grid[x + 0][y +
                                                                     -1]->c->
```

```
                                    Sigma_tr[0]) *
                                   t[13] * -2. * 1. /
                                   3. +
                                   1. /
                                   (grid[x + 0][y + 0]->
                                    c->Sigma_tr[0] +
                                    grid[x + 0][y +
                                                1]->c->
                                    Sigma_tr[0]) *
                                   t[13] * -2. * 1. /
                                   3. - grid[x + 0][y +
                                                    0]->
                                   c->Sigma_r[0] + .993 * grid[x + 0]
                                   [y +
                                    0]->c->Chi[0] *
                                   fcal[2] * grid[x +
                                                  0][y +
                                                     0]->c->
                                   nu_Sigma_f[0]);
matrixdr_ADD(ROW_POS, VALUE);
COL_POS = X_st + 5 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE = -2.993333333 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.6666666667 * G_h * V[1] * (.993 *
                                 grid[x + 0][y +
                                             0]->c->
                                 Chi[0] * fcal[2] *
                                 grid[x + 0][y +
                                             0]->c->
                                 nu_Sigma_f[1] +
                                 grid[x + 0][y + 0]->c->Sigma_s[1][0]);
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + 4 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE = -1.346666667 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
VALUE = -.1220000000e-1 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + 1 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE = -.3053333333e-1 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + 2 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE = -.1113333333 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + 3 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE = -.2986666667 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + P_SIZE + 6 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.2222222222 * G_h * V[1] * 1. / (.5000000000 *
                                      grid[x + 0][y +
                                                  0]->
                                      c->Sigma_tr[0] +
                                      .5000000000 *
                                      grid[x + 1][y +
                                                  0]->
                                      c->Sigma_tr[0]) * 1. / (dx * dx);
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + -P_SIZE + 6 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.2222222222 * G_h * V[1] * 1. / (.5000000000 *
                                      grid[x + 0][y +
```

```
                                                        0]->
                                                    c->Sigma_tr[0] +
                                                    .5000000000 *
                                                    grid[x + -1][y +
                                                                 0]->
                                                    c->Sigma_tr[0]) * 1. / (dx * dx);
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS =
                            X_st + -ROWLEN * P_SIZE + 6 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        VALUE =
                            -.2222222222 * G_h * V[1] * 1. / (.5000000000 *
                                                    grid[x + 0][y +
                                                                0]->c->Sigma_tr[0] +
                                                    .5000000000 * grid[x + 0][y +
                                                                           -1]->
                                                    c->Sigma_tr[0]) * 1. / (dy * dy);
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS =
                            X_st + ROWLEN * P_SIZE + 6 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        VALUE =
                            -.2222222222 * G_h * V[1] * 1. / (.5000000000 *
                                                    grid[x + 0][y +
                                                                0]->c->Sigma_tr[0] +
                                                    .5000000000 * grid[x + 0][y +
                                                                           1]->c->
                                                    Sigma_tr[0]) * 1. / (dy * dy);
                        matrixdr_ADD(COL_POS, VALUE);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        t[4] = G_h * 1. / G_ho * SUP[2];
                        t[5] = G_h * G_h;
                        t[6] = G_ho * G_ho;
                        t[9] = t[5] * 1. / t[6] * SUP[3];
                        VALUE =
                            .6666666667 * G_h * (V[1] * G_subcrit -
                                                 1. * (t[4] +
                                                       2. * t[9]) * 1. /
                                                 G_h) + 1. * SUP[1] + 1. * t[4] + 1. * t[9];
                        vectdr_wr(bid, ROW_POS, VALUE);
/*****************/
// var[1]=1
// cof[1]=-1.*G_h/G_ho*SUP[2]-2.*G_h^2/G_ho^2*SUP[3]-1.*G_h*(V[1]*G_subcrit-1.*(G_h/G_ho*SUP[2]
// var[2]=C[6]
// cof[2]=-4.49*G_h*V[1]*pChi[1]
// var[3]=T[2]
// cof[3]=-1.*G_h*V[1]*(.993*grid[x+0][y+0]->c->Chi[0]*fcal[2]*grid[x+0][y+0]->c->nu_Sigma_f[1]
// var[4]=T[1]
// cof[4]=-1.*G_h*V[1]*(-.3333333333/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[0]+.5000000000*gr
// var[5]=C[5]
// cof[5]=-2.02*G_h*V[1]*pChi[1]
// var[6]=C[1]
// cof[6]=-.183e-1*G_h*V[1]*pChi[1]
// var[7]=C[2]
// cof[7]=-.458e-1*G_h*V[1]*pChi[1]
// var[8]=C[3]
// cof[8]=-.167*G_h*V[1]*pChi[1]
// var[9]=C[4]
// cof[9]=-.448*G_h*V[1]*pChi[1]
// var[10]=T[e][1]
// cof[10]=-.3333333333*G_h*V[1]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[0]+.5000000000*grid[x
// var[11]=T[w][1]
// cof[11]=-.3333333333*G_h*V[1]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[0]+.5000000000*grid[x
// var[12]=T[n][1]
// cof[12]=-.3333333333*G_h*V[1]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[0]+.5000000000*grid[x
// var[13]=T[s][1]
```

```
// cof[13]=-.3333333333*G_h*V[1]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[0]+.5000000000*grid[x
// diag=T1[1]
// diagmult=1
/*****************/
// Row for EQ T1[1] = G_h/G_ho*SUP[2]+2*G_h^2/G_ho^2*SUP[3]+G_h*(V[1]*(1/3*(T[e][1]/(1/2*Sigma_
// Variables represented by this insertion:
//[1, C[6], T[2], T[1], C[5], C[1], C[2], C[3], C[4], T[e][1], T[w][1], T[n][1], T[s][1]]
// The diagonal element is set to:
// 1
                        ROW_POS = X_st + 8 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
// Zero mat structure  [C[6], T[2], T1[2], T2[2]]
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 2 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 7 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        VALUE = 1;
                        matrixdr_ADD(ROW_POS, VALUE);
                        COL_POS = X_st + 5 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        VALUE = -4.49 * G_h * V[1] * pChi[1];
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS = X_st + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        VALUE =
                            -1. * G_h * V[1] * (.993 *
                                                grid[x + 0][y +
                                                            0]->c->Chi[0] *
                                                fcal[2] * grid[x + 0][y +
                                                                      0]->c->
                                                nu_Sigma_f[1] + grid[x + 0][y +
                                                                            0]->c->
                                                Sigma_s[1][0]);
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                        t[2] = .5000000000 * grid[x + 0][y + 0]->c->Sigma_tr[0];
                        t[6] = 1. / (dx * dx);
                        t[17] = 1. / (dy * dy);
                        VALUE =
                            -1. * G_h * V[1] * (-.3333333333 * 1. /
                                                (t[2] +
                                                 .5000000000 * grid[x + 1][y +
                                                                          0]->
                                                 c->Sigma_tr[0]) * t[6] -
                                                .3333333333 * 1. / (t[2] +
                                                                    .5000000000 * grid[x + -1]
```

```
                                                    [y +
                                                     0]->c->Sigma_tr[0]) *
                                    t[6] - .3333333333 * 1. / (t[2] +
                                                               .5000000000 *
                                                               grid[x + 0]
                                                               [y +
                                                                -1]->c->
                                                               Sigma_tr[0])
                                    * t[17] - .3333333333 * 1. / (t[2] +
                                                                  .5000000000 *
                                                                  grid[x + 0]
                                                                  [y +
                                                                   1]->c->
                                                                  Sigma_tr[0]) *
                                    t[17] - 1. * grid[x + 0][y +
                                                             0]->c->Sigma_r[0] +
                                    .993 * grid[x + 0][y +
                                                       0]->c->Chi[0] * fcal[2] *
                                    grid[x + 0][y + 0]->c->nu_Sigma_f[0]);
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + 4 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE = -2.02 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
VALUE = -.183e-1 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + 1 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE = -.458e-1 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + 2 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE = -.167 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + 3 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE = -.448 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + P_SIZE + 6 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.3333333333 * G_h * V[1] * 1. / (.5000000000 *
                                      grid[x + 0][y +
                                                  0]->
                                      c->Sigma_tr[0] +
                                      .5000000000 *
                                      grid[x + 1][y +
                                                  0]->
                                      c->Sigma_tr[0]) * 1. / (dx * dx);
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + -P_SIZE + 6 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.3333333333 * G_h * V[1] * 1. / (.5000000000 *
                                      grid[x + 0][y +
                                                  0]->
                                      c->Sigma_tr[0] +
                                      .5000000000 *
                                      grid[x + -1][y +
                                                   0]->
                                      c->Sigma_tr[0]) * 1. / (dx * dx);
matrixdr_ADD(COL_POS, VALUE);
COL_POS =
    X_st + -ROWLEN * P_SIZE + 6 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.3333333333 * G_h * V[1] * 1. / (.5000000000 *
                                      grid[x + 0][y +
                                                  0]->c->Sigma_tr[0] +
                                      .5000000000 * grid[x + 0][y +
```

```
                                                                    -1]->
                                                  c->Sigma_tr[0]) * 1. / (dy * dy);
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS =
                            X_st + ROWLEN * P_SIZE + 6 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        VALUE =
                            -.3333333333 * G_h * V[1] * 1. / (.5000000000 *
                                                  grid[x + 0][y +
                                                             0]->c->Sigma_tr[0] +
                                                  .5000000000 * grid[x + 0][y +
                                                                        1]->c->
                                                  Sigma_tr[0]) * 1. / (dy * dy);
                        matrixdr_ADD(COL_POS, VALUE);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        t[3] = G_h * 1. / G_ho * SUP[2];
                        t[5] = G_h * G_h;
                        t[6] = G_ho * G_ho;
                        t[10] = 2. * t[5] * 1. / t[6] * SUP[3];
                        VALUE =
                            1. * t[3] + t[10] + 1. * G_h * (V[1] * G_subcrit -
                                                  1. * (t[3] + t[10]) * 1. / G_h);
                        vectdr_wr(bid, ROW_POS, VALUE);
/******************/
// var[1]=1
// cof[1]=-1.*G_h^2/G_ho^2*SUP[3]-.3333333333*G_h*(V[1]*G_subcrit-1.*(G_h/G_ho*SUP[2]+2.*G_h^2/
// var[2]=C[6]
// cof[2]=-1.496666667*G_h*V[1]*pChi[1]
// var[3]=T[2]
// cof[3]=-.3333333333*G_h*V[1]*(.993*grid[x+0][y+0]->c->Chi[0]*fcal[2]*grid[x+0][y+0]->c->nu_S
// var[4]=T[1]
// cof[4]=-.3333333333*G_h*V[1]*(-.3333333333/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[0]+.5000
// var[5]=C[5]
// cof[5]=-.6733333333*G_h*V[1]*pChi[1]
// var[6]=C[1]
// cof[6]=-.6099999999e-2*G_h*V[1]*pChi[1]
// var[7]=C[2]
// cof[7]=-.1526666667e-1*G_h*V[1]*pChi[1]
// var[8]=C[3]
// cof[8]=-.5566666666e-1*G_h*V[1]*pChi[1]
// var[9]=C[4]
// cof[9]=-.1493333333*G_h*V[1]*pChi[1]
// var[10]=T[e][1]
// cof[10]=-.1111111111*G_h*V[1]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[0]+.5000000000*grid[x
// var[11]=T[w][1]
// cof[11]=-.1111111111*G_h*V[1]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[0]+.5000000000*grid[x
// var[12]=T[n][1]
// cof[12]=-.1111111111*G_h*V[1]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[0]+.5000000000*grid[x
// var[13]=T[s][1]
// cof[13]=-.1111111111*G_h*V[1]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[0]+.5000000000*grid[x
// diag=T2[1]
// diagmult=1
/******************/
// Row for EQ T2[1] = G_h^2/G_ho^2*SUP[3]+.3333333333*G_h*(V[1]*(1/3*(T[e][1]/(1/2*Sigma_tr[1]+
// Variables represented by this insertion:
//[1, C[6], T[2], T[1], C[5], C[1], C[2], C[3], C[4], T[e][1], T[w][1], T[n][1], T[s][1]]
// The diagonal element is set to:
// 1
                        ROW_POS = X_st + 10 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
// Zero mat structure  [C[6], T[2], T1[2], T2[2]]
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
```

```
COL_POS = X_st + 2 + (x + y * ROWLEN) * P_SIZE;
matrixdr_ADDzero(COL_POS);
COL_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
matrixdr_ADDzero(COL_POS);
COL_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
matrixdr_ADDzero(COL_POS);
COL_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
matrixdr_ADDzero(COL_POS);
COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
matrixdr_ADDzero(COL_POS);
COL_POS = X_st + 7 + (x + y * ROWLEN) * P_SIZE;
matrixdr_ADDzero(COL_POS);
COL_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
matrixdr_ADDzero(COL_POS);
COL_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
matrixdr_ADDzero(COL_POS);
COL_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
matrixdr_ADDzero(COL_POS);
COL_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
matrixdr_ADDzero(COL_POS);
VALUE = 1;
matrixdr_ADD(ROW_POS, VALUE);
COL_POS = X_st + 5 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE = -1.496666667 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.3333333333 * G_h * V[1] * (.993 *
                                 grid[x + 0][y +
                                             0]->c->
                                 Chi[0] * fcal[2] *
                                 grid[x + 0][y +
                                             0]->c->
                                 nu_Sigma_f[1] +
                                 grid[x + 0][y + 0]->c->Sigma_s[1][0]);
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
t[2] = .5000000000 * grid[x + 0][y + 0]->c->Sigma_tr[0];
t[6] = 1. / (dx * dx);
t[17] = 1. / (dy * dy);
VALUE =
    -.3333333333 * G_h * V[1] * (-.3333333333 * 1. /
                                 (t[2] +
                                  .5000000000 *
                                  grid[x + 1][y +
                                              0]->c->
                                  Sigma_tr[0]) *
                                 t[6] -
                                 .3333333333 * 1. /
                                 (t[2] + .5000000000 * grid[x + -1]
                                  [y +
                                   0]->c->
                                  Sigma_tr[0]) *
                                 t[6] -
                                 .3333333333 * 1. /
                                 (t[2] + .5000000000 * grid[x + 0]
                                  [y +
                                   -1]->c->
                                  Sigma_tr[0]) *
                                 t[17] -
                                 .3333333333 * 1. /
                                 (t[2] + .5000000000 * grid[x + 0]
                                  [y +
                                   1]->c->
```

```
                                    Sigma_tr[0]) * t[17] - 1. * grid[x +
                                                                     0]
                                    [y + 0]->c->Sigma_r[0] +
                                    .993 * grid[x + 0][y +
                                                       0]->c->Chi[0] *
                                    fcal[2] * grid[x + 0][y +
                                                         0]->c->
                                    nu_Sigma_f[0]);
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 4 + P_SIZE * x;
VALUE = -.6733333333 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
VALUE = -.6099999999e-2 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 1 + P_SIZE * x;
VALUE = -.1526666667e-1 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 2 + P_SIZE * x;
VALUE = -.5566666666e-1 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 3 + P_SIZE * x;
VALUE = -.1493333333 * G_h * V[1] * pChi[1];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + P_SIZE + 6 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.1111111111 * G_h * V[1] * 1. / (.5000000000 *
                                      grid[x + 0][y +
                                                  0]->
                                      c->Sigma_tr[0] +
                                      .5000000000 *
                                      grid[x + 1][y +
                                                  0]->
                                      c->Sigma_tr[0]) * 1. / (dx * dx);
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + -P_SIZE + 6 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.1111111111 * G_h * V[1] * 1. / (.5000000000 *
                                      grid[x + 0][y +
                                                  0]->
                                      c->Sigma_tr[0] +
                                      .5000000000 *
                                      grid[x + -1][y +
                                                   0]->
                                      c->Sigma_tr[0]) * 1. / (dx * dx);
matrixdr_ADD(COL_POS, VALUE);
COL_POS =
    X_st + -ROWLEN * P_SIZE + 6 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.1111111111 * G_h * V[1] * 1. / (.5000000000 *
                                      grid[x + 0][y +
                                                  0]->c->Sigma_tr[0] +
                                      .5000000000 * grid[x + 0][y +
                                                               -1]->
                                      c->Sigma_tr[0]) * 1. / (dy * dy);
matrixdr_ADD(COL_POS, VALUE);
COL_POS =
    X_st + ROWLEN * P_SIZE + 6 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.1111111111 * G_h * V[1] * 1. / (.5000000000 *
                                      grid[x + 0][y +
                                                  0]->c->Sigma_tr[0] +
                                      .5000000000 * grid[x + 0][y +
                                                               1]->c->
```

```
                                                        Sigma_tr[0]) * 1. / (dy * dy);
                        matrixdr_ADD(COL_POS, VALUE);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        t[1] = G_h * G_h;
                        t[2] = G_ho * G_ho;
                        t[5] = t[1] * 1. / t[2] * SUP[3];
                        VALUE =
                            1. * t[5] +
                            .3333333333 * G_h * (V[1] * G_subcrit -
                                                 1. * (G_h * 1. / G_ho *
                                                       SUP[2] + 2. * t[5]) * 1. / G_h);
                        vectdr_wr(bid, ROW_POS, VALUE);
/*****************/
// var[1]=1
// cof[1]=-.6666666667*G_h*(V[2]*G_subcrit-1.*(G_h/G_ho*SUP[5]+2.*G_h^2/G_ho^2*SUP[6])/G_h)-1.*
// var[2]=C[6]
// cof[2]=-2.993333333*G_h*V[2]*pChi[2]
// var[3]=T[1]
// cof[3]=-.6666666667*G_h*V[2]*(grid[x+0][y+0]->c->Sigma_s[0][1]+.993*grid[x+0][y+0]->c->Chi[1
// var[4]=C[5]
// cof[4]=-1.346666667*G_h*V[2]*pChi[2]
// var[5]=C[1]
// cof[5]=-.1220000000e-1*G_h*V[2]*pChi[2]
// var[6]=C[2]
// cof[6]=-.3053333333e-1*G_h*V[2]*pChi[2]
// var[7]=C[3]
// cof[7]=-.1113333333*G_h*V[2]*pChi[2]
// var[8]=C[4]
// cof[8]=-.2986666667*G_h*V[2]*pChi[2]
// var[9]=T[e][2]
// cof[9]=-.2222222222*G_h*V[2]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[1]+.5000000000*grid[x+
// var[10]=T[w][2]
// cof[10]=-.2222222222*G_h*V[2]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[1]+.5000000000*grid[x
// var[11]=T[n][2]
// cof[11]=-.2222222222*G_h*V[2]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[1]+.5000000000*grid[x
// var[12]=T[s][2]
// cof[12]=-.2222222222*G_h*V[2]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[1]+.5000000000*grid[x
// diag=T[2]
// diagmult=1-.6666666667*G_h*V[2]*(-1/3/(1/2*'grid[x+0][y+0]->c->Sigma_tr[1]'+1/2*'grid[x+1][y
/*****************/
// Row for EQ T[2] = SUP[4]+G_h/G_ho*SUP[5]+G_h^2/G_ho^2*SUP[6]+.6666666667*G_h*(V[2]*(1/3*(T[e
// Variables represented by this insertion:
//[1, C[6], T[1], C[5], C[1], C[2], C[3], C[4], T[e][2], T[w][2], T[n][2], T[s][2]]
// The diagonal element is set to:
// 1-.6666666667*G_h*V[2]*(-1/3/(1/2*'grid[x+0][y+0]->c->Sigma_tr[1]'+1/2*'grid[x+1][y+0]->c->S
                        ROW_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
// Zero mat structure  [C[6], T[2], T1[2], T2[2]]
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 1;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 3;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 4;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
                        matrixdr_ADDzero(COL_POS);
```

```
COL_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
matrixdr_ADDzero(COL_POS);
COL_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
matrixdr_ADDzero(COL_POS);
COL_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
matrixdr_ADDzero(COL_POS);
COL_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
matrixdr_ADDzero(COL_POS);
t[4] = 1. / (dx * dx);
t[13] = 1. / (dy * dy);
VALUE =
    1 -
    .6666666667 * G_h * V[2] * (1. /
                                (grid[x + 0][y + 0]->
                                 c->Sigma_tr[1] +
                                 grid[x + 1][y +
                                             0]->c->
                                 Sigma_tr[1]) * t[4] *
                                -2. * 1. / 3. +
                                1. /
                                (grid[x + 0][y + 0]->
                                 c->Sigma_tr[1] +
                                 grid[x + -1][y +
                                              0]->c->
                                 Sigma_tr[1]) * t[4] *
                                -2. * 1. / 3. +
                                1. /
                                (grid[x + 0][y + 0]->
                                 c->Sigma_tr[1] +
                                 grid[x + 0][y +
                                             -1]->c->
                                 Sigma_tr[1]) *
                                t[13] * -2. * 1. /
                                3. +
                                1. /
                                (grid[x + 0][y + 0]->
                                 c->Sigma_tr[1] +
                                 grid[x + 0][y +
                                             1]->c->
                                 Sigma_tr[1]) *
                                t[13] * -2. * 1. /
                                3. - grid[x + 0][y +
                                                 0]->
                                c->Sigma_r[1] + .993 * grid[x + 0]
                                [y +
                                 0]->c->Chi[1] *
                                fcal[2] * grid[x +
                                               0][y +
                                                  0]->c->
                                nu_Sigma_f[1]);
matrixdr_ADD(ROW_POS, VALUE);
COL_POS = X_st + 5 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE = -2.993333333 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
VALUE =
    -.6666666667 * G_h * V[2] *
    (grid[x + 0][y + 0]->c->Sigma_s[0][1] +
     .993 * grid[x + 0][y +
                        0]->c->Chi[1] * fcal[2] *
     grid[x + 0][y + 0]->c->nu_Sigma_f[0]);
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 4 + P_SIZE * x;
VALUE = -1.346666667 * G_h * V[2] * pChi[2];
```

```
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
VALUE = -.1220000000e-1 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 1 + P_SIZE * x;
VALUE = -.3053333333e-1 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 2 + P_SIZE * x;
VALUE = -.1113333333 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 3 + P_SIZE * x;
VALUE = -.2986666667 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + P_SIZE + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.2222222222 * G_h * V[2] * 1. / (.5000000000 *
                                      grid[x + 0][y +
                                                  0]->
                                      c->Sigma_tr[1] +
                                      .5000000000 *
                                      grid[x + 1][y +
                                                  0]->
                                      c->Sigma_tr[1]) * 1. / (dx * dx);
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + -P_SIZE + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.2222222222 * G_h * V[2] * 1. / (.5000000000 *
                                      grid[x + 0][y +
                                                  0]->
                                      c->Sigma_tr[1] +
                                      .5000000000 *
                                      grid[x + -1][y +
                                                   0]->
                                      c->Sigma_tr[1]) * 1. / (dx * dx);
matrixdr_ADD(COL_POS, VALUE);
COL_POS =
    X_st + -ROWLEN * P_SIZE + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.2222222222 * G_h * V[2] * 1. / (.5000000000 *
                                      grid[x + 0][y +
                                                  0]->c->Sigma_tr[1] +
                                      .5000000000 * grid[x + 0][y +
                                                               -1]->
                                      c->Sigma_tr[1]) * 1. / (dy * dy);
matrixdr_ADD(COL_POS, VALUE);
COL_POS =
    X_st + ROWLEN * P_SIZE + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.2222222222 * G_h * V[2] * 1. / (.5000000000 *
                                      grid[x + 0][y +
                                                  0]->c->Sigma_tr[1] +
                                      .5000000000 * grid[x + 0][y +
                                                               1]->c->
                                      Sigma_tr[1]) * 1. / (dy * dy);
matrixdr_ADD(COL_POS, VALUE);
matrixdr_CLOSE(Mid, ROW_POS);
t[4] = G_h * 1. / G_ho * SUP[5];
t[5] = G_h * G_h;
t[6] = G_ho * G_ho;
t[9] = t[5] * 1. / t[6] * SUP[6];
VALUE =
    .6666666667 * G_h * (V[2] * G_subcrit -
                         1. * (t[4] +
                               2. * t[9]) * 1. /
```

```
                                                    G_h) + 1. * SUP[4] + 1. * t[4] + 1. * t[9];
                        vectdr_wr(bid, ROW_POS, VALUE);
/*****************/
// var[1]=1
// cof[1]=-1.*G_h/G_ho*SUP[5]-2.*G_h^2/G_ho^2*SUP[6]-1.*G_h*(V[2]*G_subcrit-1.*(G_h/G_ho*SUP[5]
// var[2]=C[6]
// cof[2]=-4.49*G_h*V[2]*pChi[2]
// var[3]=T[2]
// cof[3]=-1.*G_h*V[2]*(-.3333333333/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[1]+.5000000000*gr
// var[4]=T[1]
// cof[4]=-1.*G_h*V[2]*(grid[x+0][y+0]->c->Sigma_s[0][1]+.993*grid[x+0][y+0]->c->Chi[1]*fcal[2]
// var[5]=C[5]
// cof[5]=-2.02*G_h*V[2]*pChi[2]
// var[6]=C[1]
// cof[6]=-.183e-1*G_h*V[2]*pChi[2]
// var[7]=C[2]
// cof[7]=-.458e-1*G_h*V[2]*pChi[2]
// var[8]=C[3]
// cof[8]=-.167*G_h*V[2]*pChi[2]
// var[9]=C[4]
// cof[9]=-.448*G_h*V[2]*pChi[2]
// var[10]=T[e][2]
// cof[10]=-.3333333333*G_h*V[2]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[1]+.5000000000*grid[x
// var[11]=T[w][2]
// cof[11]=-.3333333333*G_h*V[2]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[1]+.5000000000*grid[x
// var[12]=T[n][2]
// cof[12]=-.3333333333*G_h*V[2]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[1]+.5000000000*grid[x
// var[13]=T[s][2]
// cof[13]=-.3333333333*G_h*V[2]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[1]+.5000000000*grid[x
// diag=T1[2]
// diagmult=1
/*****************/
// Row for EQ T1[2] = G_h/G_ho*SUP[5]+2*G_h^2/G_ho^2*SUP[6]+G_h*(V[2]*(1/3*(T[e][2]/(1/2*Sigma_
// Variables represented by this insertion:
//[1, C[6], T[2], T[1], C[5], C[1], C[2], C[3], C[4], T[e][2], T[w][2], T[n][2], T[s][2]]
// The diagonal element is set to:
// 1
                        ROW_POS = X_st + 9 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
// Zero mat structure  [C[6], T[2], T1[2], T2[2]]
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 1;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 3;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 4;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
```

```
VALUE = 1;
matrixdr_ADD(ROW_POS, VALUE);
COL_POS = X_st + 5 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE = -4.49 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
t[2] = .5000000000 * grid[x + 0][y + 0]->c->Sigma_tr[1];
t[6] = 1. / (dx * dx);
t[17] = 1. / (dy * dy);
VALUE =
    -1. * G_h * V[2] * (-.3333333333 * 1. /
                        (t[2] +
                         .5000000000 * grid[x + 1][y +
                                                   0]->
                         c->Sigma_tr[1]) * t[6] -
                        .3333333333 * 1. / (t[2] +
                                            .5000000000 * grid[x + -1]
                                            [y +
                                             0]->c->Sigma_tr[1]) *
                        t[6] - .3333333333 * 1. / (t[2] +
                                                   .5000000000 *
                                                   grid[x + 0]
                                                   [y +
                                                    -1]->c->
                                                   Sigma_tr[1])
                        * t[17] - .3333333333 * 1. / (t[2] +
                                                      .5000000000 *
                                                      grid[x + 0]
                                                      [y +
                                                       1]->c->
                                                      Sigma_tr[1]) *
                        t[17] - 1. * grid[x + 0][y +
                                                 0]->c->Sigma_r[1] +
                        .993 * grid[x + 0][y +
                                           0]->c->Chi[1] * fcal[2] *
                        grid[x + 0][y + 0]->c->nu_Sigma_f[1]);
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
VALUE =
    -1. * G_h * V[2] *
    (grid[x + 0][y + 0]->c->Sigma_s[0][1] +
     .993 * grid[x + 0][y +
                        0]->c->Chi[1] * fcal[2] *
     grid[x + 0][y + 0]->c->nu_Sigma_f[0]);
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 4 + P_SIZE * x;
VALUE = -2.02 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
VALUE = -.183e-1 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 1 + P_SIZE * x;
VALUE = -.458e-1 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 2 + P_SIZE * x;
VALUE = -.167 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 3 + P_SIZE * x;
VALUE = -.448 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + P_SIZE + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.3333333333 * G_h * V[2] * 1. / (.5000000000 *
                                      grid[x + 0][y +
```

```
                                                        0]->
                                                    c->Sigma_tr[1] +
                                                    .5000000000 *
                                                    grid[x + 1][y +
                                                        0]->
                                                    c->Sigma_tr[1]) * 1. / (dx * dx);
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS = X_st + -P_SIZE + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        VALUE =
                            -.3333333333 * G_h * V[2] * 1. / (.5000000000 *
                                                    grid[x + 0][y +
                                                        0]->
                                                    c->Sigma_tr[1] +
                                                    .5000000000 *
                                                    grid[x + -1][y +
                                                        0]->
                                                    c->Sigma_tr[1]) * 1. / (dx * dx);
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS =
                            X_st + -ROWLEN * P_SIZE + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        VALUE =
                            -.3333333333 * G_h * V[2] * 1. / (.5000000000 *
                                                    grid[x + 0][y +
                                                        0]->c->Sigma_tr[1] +
                                                    .5000000000 * grid[x + 0][y +
                                                                -1]->
                                                    c->Sigma_tr[1]) * 1. / (dy * dy);
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS =
                            X_st + ROWLEN * P_SIZE + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        VALUE =
                            -.3333333333 * G_h * V[2] * 1. / (.5000000000 *
                                                    grid[x + 0][y +
                                                        0]->c->Sigma_tr[1] +
                                                    .5000000000 * grid[x + 0][y +
                                                                1]->c->
                                                    Sigma_tr[1]) * 1. / (dy * dy);
                        matrixdr_ADD(COL_POS, VALUE);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        t[3] = G_h * 1. / G_ho * SUP[5];
                        t[5] = G_h * G_h;
                        t[6] = G_ho * G_ho;
                        t[10] = 2. * t[5] * 1. / t[6] * SUP[6];
                        VALUE =
                            1. * t[3] + t[10] + 1. * G_h * (V[2] * G_subcrit -
                                                1. * (t[3] + t[10]) * 1. / G_h);
                        vectdr_wr(bid, ROW_POS, VALUE);
/*****************/
// var[1]=1
// cof[1]=-1.*G_h^2/G_ho^2*SUP[6]-.3333333333*G_h*(V[2]*G_subcrit-1.*(G_h/G_ho*SUP[5]+2.*G_h^2/
// var[2]=C[6]
// cof[2]=-1.496666667*G_h*V[2]*pChi[2]
// var[3]=T[2]
// cof[3]=-.3333333333*G_h*V[2]*(-.3333333333/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[1]+.5000
// var[4]=T[1]
// cof[4]=-.3333333333*G_h*V[2]*(grid[x+0][y+0]->c->Sigma_s[0][1]+.993*grid[x+0][y+0]->c->Chi[1
// var[5]=C[5]
// cof[5]=-.6733333333*G_h*V[2]*pChi[2]
// var[6]=C[1]
// cof[6]=-.6099999999e-2*G_h*V[2]*pChi[2]
// var[7]=C[2]
// cof[7]=-.1526666667e-1*G_h*V[2]*pChi[2]
// var[8]=C[3]
// cof[8]=-.5566666666e-1*G_h*V[2]*pChi[2]
```

```
// var[9]=C[4]
// cof[9]=-.1493333333*G_h*V[2]*pChi[2]
// var[10]=T[e][2]
// cof[10]=-.1111111111*G_h*V[2]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[1]+.5000000000*grid[x
// var[11]=T[w][2]
// cof[11]=-.1111111111*G_h*V[2]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[1]+.5000000000*grid[x
// var[12]=T[n][2]
// cof[12]=-.1111111111*G_h*V[2]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[1]+.5000000000*grid[x
// var[13]=T[s][2]
// cof[13]=-.1111111111*G_h*V[2]/(.5000000000*grid[x+0][y+0]->c->Sigma_tr[1]+.5000000000*grid[x
// diag=T2[2]
// diagmult=1
/*****************/
// Row for EQ T2[2] = G_h^2/G_ho^2*SUP[6]+.3333333333*G_h*(V[2]*(1/3*(T[e][2]/(1/2*Sigma_tr[2]+
// Variables represented by this insertion:
//[1, C[6], T[2], T[1], C[5], C[1], C[2], C[3], C[4], T[e][2], T[w][2], T[n][2], T[s][2]]
// The diagonal element is set to:
// 1
                        ROW_POS = X_st + 11 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
// Zero mat structure  [C[6], T[2], T1[2], T2[2]]
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 1;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 3;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 4;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        VALUE = 1;
                        matrixdr_ADD(ROW_POS, VALUE);
                        COL_POS = X_st + 5 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        VALUE = -1.496666667 * G_h * V[2] * pChi[2];
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                        t[2] = .5000000000 * grid[x + 0][y + 0]->c->Sigma_tr[1];
                        t[6] = 1. / (dx * dx);
                        t[17] = 1. / (dy * dy);
                        VALUE =
                            -.3333333333 * G_h * V[2] * (-.3333333333 * 1. /
                                                         (t[2] +
                                                          .5000000000 *
                                                          grid[x + 1][y +
                                                                      0]->c->
                                                          Sigma_tr[1]) *
                                                         t[6] -
                                                         .3333333333 * 1. /
                                                         (t[2] + .5000000000 * grid[x + -1]
```

```
                                        [y +
                                         0]->c->
                                        Sigma_tr[1]) *
                                       t[6] -
                                       .3333333333 * 1. /
                                       (t[2] + .5000000000 * grid[x + 0]
                                        [y +
                                         -1]->c->
                                        Sigma_tr[1]) *
                                       t[17] -
                                       .3333333333 * 1. /
                                       (t[2] + .5000000000 * grid[x + 0]
                                        [y +
                                         1]->c->
                                        Sigma_tr[1]) * t[17] - 1. * grid[x +
                                                                         0]
                                       [y + 0]->c->Sigma_r[1] +
                                       .993 * grid[x + 0][y +
                                                          0]->c->Chi[1] *
                                       fcal[2] * grid[x + 0][y +
                                                            0]->c->
                                       nu_Sigma_f[1]);
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
VALUE =
    -.3333333333 * G_h * V[2] *
    (grid[x + 0][y + 0]->c->Sigma_s[0][1] +
     .993 * grid[x + 0][y +
                        0]->c->Chi[1] * fcal[2] *
     grid[x + 0][y + 0]->c->nu_Sigma_f[0]);
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 4 + P_SIZE * x;
VALUE = -.6733333333 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
VALUE = -.6099999999e-2 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 1 + P_SIZE * x;
VALUE = -.1526666667e-1 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 2 + P_SIZE * x;
VALUE = -.5566666666e-1 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + y * ROWLEN * P_SIZE + 3 + P_SIZE * x;
VALUE = -.1493333333 * G_h * V[2] * pChi[2];
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + P_SIZE + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.1111111111 * G_h * V[2] * 1. / (.5000000000 *
                                      grid[x + 0][y +
                                                  0]->
                                      c->Sigma_tr[1] +
                                      .5000000000 *
                                      grid[x + 1][y +
                                                  0]->
                                      c->Sigma_tr[1]) * 1. / (dx * dx);
matrixdr_ADD(COL_POS, VALUE);
COL_POS = X_st + -P_SIZE + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
VALUE =
    -.1111111111 * G_h * V[2] * 1. / (.5000000000 *
                                      grid[x + 0][y +
                                                  0]->
                                      c->Sigma_tr[1] +
                                      .5000000000 *
```

```
                                                grid[x + -1][y +
                                                             0]->
                                                c->Sigma_tr[1]) * 1. / (dx * dx);
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS =
                            X_st + -ROWLEN * P_SIZE + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        VALUE =
                            -.1111111111 * G_h * V[2] * 1. / (.5000000000 *
                                                grid[x + 0][y +
                                                             0]->c->Sigma_tr[1] +
                                                .5000000000 * grid[x + 0][y +
                                                                          -1]->
                                                c->Sigma_tr[1]) * 1. / (dy * dy);
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS =
                            X_st + ROWLEN * P_SIZE + 7 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        VALUE =
                            -.1111111111 * G_h * V[2] * 1. / (.5000000000 *
                                                grid[x + 0][y +
                                                             0]->c->Sigma_tr[1] +
                                                .5000000000 * grid[x + 0][y +
                                                                          1]->c->
                                                Sigma_tr[1]) * 1. / (dy * dy);
                        matrixdr_ADD(COL_POS, VALUE);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        t[1] = G_h * G_h;
                        t[2] = G_ho * G_ho;
                        t[5] = t[1] * 1. / t[2] * SUP[6];
                        VALUE =
                            1. * t[5] +
                            .3333333333 * G_h * (V[2] * G_subcrit -
                                                 1. * (G_h * 1. / G_ho *
                                                       SUP[5] + 2. * t[5]) * 1. / G_h);
                        vectdr_wr(bid, ROW_POS, VALUE);
/******************/
// var[1]=1
// cof[1]=-1.*SUP[7]-.5*G_h*(-.183e-1*SUP[7]+fcal[2]*(.266e-3*grid[x+0][y+0]->c->nu_Sigma_f[0]*
// var[2]=T[2]
// cof[2]=-.1330e-3*G_h*fcal[2]*grid[x+0][y+0]->c->nu_Sigma_f[1]
// var[3]=T[1]
// cof[3]=-.1330e-3*G_h*fcal[2]*grid[x+0][y+0]->c->nu_Sigma_f[0]
// diag=C[1]
// diagmult=1+.915e-2*G_h
/******************/
// Row for EQ C[1] = SUP[7]+.5*G_h*(-.183e-1*C[1]+fcal[2]*(.266e-3*nu_Sigma_f[1]*T[1]+.266e-3*n
// Variables represented by this insertion:
//[1, T[2], T[1]]
// The diagonal element is set to:
// 1+.915e-2*G_h
                        ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
// Zero mat structure  [C[6], T[2], T1[2], T2[2]]
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 1;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 3;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 4;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
```

```
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        VALUE = 1 + .915e-2 * G_h;
                        matrixdr_ADD(ROW_POS, VALUE);
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                        VALUE =
                            -.1330e-3 * G_h * fcal[2] * grid[x + 0][y + 0]->c->nu_Sigma_f[1];
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                        VALUE =
                            -.1330e-3 * G_h * fcal[2] * grid[x + 0][y + 0]->c->nu_Sigma_f[0];
                        matrixdr_ADD(COL_POS, VALUE);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        VALUE =
                            1. * SUP[7] + .5 * G_h * (-.183e-1 * SUP[7] +
                                                      fcal[2] * (.266e-3 *
                                                                 grid[x + 0][y +
                                                                             0]->c->
                                                                 nu_Sigma_f[0]
                                                                 * SUP[1] + .266e-3 * grid[x +
                                                                                           0][y
                                                                                              +
                                                                                              0]->
                                                                 c->nu_Sigma_f[1]
                                                                 * SUP[4]));
                        vectdr_wr(bid, ROW_POS, VALUE);
/******************/
// var[1]=1
// cof[1]=-1.*SUP[8]-.5*G_h*(-.458e-1*SUP[8]+fcal[2]*(.1491e-2*grid[x+0][y+0]->c->nu_Sigma_f[0]
// var[2]=T[2]
// cof[2]=-.7455e-3*G_h*fcal[2]*grid[x+0][y+0]->c->nu_Sigma_f[1]
// var[3]=T[1]
// cof[3]=-.7455e-3*G_h*fcal[2]*grid[x+0][y+0]->c->nu_Sigma_f[0]
// diag=C[2]
// diagmult=1+.2290e-1*G_h
/******************/
// Row for EQ C[2] = SUP[8]+.5*G_h*(-.458e-1*C[2]+fcal[2]*(.1491e-2*nu_Sigma_f[1]*T[1]+.1491e-2
// Variables represented by this insertion:
//[1, T[2], T[1]]
// The diagonal element is set to:
// 1+.2290e-1*G_h
                        ROW_POS = X_st + y * ROWLEN * P_SIZE + 1 + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
// Zero mat structure  [C[6], T[2], T1[2], T2[2]]
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 1;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 3;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 4;
                        matrixdr_ADDzero(COL_POS);
```

```
                        COL_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        VALUE = 1 + .2290e-1 * G_h;
                        matrixdr_ADD(ROW_POS, VALUE);
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                        VALUE =
                            -.7455e-3 * G_h * fcal[2] * grid[x + 0][y + 0]->c->nu_Sigma_f[1];
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                        VALUE =
                            -.7455e-3 * G_h * fcal[2] * grid[x + 0][y + 0]->c->nu_Sigma_f[0];
                        matrixdr_ADD(COL_POS, VALUE);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        VALUE =
                            1. * SUP[8] + .5 * G_h * (-.458e-1 * SUP[8] +
                                                      fcal[2] * (.1491e-2 *
                                                                 grid[x + 0][y +
                                                                             0]->c->
                                                                 nu_Sigma_f[0]
                                                                 * SUP[1] + .1491e-2 * grid[x +
                                                                                            0]
                                                                 [y + 0]->c->nu_Sigma_f[1]
                                                                 * SUP[4]));
                        vectdr_wr(bid, ROW_POS, VALUE);
/*****************/
// var[1]=1
// cof[1]=-1.*SUP[9]-.5*G_h*(-.167*SUP[9]+fcal[2]*(.1316e-2*grid[x+0][y+0]->c->nu_Sigma_f[0]*SU
// var[2]=T[2]
// cof[2]=-.6580e-3*G_h*fcal[2]*grid[x+0][y+0]->c->nu_Sigma_f[1]
// var[3]=T[1]
// cof[3]=-.6580e-3*G_h*fcal[2]*grid[x+0][y+0]->c->nu_Sigma_f[0]
// diag=C[3]
// diagmult=1+.835e-1*G_h
/*****************/
// Row for EQ C[3] = SUP[9]+.5*G_h*(-.167*C[3]+fcal[2]*(.1316e-2*nu_Sigma_f[1]*T[1]+.1316e-2*nu
// Variables represented by this insertion:
//[1, T[2], T[1]]
// The diagonal element is set to:
// 1+.835e-1*G_h
                        ROW_POS = X_st + y * ROWLEN * P_SIZE + 2 + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
// Zero mat structure  [C[6], T[2], T1[2], T2[2]]
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
```

```
                        COL_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        VALUE = 1 + .835e-1 * G_h;
                        matrixdr_ADD(ROW_POS, VALUE);
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                        VALUE =
                            -.6580e-3 * G_h * fcal[2] * grid[x + 0][y + 0]->c->nu_Sigma_f[1];
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                        VALUE =
                            -.6580e-3 * G_h * fcal[2] * grid[x + 0][y + 0]->c->nu_Sigma_f[0];
                        matrixdr_ADD(COL_POS, VALUE);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        VALUE =
                            1. * SUP[9] + .5 * G_h * (-.167 * SUP[9] +
                                                      fcal[2] * (.1316e-2 *
                                                                 grid[x + 0][y +
                                                                             0]->c->
                                                                 nu_Sigma_f[0]
                                                                 * SUP[1] + .1316e-2 * grid[x +
                                                                                            0]
                                                                 [y + 0]->c->nu_Sigma_f[1]
                                                                 * SUP[4]));
                        vectdr_wr(bid, ROW_POS, VALUE);
/*****************/
// var[1]=1
// cof[1]=-1.*SUP[10]-.5*G_h*(-.448*SUP[10]+fcal[2]*(.2849e-2*grid[x+0][y+0]->c->nu_Sigma_f[0]*
// var[2]=T[2]
// cof[2]=-.14245e-2*G_h*fcal[2]*grid[x+0][y+0]->c->nu_Sigma_f[1]
// var[3]=T[1]
// cof[3]=-.14245e-2*G_h*fcal[2]*grid[x+0][y+0]->c->nu_Sigma_f[0]
// diag=C[4]
// diagmult=1+.2240*G_h
/*****************/
// Row for EQ C[4] = SUP[10]+.5*G_h*(-.448*C[4]+fcal[2]*(.2849e-2*nu_Sigma_f[1]*T[1]+.2849e-2*n
// Variables represented by this insertion:
//[1, T[2], T[1]]
// The diagonal element is set to:
// 1+.2240*G_h
                        ROW_POS = X_st + 3 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
// Zero mat structure  [C[6], T[2], T1[2], T2[2]]
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
```

```
                        COL_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        VALUE = 1 + .2240 * G_h;
                        matrixdr_ADD(ROW_POS, VALUE);
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                        VALUE =
                            -.14245e-2 * G_h * fcal[2] * grid[x + 0][y + 0]->c->nu_Sigma_f[1];
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                        VALUE =
                            -.14245e-2 * G_h * fcal[2] * grid[x + 0][y + 0]->c->nu_Sigma_f[0];
                        matrixdr_ADD(COL_POS, VALUE);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        VALUE =
                            1. * SUP[10] + .5 * G_h * (-.448 * SUP[10] +
                                                       fcal[2] * (.2849e-2 *
                                                                  grid[x +
                                                                       0][y +
                                                                          0]->
                                                                  c->
                                                                  nu_Sigma_f
                                                                  [0] *
                                                                  SUP[1] +
                                                                  .2849e-2 *
                                                                  grid[x +
                                                                       0][y +
                                                                          0]->
                                                                  c->nu_Sigma_f[1] * SUP[4]));
                        vectdr_wr(bid, ROW_POS, VALUE);
/*****************/
// var[1]=1
// cof[1]=-1.*SUP[11]-.5*G_h*(-2.02*SUP[11]+fcal[2]*(.896e-3*grid[x+0][y+0]->c->nu_Sigma_f[0]*S
// var[2]=T[2]
// cof[2]=-.4480e-3*G_h*fcal[2]*grid[x+0][y+0]->c->nu_Sigma_f[1]
// var[3]=T[1]
// cof[3]=-.4480e-3*G_h*fcal[2]*grid[x+0][y+0]->c->nu_Sigma_f[0]
// diag=C[5]
// diagmult=1+1.010*G_h
/*****************/
// Row for EQ C[5] = SUP[11]+.5*G_h*(-2.02*C[5]+fcal[2]*(.896e-3*nu_Sigma_f[1]*T[1]+.896e-3*nu_
// Variables represented by this insertion:
//[1, T[2], T[1]]
// The diagonal element is set to:
// 1+1.010*G_h
                        ROW_POS = X_st + 4 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
// Zero mat structure  [C[6], T[2], T1[2], T2[2]]
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
```

```
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        COL_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_ADDzero(COL_POS);
                        VALUE = 1 + 1.010 * G_h;
                        matrixdr_ADD(ROW_POS, VALUE);
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                        VALUE =
                            -.4480e-3 * G_h * fcal[2] * grid[x + 0][y + 0]->c->nu_Sigma_f[1];
                        matrixdr_ADD(COL_POS, VALUE);
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                        VALUE =
                            -.4480e-3 * G_h * fcal[2] * grid[x + 0][y + 0]->c->nu_Sigma_f[0];
                        matrixdr_ADD(COL_POS, VALUE);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        VALUE =
                            1. * SUP[11] + .5 * G_h * (-2.02 * SUP[11] +
                                                       fcal[2] * (.896e-3 *
                                                                  grid[x +
                                                                       0][y +
                                                                          0]->
                                                                  c->
                                                                  nu_Sigma_f
                                                                  [0] *
                                                                  SUP[1] +
                                                                  .896e-3 *
                                                                  grid[x +
                                                                       0][y +
                                                                          0]->
                                                                  c->nu_Sigma_f[1] * SUP[4]));
                        vectdr_wr(bid, ROW_POS, VALUE);
/*****************/
// var[1]=1
// cof[1]=-1.*SUP[12]-.5*G_h*(-4.49*SUP[12]+fcal[2]*(.182e-3*grid[x+0][y+0]->c->nu_Sigma_f[0]*S
// var[2]=T[2]
// cof[2]=-.910e-4*G_h*fcal[2]*grid[x+0][y+0]->c->nu_Sigma_f[1]
// var[3]=T[1]
// cof[3]=-.910e-4*G_h*fcal[2]*grid[x+0][y+0]->c->nu_Sigma_f[0]
// diag=C[6]
// diagmult=1+2.245*G_h
/*****************/
// Row for EQ C[6] = SUP[12]+.5*G_h*(-4.49*C[6]+fcal[2]*(.182e-3*nu_Sigma_f[1]*T[1]+.182e-3*nu_
// Variables represented by this insertion:
//[1, T[2], T[1]]
// The diagonal element is set to:
// 1+2.245*G_h
                        ROW_POS = X_st + 5 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
// Zero mat structure  [C[6], T[2], T1[2], T2[2]]
```

```
                    COL_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_ADDzero(COL_POS);
                    COL_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_ADDzero(COL_POS);
                    COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
                    matrixdr_ADDzero(COL_POS);
                    COL_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_ADDzero(COL_POS);
                    COL_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_ADDzero(COL_POS);
                    COL_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_ADDzero(COL_POS);
                    COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
                    matrixdr_ADDzero(COL_POS);
                    COL_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
                    matrixdr_ADDzero(COL_POS);
                    COL_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_ADDzero(COL_POS);
                    COL_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_ADDzero(COL_POS);
                    COL_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_ADDzero(COL_POS);
                    COL_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_ADDzero(COL_POS);
                    VALUE = 1 + 2.245 * G_h;
                    matrixdr_ADD(ROW_POS, VALUE);
                    COL_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                    VALUE =
                        -.910e-4 * G_h * fcal[2] * grid[x + 0][y + 0]->c->nu_Sigma_f[1];
                    matrixdr_ADD(COL_POS, VALUE);
                    COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                    VALUE =
                        -.910e-4 * G_h * fcal[2] * grid[x + 0][y + 0]->c->nu_Sigma_f[0];
                    matrixdr_ADD(COL_POS, VALUE);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    VALUE =
                        1. * SUP[12] + .5 * G_h * (-4.49 * SUP[12] +
                                                   fcal[2] * (.182e-3 *
                                                              grid[x +
                                                                   0][y +
                                                                      0]->
                                                              c->
                                                              nu_Sigma_f
                                                              [0] *
                                                              SUP[1] +
                                                              .182e-3 *
                                                              grid[x +
                                                                   0][y +
                                                                      0]->
                                                              c->nu_Sigma_f[1] * SUP[4]));
                    vectdr_wr(bid, ROW_POS, VALUE);
                }           // ***** end_if((grid[x][y]->cell_id=key))
            }               // ***** end_for(x)
// The mode is 2
//*******************Call made to ring:-corners()
            if (MAT_ROW_fin[Mid][X_st] == 0) {
                y = 0;
                x = 0;
                if (rs_ptr_get(idx, posx + (x - .5) * dx, +posy + (y - .5) * dy, 1)
                    == NULL) {
// ********************* Call made to gen:-reflect
// There are no maps beyond this boarder so reflect vars
// reflection for [C[6], T[2], T1[2], T2[2]]
                    y = 0;
```

```
x = 0;
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + (x + 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 1 + (x + 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 2 + (x + 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 3 + (x + 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 4 + (x + 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 5 + (x + 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 6 + (x + 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 7 + (x + 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 8 + (x + 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
```

```
                    ROW_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    COL_POS = X_st + 9 + (x + 1 + (y + 1) * ROWLEN) * P_SIZE;
                    matrixdr_ADD(COL_POS, -1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                    ROW_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    COL_POS = X_st + 10 + (x + 1 + (y + 1) * ROWLEN) * P_SIZE;
                    matrixdr_ADD(COL_POS, -1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                    ROW_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    COL_POS = X_st + 11 + (x + 1 + (y + 1) * ROWLEN) * P_SIZE;
                    matrixdr_ADD(COL_POS, -1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                } else {
// ********************* Call made to ring:-interp_loop
// ********************* Call made to ring:-interp_row
                    y = 0;
                    x = 0;
                    ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = 0;
                    x = 0;
                    ROW_POS = X_st + 1 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = 0;
                    x = 0;
                    ROW_POS = X_st + y * ROWLEN * P_SIZE + 2 + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 3, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = 0;
```

```
                    x = 0;
                    ROW_POS = X_st + 3 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 4, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = 0;
                    x = 0;
                    ROW_POS = X_st + 4 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 5, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = 0;
                    x = 0;
                    ROW_POS = X_st + 5 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 6, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = 0;
                    x = 0;
                    ROW_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = 0;
                    x = 0;
                    ROW_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
```

```
                    y = 0;
                    x = 0;
                    ROW_POS = X_st + 8 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T1,
                                  (posx + x * dx - .5 * dx),
                                  (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = 0;
                    x = 0;
                    ROW_POS = X_st + 9 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T1,
                                  (posx + x * dx - .5 * dx),
                                  (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = 0;
                    x = 0;
                    ROW_POS = X_st + 10 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T2,
                                  (posx + x * dx - .5 * dx),
                                  (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = 0;
                    x = 0;
                    ROW_POS = X_st + 11 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T2,
                                  (posx + x * dx - .5 * dx),
                                  (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
                }
                y = 0;
                x = dimx + 1;
                if (rs_ptr_get(idx, posx + (x - .5) * dx, +posy + (y - .5) * dy, 1)
                    == NULL) {
// ********************* Call made to gen:-reflect
// There are no maps beyond this boarder so reflect vars
// reflection for [C[6], T[2], T1[2], T2[2]]
                    y = 0;
                    x = dimx + 1;
                    ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
```

```
COL_POS = X_st + (x - 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 1 + (x - 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 2 + (x - 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 3 + (x - 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 4 + (x - 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 5 + (x - 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 6 + (x - 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 7 + (x - 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 8 + (x - 1 + (y + 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 9 + (x - 1 + (y + 1) * ROWLEN) * P_SIZE;
```

```
                    matrixdr_ADD(COL_POS, -1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                    ROW_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    COL_POS = X_st + 10 + (x - 1 + (y + 1) * ROWLEN) * P_SIZE;
                    matrixdr_ADD(COL_POS, -1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                    ROW_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    COL_POS = X_st + 11 + (x - 1 + (y + 1) * ROWLEN) * P_SIZE;
                    matrixdr_ADD(COL_POS, -1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                } else {
// ******************** Call made to ring:-interp_loop
// ******************** Call made to ring:-interp_row
                    y = 0;
                    x = dimx + 1;
                    ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                  (posx + x * dx - .5 * dx),
                                  (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = 0;
                    x = dimx + 1;
                    ROW_POS = X_st + 1 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                  (posx + x * dx - .5 * dx),
                                  (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = 0;
                    x = dimx + 1;
                    ROW_POS = X_st + y * ROWLEN * P_SIZE + 2 + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                  (posx + x * dx - .5 * dx),
                                  (posy + y * dy - .5 * dy), 3, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = 0;
                    x = dimx + 1;
                    ROW_POS = X_st + 3 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
```

```
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 4, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = 0;
                    x = dimx + 1;
                    ROW_POS = X_st + 4 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 5, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = 0;
                    x = dimx + 1;
                    ROW_POS = X_st + 5 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 6, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = 0;
                    x = dimx + 1;
                    ROW_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = 0;
                    x = dimx + 1;
                    ROW_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = 0;
                    x = dimx + 1;
                    ROW_POS = X_st + 8 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
```

```
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T1,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = 0;
                    x = dimx + 1;
                    ROW_POS = X_st + 9 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T1,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = 0;
                    x = dimx + 1;
                    ROW_POS = X_st + 10 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T2,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = 0;
                    x = dimx + 1;
                    ROW_POS = X_st + 11 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T2,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
                }
                y = dimy + 1;
                x = 0;
                if (rs_ptr_get(idx, posx + (x - .5) * dx, +posy + (y - .5) * dy, 1)
                    == NULL) {
// ******************** Call made to gen:-reflect
// There are no maps beyond this boarder so reflect vars
// reflection for [C[6], T[2], T1[2], T2[2]]
                    y = dimy + 1;
                    x = 0;
                    ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    COL_POS = X_st + (x + 1 + (y - 1) * ROWLEN) * P_SIZE;
                    matrixdr_ADD(COL_POS, -1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
```

```
ROW_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 1 + (x + 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 2 + (x + 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 3 + (x + 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 4 + (x + 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 5 + (x + 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 6 + (x + 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 7 + (x + 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 8 + (x + 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 9 + (x + 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
```

```
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    COL_POS = X_st + 10 + (x + 1 + (y - 1) * ROWLEN) * P_SIZE;
                    matrixdr_ADD(COL_POS, -1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                    ROW_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    COL_POS = X_st + 11 + (x + 1 + (y - 1) * ROWLEN) * P_SIZE;
                    matrixdr_ADD(COL_POS, -1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                } else {
// ********************* Call made to ring:-interp_loop
// ********************* Call made to ring:-interp_row
                    y = dimy + 1;
                    x = 0;
                    ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = dimy + 1;
                    x = 0;
                    ROW_POS = X_st + 1 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = dimy + 1;
                    x = 0;
                    ROW_POS = X_st + y * ROWLEN * P_SIZE + 2 + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 3, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = dimy + 1;
                    x = 0;
                    ROW_POS = X_st + 3 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 4, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
```

```
                    matrixdr_ADD(col[i], -1 * weight[i]);
                matrixdr_CLOSE(Mid, ROW_POS);
                vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                y = dimy + 1;
                x = 0;
                ROW_POS = X_st + 4 + y * ROWLEN * P_SIZE + P_SIZE * x;
                matrixdr_OPEN(Mid, ROW_POS);
                matrixdr_ADD(ROW_POS, 1.);
                rs_far_border(idx, rs_prop_C,
                            (posx + x * dx - .5 * dx),
                            (posy + y * dy - .5 * dy), 5, 1, col, weight, ROW_POS);
                for (i = 0; i < 16; i++)
                    matrixdr_ADD(col[i], -1 * weight[i]);
                matrixdr_CLOSE(Mid, ROW_POS);
                vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                y = dimy + 1;
                x = 0;
                ROW_POS = X_st + 5 + y * ROWLEN * P_SIZE + P_SIZE * x;
                matrixdr_OPEN(Mid, ROW_POS);
                matrixdr_ADD(ROW_POS, 1.);
                rs_far_border(idx, rs_prop_C,
                            (posx + x * dx - .5 * dx),
                            (posy + y * dy - .5 * dy), 6, 1, col, weight, ROW_POS);
                for (i = 0; i < 16; i++)
                    matrixdr_ADD(col[i], -1 * weight[i]);
                matrixdr_CLOSE(Mid, ROW_POS);
                vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                y = dimy + 1;
                x = 0;
                ROW_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                matrixdr_OPEN(Mid, ROW_POS);
                matrixdr_ADD(ROW_POS, 1.);
                rs_far_border(idx, rs_prop_T,
                            (posx + x * dx - .5 * dx),
                            (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                for (i = 0; i < 16; i++)
                    matrixdr_ADD(col[i], -1 * weight[i]);
                matrixdr_CLOSE(Mid, ROW_POS);
                vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                y = dimy + 1;
                x = 0;
                ROW_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                matrixdr_OPEN(Mid, ROW_POS);
                matrixdr_ADD(ROW_POS, 1.);
                rs_far_border(idx, rs_prop_T,
                            (posx + x * dx - .5 * dx),
                            (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                for (i = 0; i < 16; i++)
                    matrixdr_ADD(col[i], -1 * weight[i]);
                matrixdr_CLOSE(Mid, ROW_POS);
                vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                y = dimy + 1;
                x = 0;
                ROW_POS = X_st + 8 + y * ROWLEN * P_SIZE + P_SIZE * x;
                matrixdr_OPEN(Mid, ROW_POS);
                matrixdr_ADD(ROW_POS, 1.);
                rs_far_border(idx, rs_prop_T1,
                            (posx + x * dx - .5 * dx),
                            (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
```

```
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = dimy + 1;
                    x = 0;
                    ROW_POS = X_st + 9 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T1,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = dimy + 1;
                    x = 0;
                    ROW_POS = X_st + 10 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T2,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = dimy + 1;
                    x = 0;
                    ROW_POS = X_st + 11 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T2,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
                }
                y = dimy + 1;
                x = dimx + 1;
                if (rs_ptr_get(idx, posx + (x - .5) * dx, +posy + (y - .5) * dy, 1)
                    == NULL) {
// ********************* Call made to gen:-reflect
// There are no maps beyond this boarder so reflect vars
// reflection for [C[6], T[2], T1[2], T2[2]]
                    y = dimy + 1;
                    x = dimx + 1;
                    ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    COL_POS = X_st + (x - 1 + (y - 1) * ROWLEN) * P_SIZE;
                    matrixdr_ADD(COL_POS, -1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                    ROW_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    COL_POS = X_st + 1 + (x - 1 + (y - 1) * ROWLEN) * P_SIZE;
```

```
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 2 + (x - 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 3 + (x - 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 4 + (x - 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 5 + (x - 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 6 + (x - 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 7 + (x - 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 8 + (x - 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 9 + (x - 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 10 + (x - 1 + (y - 1) * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
```

```
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                    ROW_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    COL_POS = X_st + 11 + (x - 1 + (y - 1) * ROWLEN) * P_SIZE;
                    matrixdr_ADD(COL_POS, -1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                } else {
// ********************* Call made to ring:-interp_loop
// ********************* Call made to ring:-interp_row
                    y = dimy + 1;
                    x = dimx + 1;
                    ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = dimy + 1;
                    x = dimx + 1;
                    ROW_POS = X_st + 1 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = dimy + 1;
                    x = dimx + 1;
                    ROW_POS = X_st + y * ROWLEN * P_SIZE + 2 + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 3, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                    y = dimy + 1;
                    x = dimx + 1;
                    ROW_POS = X_st + 3 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 4, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
```

```
                    y = dimy + 1;
                    x = dimx + 1;
                    ROW_POS = X_st + 4 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                  (posx + x * dx - .5 * dx),
                                  (posy + y * dy - .5 * dy), 5, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = dimy + 1;
                    x = dimx + 1;
                    ROW_POS = X_st + 5 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_C,
                                  (posx + x * dx - .5 * dx),
                                  (posy + y * dy - .5 * dy), 6, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = dimy + 1;
                    x = dimx + 1;
                    ROW_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T,
                                  (posx + x * dx - .5 * dx),
                                  (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = dimy + 1;
                    x = dimx + 1;
                    ROW_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T,
                                  (posx + x * dx - .5 * dx),
                                  (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = dimy + 1;
                    x = dimx + 1;
                    ROW_POS = X_st + 8 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T1,
                                  (posx + x * dx - .5 * dx),
                                  (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
```

```
// ******************** Call made to ring:-interp_row
                    y = dimy + 1;
                    x = dimx + 1;
                    ROW_POS = X_st + 9 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T1,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = dimy + 1;
                    x = dimx + 1;
                    ROW_POS = X_st + 10 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T2,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                    y = dimy + 1;
                    x = dimx + 1;
                    ROW_POS = X_st + 11 + y * ROWLEN * P_SIZE + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    rs_far_border(idx, rs_prop_T2,
                                (posx + x * dx - .5 * dx),
                                (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                    for (i = 0; i < 16; i++)
                        matrixdr_ADD(col[i], -1 * weight[i]);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
                }
            }
//******************Call made to ring:-make()
/*------------------ East border ----------------------*/
            for (xyindex = 1; xyindex <= dimy; xyindex++) {
                cons_method = 1;
                if (grid[dimx][xyindex]->cell_id == key)
                    if (grid[dimx + 1][xyindex]->Be == 0) {
// ******************** Call made to gen:-reflect
// There are no maps beyond this boarder so reflect vars
// reflection for [C[6], T[2], T1[2], T2[2]]
                        y = xyindex;
                        x = dimx + 1;
                        ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        COL_POS = X_st + (x - 1 + y * ROWLEN) * P_SIZE;
                        matrixdr_ADD(COL_POS, -1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                        ROW_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        COL_POS = X_st + 1 + (x - 1 + y * ROWLEN) * P_SIZE;
                        matrixdr_ADD(COL_POS, -1.);
```

```
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 2 + (x - 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 3 + (x - 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 4 + (x - 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 5 + (x - 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 6 + (x - 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 7 + (x - 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 8 + (x - 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 9 + (x - 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 10 + (x - 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
```

```
                        vectdr_wr(bid, ROW_POS, 0.);
                        ROW_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        COL_POS = X_st + 11 + (x - 1 + y * ROWLEN) * P_SIZE;
                        matrixdr_ADD(COL_POS, -1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                    } else if (grid[dimx + 1][xyindex]->Be == 1) {
// ********************* Call made to ring:-interp_loop
// ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = dimx + 1;
                        ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_C,
                                      (posx + x * dx - .5 * dx),
                                      (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = dimx + 1;
                        ROW_POS = X_st + 1 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_C,
                                      (posx + x * dx - .5 * dx),
                                      (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = dimx + 1;
                        ROW_POS = X_st + y * ROWLEN * P_SIZE + 2 + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_C,
                                      (posx + x * dx - .5 * dx),
                                      (posy + y * dy - .5 * dy), 3, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = dimx + 1;
                        ROW_POS = X_st + 3 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_C,
                                      (posx + x * dx - .5 * dx),
                                      (posy + y * dy - .5 * dy), 4, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                        y = xyindex;
```

```
                        x = dimx + 1;
                        ROW_POS = X_st + 4 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_C,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 5, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                        y = xyindex;
                        x = dimx + 1;
                        ROW_POS = X_st + 5 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_C,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 6, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                        y = xyindex;
                        x = dimx + 1;
                        ROW_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_T,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                        y = xyindex;
                        x = dimx + 1;
                        ROW_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_T,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
                        y = xyindex;
                        x = dimx + 1;
                        ROW_POS = X_st + 8 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_T1,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ******************** Call made to ring:-interp_row
```

```
                        y = xyindex;
                        x = dimx + 1;
                        ROW_POS = X_st + 9 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_T1,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = dimx + 1;
                        ROW_POS = X_st + 10 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_T2,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = dimx + 1;
                        ROW_POS = X_st + 11 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_T2,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
                    } else if (grid[dimx + 1][xyindex]->Be > 1) {
                        cons_method = grid[dimx + 1][xyindex]->Be;
// ************* ring:-conserve
// LAPL_VARS1:={LAPL(1/3/Sigma_tr[1]*T[1]), LAPL(1/3/Sigma_tr[2]*T[2])}
// PDEs_arg:=[[T[1] = sup[T[1],3]+G_h/G_ho*sup[T1[1],3]+G_h^2/G_ho^2*sup[T2[1],3]+.6666666667*G
// ************* ring:-conserve_shape_code called for T[1]
// the geometric position for the row is based on little_phi*
                        y = xyindex;
                        x = dimx + 1;
                        ROW_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
// ************* ring:-conserve_exact_big_PHI_W_E for T[1]
// set fy, the floating point x position to the E/W edge
                        y = xyindex;
                        fx = posx + dx * (dimx + .1);
                        ocp = (void *) &grid[dimx][y];
                        rs_exact_PHI_WE(idx, fx, posy + (xyindex - 1) * dy,
                                        dy, PHIs, PHI, PHIsx, PHIsy, fr,
                                        (void **) cps, (void **) cp, &unfr, ocp);
                        reject_fr = 100;        // must be initialized to zero to work
                        i = 0;
                        while (fr[i] >= 0) {
// Computing far constant 1/Sigma_tr[1]
                            con = 1.0 * 1 / cp[i]->c->Sigma_tr[0];
// Computing far constant 1/Sigma_tr[1]
                            cons = 1.0 * 1 / cps[i]->c->Sigma_tr[0];
```

```
// disabled reject_fr+=dy*fr[i]*Abs(con-cons)/(Abs(con)+Abs(cons));
                            i++;
                            if (i > MAX_R)
                                abort();
                        }
                        if (cons_method != 3)
                            reject_fr = 100;
                        if (reject_fr < G_RTHRESH) {
                            printf("r(%g)", reject_fr);
                        } else {
                            i = 0;
                            while (fr[i] >= 0) {
// Computing far constant 1/Sigma_tr[1]
                                con = 1.0 * 1 / cp[i]->c->Sigma_tr[0];
// Computing far constant 1/Sigma_tr[1]
                                cons = 1.0 * 1 / cps[i]->c->Sigma_tr[0];
                                matrixdr_ADD(PHI[i] + 6, fr[i] * (con + cons) / 2);
                                matrixdr_ADD(PHIs[i] + 6, -fr[i] * (con + cons) / 2);
                                i++;
                            }
                        }
// ************* ring:-conserve_shape_small_PHI_W_E for T[1]
                        y = xyindex;
                        x = dimx;
// Computing far constant 1/Sigma_tr[1]
                        con = 1.0 * 1 / grid[x + 0][y + 0]->c->Sigma_tr[0];
// Computing far constant 1/Sigma_tr[1]
                        cons = 1.0 * 1 / grid[x + 1][y + 0]->c->Sigma_tr[0];
                        if (reject_fr > G_RTHRESH) {
                            COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                            matrixdr_ADD(COL_POS, (con + cons) * dy / dx / 2);
                        }
                        x = dimx + 1;
                        if (reject_fr < G_RTHRESH) {
                            rs_far_border(idx, rs_prop_T,
                                          (posx + x * dx - .5 * dx),
                                          (posy + y * dy - .5 * dy), 1, 1, col, weight,
                                          ROW_POS);
                            for (i = 0; i < 16; i++)
                                matrixdr_ADD(col[i], (con + cons) * dy / dx / 2 * weight[i]);
                        }
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                        matrixdr_ADD(COL_POS, -(con + cons) * dy / dx / 2);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ************* ring:-conserve_shape_code called for T[2]
// the geometric position for the row is based on little_phi*
                        y = xyindex;
                        x = dimx + 1;
                        ROW_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
// ************* ring:-conserve_exact_big_PHI_W_E for T[2]
// set fy, the floating point x position to the E/W edge
                        y = xyindex;
                        fx = posx + dx * (dimx + .1);
                        ocp = (void *) &grid[dimx][y];
                        rs_exact_PHI_WE(idx, fx, posy + (xyindex - 1) * dy,
                                        dy, PHIs, PHI, PHIsx, PHIsy, fr,
                                        (void **) cps, (void **) cp, &unfr, ocp);
                        reject_fr = 100;        // must be initialized to zero to work
                        i = 0;
                        while (fr[i] >= 0) {
// Computing far constant 1/Sigma_tr[2]
                            con = 1.0 * 1 / cp[i]->c->Sigma_tr[1];
```

```
// Computing far constant 1/Sigma_tr[2]
                            cons = 1.0 * 1 / cps[i]->c->Sigma_tr[1];
// disabled reject_fr+=dy*fr[i]*Abs(con-cons)/(Abs(con)+Abs(cons));
                            i++;
                            if (i > MAX_R)
                                abort();
                        }
                        if (cons_method != 3)
                            reject_fr = 100;
                        if (reject_fr < G_RTHRESH) {
                            printf("r(%g)", reject_fr);
                        } else {
                            i = 0;
                            while (fr[i] >= 0) {
// Computing far constant 1/Sigma_tr[2]
                                con = 1.0 * 1 / cp[i]->c->Sigma_tr[1];
// Computing far constant 1/Sigma_tr[2]
                                cons = 1.0 * 1 / cps[i]->c->Sigma_tr[1];
                                matrixdr_ADD(PHI[i] + 7, fr[i] * (con + cons) / 2);
                                matrixdr_ADD(PHIs[i] + 7, -fr[i] * (con + cons) / 2);
                                i++;
                            }
                        }
// ************* ring:-conserve_shape_small_PHI_W_E for T[2]
                        y = xyindex;
                        x = dimx;
// Computing far constant 1/Sigma_tr[2]
                        con = 1.0 * 1 / grid[x + 0][y + 0]->c->Sigma_tr[1];
// Computing far constant 1/Sigma_tr[2]
                        cons = 1.0 * 1 / grid[x + 1][y + 0]->c->Sigma_tr[1];
                        if (reject_fr > G_RTHRESH) {
                            COL_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                            matrixdr_ADD(COL_POS, (con + cons) * dy / dx / 2);
                        }
                        x = dimx + 1;
                        if (reject_fr < G_RTHRESH) {
                            rs_far_border(idx, rs_prop_T,
                                          (posx + x * dx - .5 * dx),
                                          (posy + y * dy - .5 * dy), 2, 1, col, weight,
                                          ROW_POS);
                            for (i = 0; i < 16; i++)
                                matrixdr_ADD(col[i], (con + cons) * dy / dx / 2 * weight[i]);
                        }
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                        matrixdr_ADD(COL_POS, -(con + cons) * dy / dx / 2);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
/****************** Call made to ring:-end_fill called ALL_SYMS=[T[2], T[1]]*/
                        y = xyindex;
                        x = dimx + 1;
// Zero Filling structure [C[6], T[2], T1[2], T2[2]]
// Symbols excluded [T[2], T[1]]
                        ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                        ROW_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                        ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
                        matrixdr_OPEN(Mid, ROW_POS);
```

```
                        matrixdr_ADD(ROW_POS, 1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                        ROW_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                        ROW_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                        ROW_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                        ROW_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                        ROW_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                        ROW_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                        ROW_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                    } else
                        abort();
            };                  //****************** end for(xyindex)
/*----------------- West border ----------------------*/
            for (xyindex = 1; xyindex <= dimy; xyindex++) {
                if (grid[1][xyindex]->cell_id == key)
                    if (grid[0][xyindex]->Bw == 0) {
// ******************** Call made to gen:-reflect
// There are no maps beyond this boarder so reflect vars
// reflection for [C[6], T[2], T1[2], T2[2]]
                        y = xyindex;
                        x = 0;
                        ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        COL_POS = X_st + (x + 1 + y * ROWLEN) * P_SIZE;
                        matrixdr_ADD(COL_POS, -1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                        ROW_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        COL_POS = X_st + 1 + (x + 1 + y * ROWLEN) * P_SIZE;
                        matrixdr_ADD(COL_POS, -1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
```

```
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 2 + (x + 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 3 + (x + 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 4 + (x + 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 5 + (x + 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 6;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 6 + (x + 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 7;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 7 + (x + 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 8 + (x + 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 9 + (x + 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
matrixdr_OPEN(Mid, ROW_POS);
matrixdr_ADD(ROW_POS, 1.);
COL_POS = X_st + 10 + (x + 1 + y * ROWLEN) * P_SIZE;
matrixdr_ADD(COL_POS, -1.);
matrixdr_CLOSE(Mid, ROW_POS);
vectdr_wr(bid, ROW_POS, 0.);
ROW_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
```

```
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        COL_POS = X_st + 11 + (x + 1 + y * ROWLEN) * P_SIZE;
                        matrixdr_ADD(COL_POS, -1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                    } else if (grid[0][xyindex]->Bw == 1) {
// ********************* Call made to ring:-interp_loop
// ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = 0;
                        ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_C,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = 0;
                        ROW_POS = X_st + 1 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_C,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = 0;
                        ROW_POS = X_st + y * ROWLEN * P_SIZE + 2 + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_C,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 3, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = 0;
                        ROW_POS = X_st + 3 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_C,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 4, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = 0;
                        ROW_POS = X_st + 4 + y * ROWLEN * P_SIZE + P_SIZE * x;
```

```
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_C,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 5, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
    // ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = 0;
                        ROW_POS = X_st + 5 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_C,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 6, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
    // ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = 0;
                        ROW_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_T,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
    // ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = 0;
                        ROW_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_T,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
    // ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = 0;
                        ROW_POS = X_st + 8 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_T1,
                                    (posx + x * dx - .5 * dx),
                                    (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
    // ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = 0;
```

```
                        ROW_POS = X_st + 9 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_T1,
                                      (posx + x * dx - .5 * dx),
                                      (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = 0;
                        ROW_POS = X_st + 10 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_T2,
                                      (posx + x * dx - .5 * dx),
                                      (posy + y * dy - .5 * dy), 1, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
// ********************* Call made to ring:-interp_row
                        y = xyindex;
                        x = 0;
                        ROW_POS = X_st + 11 + y * ROWLEN * P_SIZE + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        rs_far_border(idx, rs_prop_T2,
                                      (posx + x * dx - .5 * dx),
                                      (posy + y * dy - .5 * dy), 2, 1, col, weight, ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], -1 * weight[i]);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
                    } else if (grid[0][xyindex]->Bw > 1) {
                        cons_method = grid[0][xyindex]->Bw;
// ************* ring:-conserve
// LAPL_VARS1:={LAPL(1/3/Sigma_tr[1]*T[1]), LAPL(1/3/Sigma_tr[2]*T[2])}
// PDEs_arg:=[[T[1] = sup[T[1],3]+G_h/G_ho*sup[T1[1],3]+G_h^2/G_ho^2*sup[T2[1],3]+.6666666667*G
// ************* ring:-conserve_shape_code called for T[1]
// the geometric position for the row is based on little_phi*
                        y = xyindex;
                        x = 0;
                        ROW_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                        matrixdr_OPEN(Mid, ROW_POS);
// ************* ring:-conserve_exact_big_PHI_W_E for T[1]
// set fy, the floating point x position to the E/W edge
                        y = xyindex;
                        fx = posx - .1 * dx;
                        ocp = (void *) &grid[1][y];
                        rs_exact_PHI_WE(idx, fx, posy + (xyindex - 1) * dy,
                                        dy, PHIs, PHI, PHIsx, PHIsy, fr,
                                        (void **) cps, (void **) cp, &unfr, ocp);
                        reject_fr = 100;        // must be initialized to zero to work
                        i = 0;
                        while (fr[i] >= 0) {
// Computing far constant 1/Sigma_tr[1]
                            con = 1.0 * 1 / cp[i]->c->Sigma_tr[0];
// Computing far constant 1/Sigma_tr[1]
                            cons = 1.0 * 1 / cps[i]->c->Sigma_tr[0];
// disabled reject_fr+=dy*fr[i]*Abs(con-cons)/(Abs(con)+Abs(cons));
                            i++;
```

```
                        if (i > MAX_R)
                            abort();
                    }
                    if (cons_method != 3)
                        reject_fr = 100;
                    if (reject_fr < G_RTHRESH) {
                        printf("r(%g)", reject_fr);
                    } else {
                        i = 0;
                        while (fr[i] >= 0) {
// Computing far constant 1/Sigma_tr[1]
                            con = 1.0 * 1 / cp[i]->c->Sigma_tr[0];
// Computing far constant 1/Sigma_tr[1]
                            cons = 1.0 * 1 / cps[i]->c->Sigma_tr[0];
                            matrixdr_ADD(PHI[i] + 6, fr[i] * (con + cons) / 2);
                            matrixdr_ADD(PHIs[i] + 6, -fr[i] * (con + cons) / 2);
                            i++;
                        }
                    }
// ************* ring:-conserve_shape_small_PHI_W_E for T[1]
                    y = xyindex;
                    x = 1;
// Computing far constant 1/Sigma_tr[1]
                    con = 1.0 * 1 / grid[x + 0][y + 0]->c->Sigma_tr[0];
// Computing far constant 1/Sigma_tr[1]
                    cons = 1.0 * 1 / grid[x + -1][y + 0]->c->Sigma_tr[0];
                    if (reject_fr > G_RTHRESH) {
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                        matrixdr_ADD(COL_POS, (con + cons) * dy / dx / 2);
                    }
                    x = 0;
                    if (reject_fr < G_RTHRESH) {
                        rs_far_border(idx, rs_prop_T,
                                      (posx + x * dx - .5 * dx),
                                      (posy + y * dy - .5 * dy), 1, 1, col, weight,
                                      ROW_POS);
                        for (i = 0; i < 16; i++)
                            matrixdr_ADD(col[i], (con + cons) * dy / dx / 2 * weight[i]);
                    }
                    COL_POS = X_st + y * ROWLEN * P_SIZE + 6 + P_SIZE * x;
                    matrixdr_ADD(COL_POS, -(con + cons) * dy / dx / 2);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0);
// ************* ring:-conserve_shape_code called for T[2]
// the geometric position for the row is based on little_phi*
                    y = xyindex;
                    x = 0;
                    ROW_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                    matrixdr_OPEN(Mid, ROW_POS);
// ************* ring:-conserve_exact_big_PHI_W_E for T[2]
// set fy, the floating point x position to the E/W edge
                    y = xyindex;
                    fx = posx - .1 * dx;
                    ocp = (void *) &grid[1][y];
                    rs_exact_PHI_WE(idx, fx, posy + (xyindex - 1) * dy,
                                    dy, PHIs, PHI, PHIsx, PHIsy, fr,
                                    (void **) cps, (void **) cp, &unfr, ocp);
                    reject_fr = 100;        // must be initialized to zero to work
                    i = 0;
                    while (fr[i] >= 0) {
// Computing far constant 1/Sigma_tr[2]
                        con = 1.0 * 1 / cp[i]->c->Sigma_tr[1];
// Computing far constant 1/Sigma_tr[2]
                        cons = 1.0 * 1 / cps[i]->c->Sigma_tr[1];
```

```
// disabled reject_fr+=dy*fr[i]*Abs(con-cons)/(Abs(con)+Abs(cons));
                            i++;
                            if (i > MAX_R)
                                abort();
                        }
                        if (cons_method != 3)
                            reject_fr = 100;
                        if (reject_fr < G_RTHRESH) {
                            printf("r(%g)", reject_fr);
                        } else {
                            i = 0;
                            while (fr[i] >= 0) {
// Computing far constant 1/Sigma_tr[2]
                                con = 1.0 * 1 / cp[i]->c->Sigma_tr[1];
// Computing far constant 1/Sigma_tr[2]
                                cons = 1.0 * 1 / cps[i]->c->Sigma_tr[1];
                                matrixdr_ADD(PHI[i] + 7, fr[i] * (con + cons) / 2);
                                matrixdr_ADD(PHIs[i] + 7, -fr[i] * (con + cons) / 2);
                                i++;
                            }
                        }
// ************* ring:-conserve_shape_small_PHI_W_E for T[2]
                        y = xyindex;
                        x = 1;
// Computing far constant 1/Sigma_tr[2]
                        con = 1.0 * 1 / grid[x + 0][y + 0]->c->Sigma_tr[1];
// Computing far constant 1/Sigma_tr[2]
                        cons = 1.0 * 1 / grid[x + -1][y + 0]->c->Sigma_tr[1];
                        if (reject_fr > G_RTHRESH) {
                            COL_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                            matrixdr_ADD(COL_POS, (con + cons) * dy / dx / 2);
                        }
                        x = 0;
                        if (reject_fr < G_RTHRESH) {
                            rs_far_border(idx, rs_prop_T,
                                          (posx + x * dx - .5 * dx),
                                          (posy + y * dy - .5 * dy), 2, 1, col, weight,
                                          ROW_POS);
                            for (i = 0; i < 16; i++)
                                matrixdr_ADD(col[i], (con + cons) * dy / dx / 2 * weight[i]);
                        }
                        COL_POS = X_st + y * ROWLEN * P_SIZE + 7 + P_SIZE * x;
                        matrixdr_ADD(COL_POS, -(con + cons) * dy / dx / 2);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0);
/****************** Call made to ring:-end_fill called ALL_SYMS=[T[2], T[1]]*/
                        y = xyindex;
                        x = 0;
// Zero Filling structure [C[6], T[2], T1[2], T2[2]]
// Symbols excluded [T[2], T[1]]
                        ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                        ROW_POS = X_st + 1 + (x + y * ROWLEN) * P_SIZE;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
                        vectdr_wr(bid, ROW_POS, 0.);
                        ROW_POS = X_st + (x + y * ROWLEN) * P_SIZE + 2;
                        matrixdr_OPEN(Mid, ROW_POS);
                        matrixdr_ADD(ROW_POS, 1.);
                        matrixdr_CLOSE(Mid, ROW_POS);
```

```
                    vectdr_wr(bid, ROW_POS, 0.);
                    ROW_POS = X_st + 3 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                    ROW_POS = X_st + 4 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                    ROW_POS = X_st + 5 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                    ROW_POS = X_st + 8 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                    ROW_POS = X_st + 9 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                    ROW_POS = X_st + 10 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                    ROW_POS = X_st + 11 + (x + y * ROWLEN) * P_SIZE;
                    matrixdr_OPEN(Mid, ROW_POS);
                    matrixdr_ADD(ROW_POS, 1.);
                    matrixdr_CLOSE(Mid, ROW_POS);
                    vectdr_wr(bid, ROW_POS, 0.);
                } else
                    abort();
            };              //****************** end for(xyindex)
/*----------------- North border ---------------------*/

            ...
```