ALARA: Analytic and Laplacian Adaptive Radioactivity Analysis

Paul P.H. Wilson

April 1999

UWFDM-1098

ALARA: Analytic and Laplacian Adaptive Radioactivity Analysis

by:

Paul Philip Hood Wilson

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Nuclear Engineering and Engineering Physics)

at the

UNIVERSITY OF WISCONSIN - MADISON

1999

to my father, Philip S. Wilson

with thanks for the Sunday night BASIC lessons

Abstract

While many codes have been written to compute the induced activation and changes in composition caused by neutron irradiation, most of those which are still being updated are only slowly adding functionality and not improving the accuracy, speed and usability of their existing methods. ALARA moves forward in all four of these areas, with primary importance being placed on the accuracy and speed of solution.

By carefully analyzing the various ways to model the physical system, the methods to solve the mathematical problem and the interaction between these two issues, ALARA chooses an optimum combination to achieve high accuracy, fast computation, and enhanced versatility and ease of use. In addition to a set of base features, standard to any activation code, ALARA offers a number of extensions, including arbitray hierarchical irradiation schedules and a form of reverse problem for calculating the detailed activation of specific isotopes.

The physical system is modeled using advanced linear chains, which include the contributions from straightened loops in the reaction scheme, while the truncation philosophy minimizes the discrepancies between the model and the real problem. The mathematical method is then adaptively chosen based on the characteristics of each linear chain to use analytically exact methods when possible and an accurate expansion technique otherwise.

ALARA has been successfully validated against established fusion activation codes using a standard activation benchmark problem. In addition to demonstrating ALARA's accuracy, this validation excerise has demonstrated its speed. Furthermore, by extending the benchmark problem to validate its advanced features, ALARA's flexibility has been proven.

With its modern computational techniques and continuing development, it is hoped that ALARA will become a widely used code for the activation analysis of nuclear systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Background

Since the advent of the nuclear age nearly six decades ago, it has become necessary to study and simulate the effect of radiation on materials. Most nuclear systems, including existing fission power reactors, anticipated fusion power reactors, and a wide variety of experimental facilities, produce large numbers of energetic neutrons. These neutrons interact with the systems' materials, inducing a variety of responses.

1.1 Problem Definition

Activation is just one of the many possible responses resulting from the neutron irradiation of materials. The neutrons interact with the material's nuclei, converting them to different isotopes. With many such reactions possible for most isotopes, each of the original material's isotopes can be partially converted into over 20 others after just one generation. These isotopes, in turn, can undergo similar interactions, leading to yet more isotopes, and so on. Furthermore, many of these isotopes may be radioactive, and their decay products introduce even more isotopes to the physical system. If represented graphically (Figure 1.1), this process forms a tree of isotopes, where each branch in the tree represents either a nuclear reaction (*e.g.* $A \rightarrow B$) or a nuclear decay (*e.g.* $C \rightarrow G$). This process of converting a non-radioactive material to a radioactive one is known as activation.

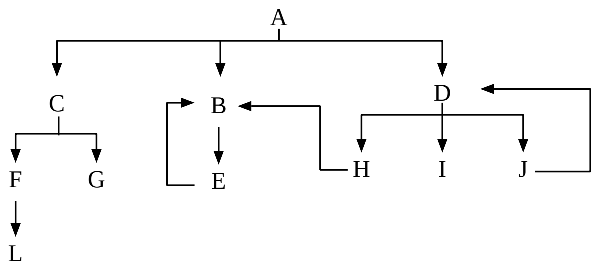


Figure 1.1: A sample activation tree showing the results of activation of isotope A.

After calculating the concentrations of all the various isotopes created by the activation process, other engineering responses can be determined. Multiplying by the radioactive isotopes' decay constants ($\lambda = \ln 2/t_{\frac{1}{2}}$) determines the material's radioactivity. Further multiplying by the average energy of each decay, these radioactivity values can be converted to decay heat results. By comparing the radioactivity to regulated limits, the waste disposal ratings can be determined, indicating how the material must be handled. Incorporating the gamma ray emission information for each radioactive isotope gives the gamma ray source, which may be used as the source term for a radiation transport calculation to determine a radiation dose at some spatial point.

These responses are essential when designing, operating and costing a nuclear system. For safety considerations, it is important to know the inventory of all the radioactive isotopes which may be released and to know the decay heat transient after the system is shutdown. Furthermore, if the radiation dose at a critical point is too high, either for the other components of the system (*e.g.* fusion magnets) or for the personnel who must work with the system, the doses must be mitigated by the addition of shielding to the design. Amongst many other factors, the cost of a system is affected by whether "hands-on" maintenance is possible or remote handling is required. When the lifetime of the system has been reached, the decommissioning cost is influenced by the levels of radioactivity in the various materials.

The mathematical description of the activation process is quite straightforward. The production rate of one isotope, i, from another, j, is proportional to the concentration of that other isotope, N_j. The constant of proportionality is some reaction rate, $P_{j\to i}$, based on nuclear data. For decay reaction paths, the production rate is equal to the product of the decay rate of isotope j and the branching ratio for the reaction leading to i: $P_{j\to i} = \lambda_j b_{j\to i}$. For nuclear reactions, the reaction rate is the inner product of the reaction cross-section for the reaction from j to i and the neutron flux, $P_{j\to i} = \int_0^\infty \sigma^{j\to i}(E)\phi(E)dE$. With the cross-sections represented as group constants, as is usual, this production rate is: $P_{j\to i} = \sum_{g=1}^{G} \sigma_g^{j\to i}\phi_g$. While each such term represents a production path for isotope i, it also represents one for the destruction paths for isotope j. Assuming that there are no other sources (such as mass flux into the system), the rate of change of an isotope's concentration is simply the sum of all the production terms from other isotopes, j, and the destruction terms to other isotopes, k:

$$\dot{N}_i(t) = \sum_{j=1}^{n} P_{j\to i}\left[\phi(t)\right] N_j(t) - \sum_{k=1}^{n} P_{i\to k}\left[\phi(t)\right] N_i(t).$$

By combining all the destruction rates to isotopes k into a total destruction rate for isotope i, and writing the production rates as P_{ij} without explicitly including the dependence on the neutron flux, this ordinary differential equation [ODE] is reduced to

$$\dot{N}_i(t) = \sum_{j=1}^{n} P_{ij} N_j(t) - d_i N_i(t).$$

With one such equation for each of the isotopes in the activation tree, and one activation tree for each of the isotopes in the initial material, the large system of these coupled ordinary differential equations [ODE's] can be written in a matrix formulation as

$$\dot{\vec{N}}(t) = \mathbf{A}\vec{N}(t), \tag{1.1}$$

where $A_{ii} = -d_i$ and $A_{ij} = P_{ij}$. The formal solution to this equation is the matrix exponential:

$$\vec{N}(t) = e^{\mathbf{A}t}\vec{N}(0). \tag{1.2}$$

For large problems with many initial isotopes, many fluxes at different spatial points, and complicated irradiation histories, an activation code is required to calculate the induced radioactivity levels. An activation code must perform two distinct tasks: the physical modeling of the activation tree and the solution of the corresponding mathematical problem. Starting with a list of initial isotopes, the code must first build the activation trees, deciding how large they need to be in order to include all the important contributions. The trees are then converted into their mathematical equivalent and some technique is used to solve the matrix exponential problem. Although these two tasks will be treated as distinct in this work, the way that the trees are built and sub-divided has an important impact on the kind of mathematical method that can be accurately implemented, and vice versa.

1.2 Historical Overview

The computational solutions to this problem have been well studied.[1] Many different approaches for modeling the physical problem have been combined with at least as many mathematical solution techniques. Each combination has advantages and disadvantages, but none has arrived at an optimum mixture of accuracy, efficiency and usability. Even ignoring the issue of usability (where this author feels many codes fail), there are few codes which are keeping up with the demands for greater accuracy in the physical models and mathematical solutions without becoming inconveniently slow.

The predecessors to many modern activation codes were inventory codes (also called burn-up or depletion codes) designed for modeling the burn-up of nuclear fuel and build-up of fission products in nuclear reactors. These codes use a variety of methods for modeling the physical system and solving the mathematical problem, but have historically been divided into three classes based on the mathematical method: time-step based ODE solvers, matrix exponential methods, and linear chain methods.

The time-step based ODE solvers, such as used in FISPIN,[2] use some algebraic approximation of the derivative on the left hand side of equation 1.1. One simple form of this approximation is based on the first principles definition of the derivative:

$$\dot{N}(t) = \lim_{t_i - t_{i-1} \to 0} \frac{N(t_i) - N(t_{i-1})}{t_i - t_{i-1}}$$
$$\therefore \dot{N}(t) \approx \frac{N(t_i) - N(t_{i-1})}{t_i - t_{i-1}}.$$

For the activation problem, where $\vec{N}_i$ is the number density vector at time i, and $\Delta t = t_i - t_{i-1}$, this can be implemented simply in the explicit form:

$$\vec{N}_i = \Delta t \mathbf{A} \vec{N}_{i-1} + \vec{N}_{i-1}. \tag{1.3}$$

More complicated differencing schemes with more accuracy can be developed based on Taylor series expansions in one or two variables, such as the well known Runge Kutta method.

In all cases, however, to ensure accuracy these methods must use time steps small enough that the number density of any single isotope does not change too much during the time step. For a problem with very short-lived isotopes, this time step must be very short, requiring many steps to solve the entire irradiation history, and therefore these methods can be very slow.

The original matrix methods employed in inventory codes such as ORIGEN[3] calculate the series expansion of the $e^{\mathbf{A}t}$ exponential:

$$e^{\mathbf{A}t} = \mathbf{I} + \mathbf{A}t + \frac{\mathbf{A}^2 t^2}{2!} + \frac{\mathbf{A}^3 t^3}{3!} + \ldots .$$

In addition to being prone to round-off error, this expansion may need many terms to converge (if it does converge), which is computationally expensive, due to the large number of matrix multiplications. More recently, this class of methods includes matrix decomposition methods to solve the matrix exponential (see chapter 3).

The final class of solution methods is based on a principle known as linear chains, of which CINDER[4] was one of the first implementations. While both the time-step methods and the matrix exponential methods have traditionally attempted to solve the entire physical problem as one large system of ODE's, the linear chain method breaks the activation tree into a number of chains so that each isotope has a single production term and a single destruction term. This creates a smaller system of ODE's in which the transfer matrix, **A**, is exactly bidiagonal allowing an analytical solution commonly known as the Bateman equations[5] to be used:

$$N_i(t) = N_{i_o} e^{-d_i t} + \sum_{j=1}^{i-1} N_{j_o} \left[\sum_{k=j}^{i-1} \frac{P_{k+1}(e^{-d_k t} - e^{-d_i t})}{d_i - d_k} \prod_{\substack{l=j \\ l \neq k}}^{i-1} \frac{P_{l+1}}{d_l - d_k} \right], \tag{1.4}$$

where $P_{k+1} = P_{k \to k+1}$.

One of the biggest limitations of this class of methods is its inability to model loops in the activation tree. It can be seen in the above equation that there is a singularity when two of the destruction rates are identical. While this may happen coincidentally in any linear chain, it is guaranteed to be the case if the same isotope occurs more than once in the chain. This issue may be less significant for the simulation of fission reactor problems because most of the nuclear reactions required for loops (*e.g.* (n,p) or (n,2n)) are threshold reactions with low or zero cross-sections in the energy domain of fission neutrons.

Activation codes, many of which have been developed for fusion applications, also exist in each of these three classes of solution method. Some are derived directly from an inventory code while others have no such obvious ancestry. Conceptually activation calculations and inventory calculations are one and the same, but the wide variety in the nature of the systems being simulated in an activation problem can require more flexibility than an inventory code may provide. For example, as suggested above, there are some reaction channels which are not relevant to fission inventory calculations. In systems with higher energy neutron fluxes, such as fusion reactors or accelerator-based neutron sources, these additional channels can become important, if not dominant. Additionally, the irradiation history for an activation calculation may be very different from that of a fission reaction system, with, for example, many frequent pulses.

The most widely used fusion activation codes include FISPACT,[6] a direct descendant of the FISPIN inventory code, RACC[7] (and variations[8,9]), REAC,[10] ACAB,[11] and DKR[12] (and variations[1,13,14]). Each uses a different combination of physical and mathematical methods, enabling each to implement unique features. For example, while FISPACT is limited to solutions at a single spatial point and cannot model pulsed histories exactly, the newest versions support sensitivity analyses and secondary activation caused by the light ions emitted by neutron reactions. Newer versions of RACC switched from the time-step based GEAR ODE solver to matrix decomposition methods, enabling the efficient solution of pulsing histories with loops in the activation tree. RACC's method for truncating the activation trees, however, is not ideal and may lead to an inaccurate final solution. Not only was DKR the first to implement exact pulsing solutions, it is able to efficiently solve the activation problem across many points of a complicated geometry. Due to its reliance on linear chains and the Bateman solution, however, DKR has long been criticized for its inability to handle loops in the activation tree. Additionally, DKR has not implemented the ability to model the accumulation of light ions emitted from nuclear reactions.

Despite the wide variety of available activation codes and their various capabilities, there is no single code which provides a complete range of these capabilities. Furthermore, because of their original choice of physical methods, mathematical techniques and/or computational design, few, if any are extensible to include additional capabilities.

1.3 Goals

The goal of this work is to design a fast, accurate and flexible activation code with a wide array of capabilities and features. In addition to establishing a base set of capabilities and features, it is important to look forward to a more advanced set and ensure that the new code will accomodate those features. ALARA is designed to implement the following basic features:

Problem Geometry

- simultaneous solution of activation problem at arbitrary number of spatial points

Truncation

- user-defined calculation precision

Irradiation History

- multi-level pulsed operation histories

Physical Modeling

- accurate handling of loops in the activation tree
- modeling of light ion accumulation

User Features

- user-friendly input file format
- flexible geometry definition options
- user-defined output resolution
- user-defined output responses

The advanced features of ALARA include:

Physical Modeling

- modeling of reverse problem for detailed studies

Irradiation History

- fully arbitrary operation schedules

Mathematical Techniques

- adaptive selection of mathematical method to optimize speed and accuracy

Prior to implementing these features, a careful analysis of the various methods for modeling the physical system, the techniques for solving the mathematical problem, and the way that they influence each other, must be performed. Once the best methods and techniques have been determined, the importance of efficient and portable implementation must not be underestimated. Even the best methods can be implemented poorly leading to inaccurate and slow solutions.

ALARA has been designed with three basic principles in mind: accuracy, speed, and simplicity. These three qualities have been maximized in ALARA after extensive research of the models involved in such calculations. The errors, time of execution, and learning curve have all been made "as low as reasonably achievable".[a] The methods used to model the physical system and to perform the mathematical solution are carefully combined to preserve or enhance the accuracy while accelerating the solution. Throughout all this, there is an underlying effort to ensure that ALARA be user-friendly by providing a simple, well-documented input file format, checking this input for errors, and providing a broad, flexible range of options.

[a]This phrase is the origin of the term ALARA, a well known philosophy in the nuclear industry related to the minimization of radiation exposure when working in radioactive environments.

1.3.1 Accuracy

Despite the list of shortcomings for existing codes, various studies[15,16] have demonstrated that most of these codes achieve a reasonable degree of accuracy, compared to each other as well as compared to analytical solutions. It is important, therefore, that ALARA at least maintain this level of accuracy as it expands its range of modeling options.

The accuracy of the final solution is affected both by how realistically the physical system is modeled and by what mathematical methods are employed for the final solution. Unfortunately, these two requirements often conflict; as the physical model becomes more realistic the required mathematical methods become more approximate or error prone. When modeling the physical problem, two of the most important issues are how to deal with loops in the reaction scheme and how to truncate the theoretically infinite isotopic composition to a finite problem. While the effect of the latter on the mathematical method is negligible, the former has a great impact. In the past, the unwritten rule has been that realistic treatment of loops requires complicated/inefficient mathematical methods. ALARA has broken that rule by finding a physical approximation to the loops which retains problem accuracy and allows for quite simple and efficient mathematical methods. The keys to ALARA's mathematical accuracy are its ability to adaptively choose the mathematical technique and the accuracy of those techniques. Two of the three mathematical techniques which ALARA employs are mathematically exact!

1.3.2 Speed

The most significant factor affecting the speed is the chosen class of mathematical method. In particular, unless a linear transformation matrix method is used, the exact modeling of a pulsed history will require a long time. ALARA employs such matrix methods, solving for the linear transformation from the initial isotopic composition to the final composition for each pulse and inter-pulse dwell period, and then multiplying these matrices to obtain a complete linear transformation for the entire history. In addition to this decision, speed was considered throughout the code design process. For example, data library formats and internal data handling have been implemented with modern techniques to enhance versatility without sacrificing speed.

1.3.3 Simplicity

While accuracy and speed have long been issues in the creation of engineering codes, their simplicity is of increasing importance. In this context, simplicity is an issue for both modification/maintenance and use of the code. Since ALARA is written in *C++*, it benefits from some of the philosophies of object-oriented code design. This allows the code itself to be more readable to future programmers and facilitates enhanced modularity. This modularity means that if new functionality is added to the code, it can be optimized internally with minimal detrimental effect on the existing code.

ALARA has been designed with the user in mind. Even though improved methods have existed for years, many codes have continued to use input formats which are reminiscent of punch card input entry. Furthermore, most tools in this field have been designed for the solution at a single spatial point, requiring many subsequent and slightly altered runs to get any kind of spatial information. ALARA allows the user to find the solution to an activation problem in a variety of different multi-dimensional geometries, using a flexible system to define the material properties and allowing a complicated pulsed/intermittent irradiation history and a variety of after-shutdown solution times. Furthermore, the input file can be fully commented, preventing the common difficulty of creating a long list of seemingly disconnected numbers for code input.

Chapter 2

Physical Model

Section 1.1 describes the basic activation process and shows a sample activation tree in figure 1.1. Figure 2.1 shows the same tree with annotations to define the nomenclature to be used in this chapter and others.

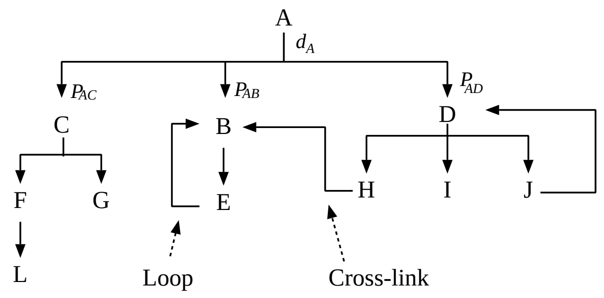


Figure 2.1: Annotated sample activation tree showing loops and cross-links.

A tree is constructed of *nodes*, each representing a single isotope, and *branches*, each representing a reaction path between the *parent* isotope and *daughter* isotope for that reaction. The top isotope in the tree will be called the *root* and each succeeding generation of reaction products will be referred to as a *rank*, giving a measure of the depth of the tree. Each isotope has a production rate, P_{ij} (the root has no production rate), dependent on the reaction path by which it was produced, and a unique total destruction rate, d_i. For decay reactions, this production rate is dependent only on the parent's nuclear data (specifically, the half-life) and not on the neutron flux. For transmutation reactions, it is a function of the parent's nuclear data and the spectral distribution of the flux. The destruction rate may be made of a combination of transmutation data, and thus the neutron flux, and decay data, depending on the parent isotope. The raw data used to form these production rates are read from large data libraries, either as decay rate/branching ratio data from decay libraries or as transmutation cross-sections from transmutation libraries. While the methods used to measure, evaluate and compile such data[18, 19] will not be discussed here, it is important to note that seemingly small changes in these data can lead to observable differences in the results of an activation calculation.

It is possible for one nucleus to undergo a series of reactions, being converted from one isotope to another and so on and eventually back to the original isotope. Loops such as this are of specific importance when modeling this physical problem. The nature of such loops is somewhat random; they can begin at any rank in the tree and can undergo any number of reactions before closing the loop. If the *order* of a loop is defined here as the number of isotopes between two occurrences of the same isotope in a loop, then the order can range from 1 to greater than 10. Loops are only physically possible during irradiation. During periods of pure decay, loops are physically disallowed for the simple reason that nuclear decay is a transition to a lower energy state. This can only be reversed by the introduction of an energy source, provided by the bombarding neutrons during irradiation. One example of a loop is

$$^{28}\mathrm{Si} \xrightarrow{(n,p)} {}^{28}\mathrm{Al} \xrightarrow{\beta^-} {}^{28}\mathrm{Si},$$

in which ^{28}Si is transmuted by a neutron reaction with the emission of a proton to ^{28}Al. ^{28}Al, in turn, decays back to ^{28}Si through the emission of a β^- particle. There are many other variations of loops, involving different nuclear reactions and with more than two isotopes involved in the loop.

A related but less important phenomenon is that of *cross-linking* of subtrees. This is caused when two different isotopes, which could each be at any rank, both undergo reactions to the same isotope. Between loops and cross-links, the tree can become quite tangled, departing from the classical tree structure known and studied in computer science.

Since most isotopes will undergo nuclear reactions, as soon as they are created by either transmutation or decay, there is a finite chance of them being transmuted to other isotopes. As a result, an activation tree can, in theory, grow to become a large connected graph including all the isotopes for which data exists, with many cross-links and loops of various orders. If many reactions are required to reach a certain isotope from the root isotope, however, their production levels will be insignificant. For the purpose of a practical numerical solution, therefore, it is necessary to truncate the tree based on some reasonable criteria.

The methods for modeling activation trees, including loop handling and tree truncation, can have a significant impact both on the type of mathematical technique employed to generate a solution, and on the accuracy of the results. Section 2.1 addresses these issues and others in the creation of activation trees.

Current designs for fusion power reactors of all types often include the necessity for pulses, from the short frequent pulses of an inertial confinement system to the long infrequent pulses of a magnetic confinement system. Other neutron-producing systems, such as experimental fusion reactors or accelerator-based neutron sources, may have more complicated irradiation histories, with varying pulsing frequencies, pulsing hierarchies, maintenance periods, etc. Furthermore, with changing conditions the flux spectrum and/or magnitude may differ from one part of the history to another. This pulsing creates an important effect[21–23] since between each pulse, the radioactive isotopes which have been created are able to decay while the stable isotopes remain unchanged. This changes the distribution of isotopes, having important implications on the reactions during the subsequent pulses. Section 2.2 will describe the approximations and assumptions used in modeling this aspect of the physical model.

Although activation calculations have traditionally been used to find all the activation products of a given material composition exposed to a given neutron flux history, this mode is not suited to the task of determining the concentrations of specific trace activation products. If certain isotopes are produced in very small quantities, the accurate determination of their concentration requires that the activation trees be allowed to grow to large sizes, and much computational effort will be wasted calculating the concentrations of uninteresting isotopes. Using many of the same modeling methods described in this chapter, small modifications can lead to a *reverse* activation calculation mode, whereby the concentrations of certain target isotopes can be calculated from the initial mixtures without calculating the concentrations of too many non-target isotopes. Section 2.3 describes the adaptations necessary for such a reverse calculation.

The last section of this chapter addresses some of the software design and implementation issues related to the various modeling methods discussed for the physical problem, including data structures, computational efficiency and optimum memory usage.

2.1 Activation Tree Modeling

As mentioned above, the theoretical activation tree can become a large connected graph including every isotope for which data exists. This could, in principle, be converted into a large matrix of production and destruction rates, and solved directly for all the initial isotopes at a given point in space. This is not practical, however, both due to the size of the matrix and the mathematical methods available for solving such a matrix. For a typical fusion activation library, the matrix would be roughly 2000×2000, sparsely filled (only about 1 or 2%) and would exhibit little pattern (*i.e.* banded or triangular matrices). By introducing a variety of concepts, this problem can be reduced to a finite number of tractable sub-problems.

There are two primary issues related to the modeling of activation trees: loop handling and tree truncation.

2.1.1 Tree Straightening and Loop Handling

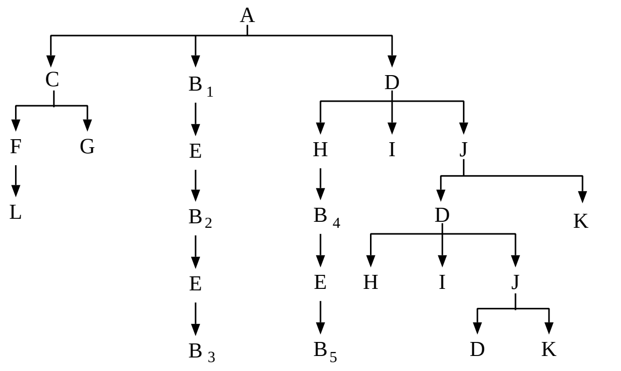


Figure 2.2: Fully straightened and unlinked reaction tree.

One alternative for loop handling is to explicitly model them in the mathematical representation of the physical problem. In the rate equation for node i, there is a production term from some node $j > i$ and in the matrix formulation shown in equation 1.1, the transfer matrix, $\mathbf{A}$, has terms above the diagonal. While time-step based ODE solvers can be used with this method,[6] the non-triangular nature of this matrix limits the matrix methods requiring some form of matrix exponential method,[8] which can be computationally complex and/or prone to numerical error. The philosophy behind the use of these exact physical modeling methods is that loops may be significant in some cases, and thus should be included.

The alternative is to introduce a method of *tree straightening*, converting the connected graph into a traditional n-ary tree. Figure 2.2 shows a straightened tree representation of the same tree shown in figure 1.1. Tree straightening is accomplished by the introduction of what will here be called *partial-contribution* nodes, or simply *pc*-nodes. While the basic physical model described above only allows for one occurrence of each isotope in a tree, a *pc*-node represents an isotope which may occur elsewhere in the tree, and thus the production of this node gives only a partial contribution to the total production of the isotope which it represents. By introducing *pc*-nodes, a somewhat larger system is created, but its solution is more simply achieved. After the full solution of the problem, the results at each *pc*-node are collapsed into unique isotopes, with the contribution from each being accounted for appropriately.

After all the loops and cross-links have been removed, each tree is traversed in a *depth-first search*[a], creating a number of independent linear chains (see Figure 2.3). Although the term linear chain traditionally refers to models that do not include any loops, it will be used here to refer to physical models which have implemented tree straightening. The conversion to linear chains results in many sub-problems, each with a matrix size equal to the chain's rank, rather than a single sub-problem with a matrix size equal to the number of nodes in the tree.

In the mathematical representation of this alternative, the rate equation for any node i will only have a production term from the preceding node $i-1$. In the matrix formulation, therefore, $\mathbf{A}$ is always (lower) bidiagonal. However, because loops cause more than one node to represent the same isotope, there are guaranteed to be at least two equations with the same diagonal element. Since the diagonal elements of a bidiagonal matrix are the eigenvalues, $\mathbf{A}$ is guaranteed to be defective. Thus, even though the bidiagonal nature permits simpler matrix methods than with non-triangular matrices, the defective nature of the matrix still prevents the use of the traditional analytic methods based on the Bateman equations.

[a]A depth-first search is an algorithm which moves deeper into a tree as far as it can go before backtracking and moving down a different path.

In the past, some methods[12–14] have simply ignored all but the simplest of loops, 1st order loops at rank 1, and analytically handled these simple loops as special cases. Thus, for the majority of the problem, only bidiagonal non-defective transfer matrices are solved, permitting the implementation of purely analytical methods based on the Bateman equations. The decision to ignore most loops is based on the philosophy that in most problems, only the first few generations contribute significantly to the total solution, and thus higher order loops and/or loops occurring deeper in the tree contribute little to the solution.

A compromise is to include a finite number of loop iterations. As with any approximation based on successive corrections, the solution approaches the exact result at the limit of an infinite number of corrections, where each of the loop iterations is considered as a correction to the solution without loops,

$$\begin{tikzcd}
A \arrow[d] & A \arrow[d] & A \arrow[d] & A \arrow[d] & A \arrow[d] & A \arrow[d] & A \arrow[d] & A \arrow[d] & A \arrow[d] & A \arrow[d] \\
C \arrow[d] & C \arrow[d] & B_1 \arrow[d] & D_1 \arrow[d] & D_1 \arrow[d] & D_1 \arrow[d] & D_1 \arrow[d] & D_1 \arrow[d] & D_1 \arrow[d] & D_1 \arrow[d] \\
F \arrow[d] & G & E_1 \arrow[d] & H_1 \arrow[d] & I_1 & J_1 \arrow[d] & J_1 \arrow[d] & J_1 \arrow[d] & J_1 \arrow[d] & J_1 \arrow[d] \\
L & & B_2 \arrow[d] & B_4 \arrow[d] & & D_2 \arrow[d] & D_2 \arrow[d] & D_2 \arrow[d] & D_2 \arrow[d] & K_2 \\
& & E_2 \arrow[d] & E_3 \arrow[d] & & H_2 & I_2 & J_2 \arrow[d] & J_2 \arrow[d] & \\
& & B_3 & B_5 & & & & D_3 & K_1 &
\end{tikzcd}$$

Figure 2.3: Separated linear chain representation of activation tree.

A conservative analysis of the error associated with this approach,[17] including the impact of ignoring loops altogether (0 corrections), is made possible by comparing the approximate solution to the analytical solution for a 1st order, rank 1 loop, using an increasing number of corrections. Since production rates are always finite, it is intuitive that higher order loops contribute less than lower order loops originating at the same rank. Choosing a rank 1 loop is simply a matter of convenience in calculating the exact solution, but the error can be considered as the relative error associated with ignoring a 1st order loop at any rank in the tree.

Table 2.1: Relative Errors when Using the Straightened Loop Method

(n,p) Reaction	# of corrections				
	0	1	2	3	4
$^{13}C(n,p)^{13}B$	4.06e-05	8.23e-10	1.10e-14	1.12e-16	1.12e-16
$^{16}O(n,p)^{16}N$	0.000351	6.15e-08	7.20e-12	5.62e-16	1.12e-16
$^{28}Si(n,p)^{28}Al$	0.00243	2.95e-06	2.38e-09	1.45e-12	1.02e-15
$^{49}Ti(n,p)^{49}V$	0.000247	3.05e-08	2.51e-12	0	1.21e-16
$^{56}Fe(n,p)^{56}Mn$	0.00103	5.35e-07	1.85e-10	4.77e-14	0

To ensure that the results are physically significant, a variety of 1st order loops were found in the existing nuclear data.[20] In particular, since decay tends to lead to higher production rates than transmutation, loops of the form

$$A \xrightarrow{(n,p)} B \xrightarrow{\beta^-} A,$$

were chosen (instead of loops with two nuclear reactions, for example). Table 2.1 shows a summary of the errors for 5 different loops after $10^{10}s \approx 317y$ and up to 4 corrections, and figure 2.4 shows how this error is a function of the steady state irradiation time used in this analysis for the worst case,

$$^{28}\text{Si} \xrightarrow{(n,p)} {}^{28}\text{Al} \xrightarrow{\beta^-} {}^{28}\text{Si}.$$

While it is clear from both table 2.1 and figure 2.4 that ignoring loops completely can introduce observable error to the solution, they also show that a finite number of corrections can reduce the error to acceptable levels. With only two corrections, the error is significantly reduced to be less than the accuracy of most calculations, while 4 corrections brings the error to within the precision of an IEEE double precision calculation. It is important to recognize the importance of this measure being a *relative*

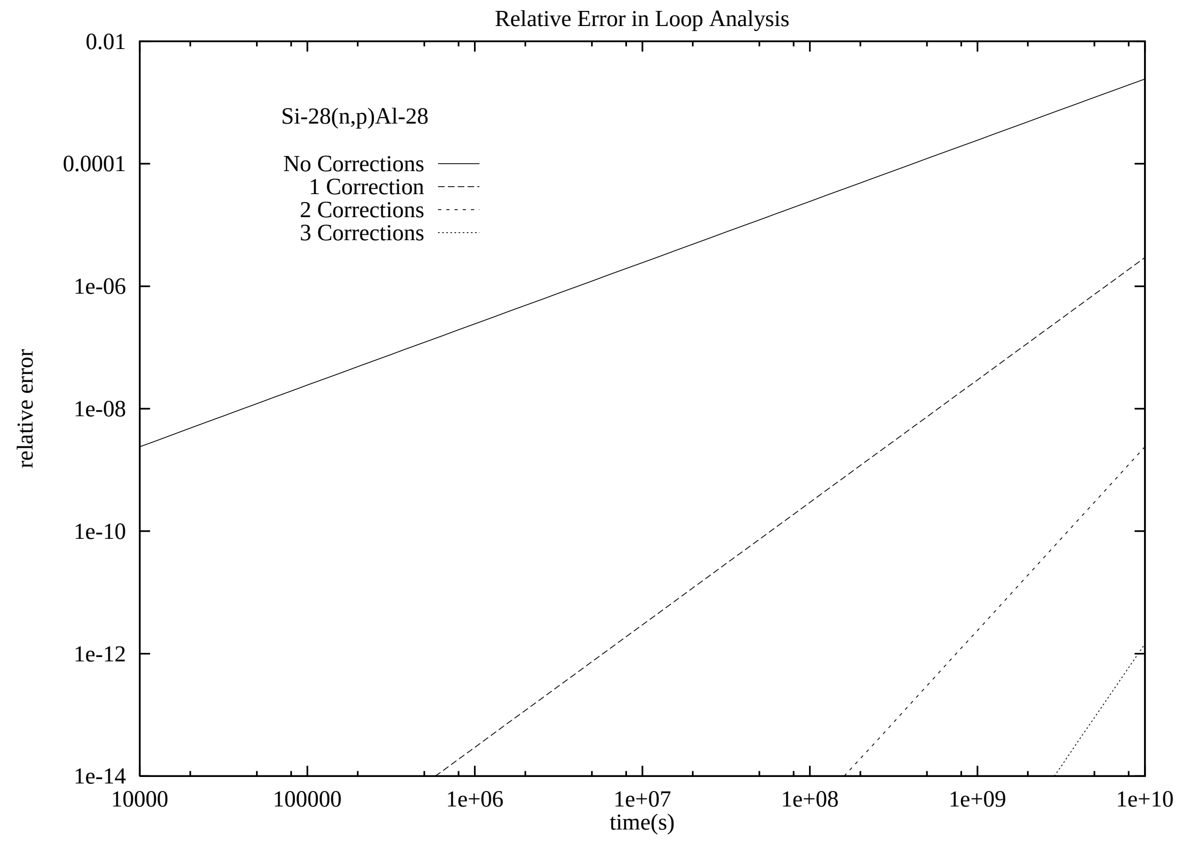


Figure 2.4: Relative error in calculation of ^{28}Si as part of (n, p) loop.

error. It has no effect on the precision of the results (discussed further in the next section), but only on the accuracy of the results.

To determine the number of corrections needed in any particular problem, it is sufficient to treat the straightened loops like any other part of the activation tree and invoke the same truncation criteria as are used elsewhere. When implemented in this way, the precision of the loop solutions is the same as the precision of the solutions to the rest of the problem.

Relative to other methods, tree and loop straightening can allow faster and more accurate mathematical methods without jeopardizing accuracy in the physical model.

2.1.2 Truncation of Activation Trees

On the surface, the concept of truncating the large trees created by modeling the physical system for these calculations is a simple one: truncate the tree once the contribution from the nodes is negligible compared to the total result. In practice, however, this is a delicate process which deserves some discussion.

As with loop handling, a variety of alternatives have been used to accomplish the truncation of activation trees in previous applications. The simplest method defines a maximum allowable rank for the tree, truncating the tree as soon as this rank is reached. Since radioactivity is often the most important result and nuclear decay branches tend to create more significant sub-trees than nuclear reaction branches, this method can be improved somewhat by limiting only the number of transmutation generations, and allowing decay branches to continue until they reach a stable isotope.[7] This method tends to be inconsistent in its estimation of the importance of the contributions of the various nodes. Some sub-trees included by such a system would contribute negligibly, if at all, to the final result, while other more significant sub-trees would be truncated too early.

A more consistent method is based on an estimate or calculation of the actual contribution of a given node in the tree to the final solution. If the contribution is too low, the tree is truncated here, otherwise the tree continues to grow. This raises two primary issues to be considered:

- how can the production of the isotopes at a certain rank in the chain be calculated or estimated without solving the whole problem, and,
- how can the importance of that production be measured.

When using time-step based ODE solvers, the first issue is moot. During the course of the solution, equations can be added to and dropped from the system as new isotopes are produced and the concentrations of isotopes become insignificant, respectively.[6]

For matrix methods, the actual contribution of a node can only be calculated by completing a full solution of the chain leading to that node, requiring the solution of the entire irradiation history each time the tree grows. As the variations in the neutron flux spectra will affect the production rates giving each point in space a different real activation tree, such a calculation should, in theory, be carried out for each point in space. This would drastically slow down the calculation.

As a first enhancement, this calculation, refered to here as a reference calculation, is performed only once each time the tree grows, using a flux which is somehow representative of all the spatial points which contain the initial isotope (see section 2.4). If every point in space has a unique set of initial isotopes, this offers no savings, but as most problems have many spatial points with identical sets of initial isotopes, the savings can be significant. Furthermore, if the flux is chosen properly, this approximation will be conservative, and would tend to produce chains which are too long rather than too short.

What is the best flux to represent all the spatial points which share an initial isotope? Since higher fluxes will tend to maximize the amount of transmutation from one isotope to another, the obvious choice is some flux which is a maximum bound for the problem. Since the flux is group-wise and it is possible that one spatial point will have the highest fast flux while another point has the highest slow neutron flux, the best choice for a bounding flux should be the group-wise maximum flux of all the points which share an initial isotope. This reference flux can then be used to solve the problem for a particular chain as it is being created.

In order to decide what result this reference calculation should provide and how this should be interpreted, it is first necessary to understand the nature of the approximation introduced by truncating the activation trees. In the real system, there will always be at least as many atoms as in the initial isotope, with the possibility of increasing that number through the accumulation of light ions emitted by nuclear reactions. In the modeled system, however, when an activation tree is truncated the destruction rate of the last node in the tree represents a pathway for atoms to pass out of and be lost from the model. The goal of any truncation concept should therefore be to minimize the *relative atom loss* from the system. This approach has the natural consequence of limiting the error in the production of any one isotope. If the relative atom loss is limited to t atoms per initial atom, and there are n truncation points in the tree, $n \cdot t$ atoms per initial atom are lost from the model. In the worst (and entirely unphysical) case that all the lost atoms are transmuted to the same isotope in the real activation tree, the production of this isotope will have an error of $n \cdot t$ atoms per initial atom.

Previous implementations of this type of truncation concept have calculated the relative inventory of the node in question, rather than the relative atom loss through that node.[12] This can result in severely premature truncation if that node has a very high destruction rate (such as a short decay half-life). The relative inventory of that node may be very low because the majority of atoms produced at this node have been further converted by transmutation or decay out of the model.

A better implementation calculates the relative inventory of the entire sub-tree rooted in the node in question. Fortunately, this is quite easily implemented. Since all atoms that pass into a node will end up either as part of that node's inventory or as part of the inventory of that node's sub-tree, it is only necessary to calculate how many atoms are passed into the node over the course of the irradiation history. This calculation is identical to the normal activation calculation, but temporarily preventing that node from being destroyed ($d_i = 0$). This value can then be compared to a user specified tolerance, interpreted as the maximum atom loss allowed in any single chain.

Although both transmutation and decay are mechanisms of atom loss, the former is only possible when there is a neutron flux present while the latter occurs throughout the operation lifetime as well as after the shutdown of the device. This difference is important since the after-shutdown lifetimes (or cooling times) can be much longer than the operation lifetimes and it is during these times that the activity is often most important. It is therefore necessary to compare the relative atom loss both at shutdown and at each after-shutdown time. If the relative atom loss is less than the tolerance at shutdown, but greater than the tolerance at any of the after shutdown times, it is a potentially important atom loss path during shutdown. It is here that we can distinguish between atom loss mechanisms. Since the only atom loss mechanism after shutdown is decay, only subsequent decay branches are important, even if the relative atom loss is greater than the tolerance.

Combining all these concepts into a single truncation philosophy gives the following algorithm. First, any relative atom loss which exceeds the tolerance at shutdown will result in a continuation of the chain. Second, if the relative atom loss is less than the tolerance both at shutdown and at all after-shutdown times, then the chain should be completely truncated at this point. Finally, if the relative atom loss is less than the tolerance at shutdown and greater at some after-shutdown time, then all transmutation branches in that sub-tree should be truncated but decay branches followed.

While faster approximations could be made to conservatively estimate the relative atom loss, the conservatisms which seem appropriate for the physical modeling can lead to the physical model being too large. Although time may have been saved during the physical modeling, the full solution must still be carried out on an unnecessarily large system. Since many problems must solve each chain at many spatial points, the savings made on a single truncation calculation are more than compensated by the losses during the multiple solutions of that chain at different neutron flux levels.

This truncation approach affords some rudimentary error estimates for the results. Using an analogy to experiment, the user specified tolerance provides a measure for the *precision* of the calculation. The smallest possible correction and hence the largest possible error for the results for any one isotope is this truncation tolerance. This will also be the dominant source of physical modeling error in the result. Since these same truncation rules are used indiscriminately for truncating straightened loops, the error from truncation will always be greater than the error caused by not using more corrections to the loop. Thus, following the analogy to experiment, the *accuracy* of loop solutions is affected by the number of corrections, while the precision is affected by truncation tolerance. Since the number of corrections is determined indirectly by the truncation tolerance, this methodology provides the most consistency across the entire problem. It should also be noted that this measure of the truncation error in the physical model is an upper bound since a group-wise maximum flux is being used for the calculations. In many spatial regions, the actual production in the final solution may be many orders of magnitude less than that of the truncation calculation.

The full implementation of this philosophy does have detrimental effects on the speed of the solution. The most significant drag is caused by the full pulsing solution of the chain for each truncation calculation. An alternative which has been implemented in the past is to combine the reference flux concept with that of a reference time, a representative steady-state simulation time to use only for truncation calculations returning to the exact pulsing solution when performing the final solution. When using this alternate method, it is important to understand the full implications. In particular, even if the chosen reference time approximates the operation history well, the reference calculation includes no after-shutdown history, a period in which many isotopic compositions may change.

Another source of drag is the calculation of completely negligible results at the truncation point. Considering the precision and extent of the available data, it is possible that the atom loss at one node clearly indicates that the chain should be continued, while the atom loss at one of its daughters is many orders of magnitude below the truncation limit. While it is obvious that the chain should be truncated, the full solution of this *pc*-node will probably lead to a negligible contribution, but at a significant computational cost. A second user defined tolerance, known as an *ignore tolerance*, can be used to determine when a truncation point should be ignored completely and the chain creation procedure should continue without performing the complete solution of this *pc*-node. To ignore all truncation points, an ignore tolerance of 1 could be used and to ignore none, an ignore tolerance of 0.

2.2 Irradiation History Representation

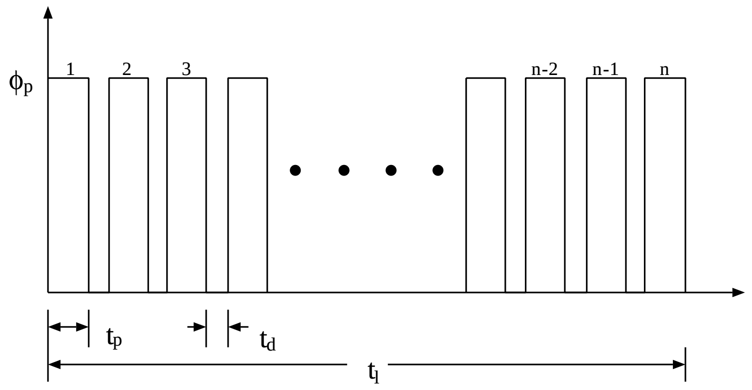


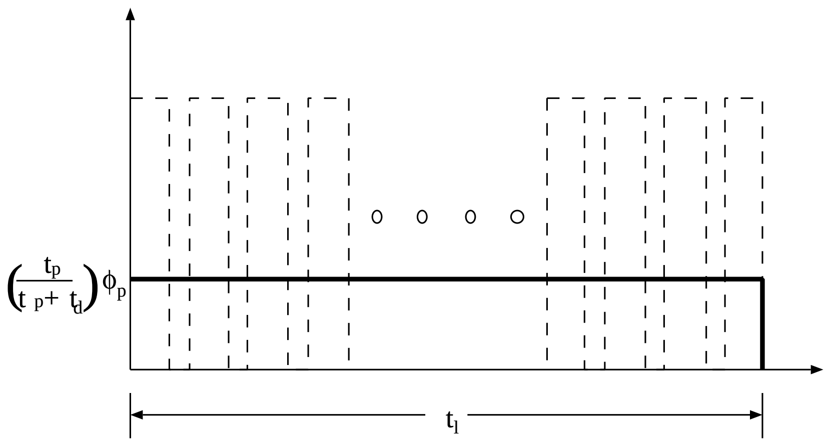
Figure 2.5: Sample pulsed irradiation history.

Although continuously varying neutron flux spectra and levels are not expected, many systems modeled in activation calculations will have intermittent and/or pulsed operation. At the very least, the systems will have to be shutdown at regular intervals for maintenance. Additionally, many kinds of fusion power systems are expected to have pulsed operation, whether it be an intrinsic part of the concept (such as in inertial fusion energy systems) or part of the technical requirements for the system (such as in magnetic fusion energy systems). The method used to model this intermittent and pulsed operation has important implications on the generic accuracy of the solution and the type of mathematical method to use.

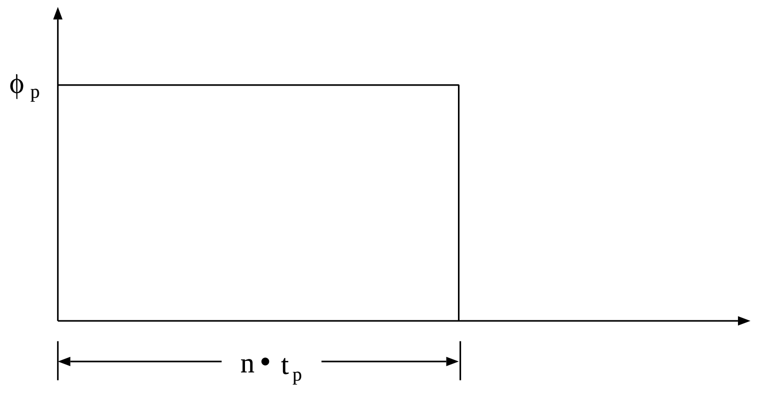
Physically, the pulsed nature results in build up of transmutation products during the operation pulses and decay of those isotopes during the "dwell" periods between pulses (see figure 2.5). For isotopes of certain half-lives relative to the ratio of dwell time to operation time, the pulsing representation can have a profound impact on the final calculated number density. The most accurate solutions will result from methods which allow for this transient behavior during the dwell times to correctly assess the radioactivity.

Again, there are a variety of methods historically implemented to model this pulsing. The simplest methods use a steady-state approximation, either averaging the flux over the full lifetime (figure 2.6(a)) or squeezing the pulses together and removing the dwell times (figure 2.6(b)).

Although one approximation is more valid than the other in certain cases, Sisolak *et al.*[21] showed that both pure steady state approximations above will incorrectly calculate the activity for certain domains of short- and medium-lived isotopes by orders of magnitude. The first approximation, often used for the analysis of magnetic confinement systems, is reasonable for long-lived isotopes that would not decay appreciably during the dwell times in the real system. Their final density is, therefore, the result of



(a) Average flux approximation.



(b) Collapsed pulses approximation.

Figure 2.6: Two popular steady state approximations to pulsed irradiation histories.

a virtually monotonic build-up throughout the operation times. On the other hand, for short-lived isotopes that would reach secular equilibrium within each pulse and decay completely during the dwell time, the exact result is equal to this secular equilibrium value. This value is proportional to the flux magnitude, which, for this approximation has been scaled to conserve the total fluence throughout the lifetime. Using the second approximation, this equilibrium value is the same as in the true pulsing problem for short-lived isotopes. On the other hand, for medium-lived isotopes, particularly those with half-lives greater than the dwell time between pulses but less than the total system lifetime, the unmodeled incomplete build-up and decay in the real system during the pulses and dwell times, respectively, result in significant over-calculation of the activity.

These approximate approaches are favored by codes that use time-step based ODE solvers, since the pulses impose additional restrictions on the length of the time step which may be used. In many cases, the first approximation can be successfully improved by modeling the majority of the pulses as a single steady state period, with a scaled flux, and then modeling the last few pulses exactly. This ensures the correct flux level for the determination of the secular equilibrium level of the short-lived isotopes.

Alternatively, if matrix methods are used, equation 1.1 can be solved for each pulse and dwell time, and then appropriately multiplied and raised to a power to represent the entire history (see section 3.5). Thus, the intermittent and/or pulsed operation is modeled exactly with no approximation.[22,23]

These exact methods become even more valuable when the operating history has more variation than simple repeated pulses. The first type of variation is in the dwell time between pulses, such as might occur when the pulsing schedule of an experimental facility follows the standard working hours of the staff. Figure 2.7 shows such a history with parameters give in table 2.2. When using matrix methods and exact history modeling, the activation equation is solved once for the pulse and for each dwell period. These matrices are then multiplied in the correct sequence to generate a transfer matrix solution for the entire history.

Changes in the flux spectra or irradiation times, due perhaps to a modification in the system's configuration or performance, can further complicate the history. In this case, the history may consist of two or more schedules, as described in table 2.2, where the flux and/or the characteristic times are different in each of the successive schedules.

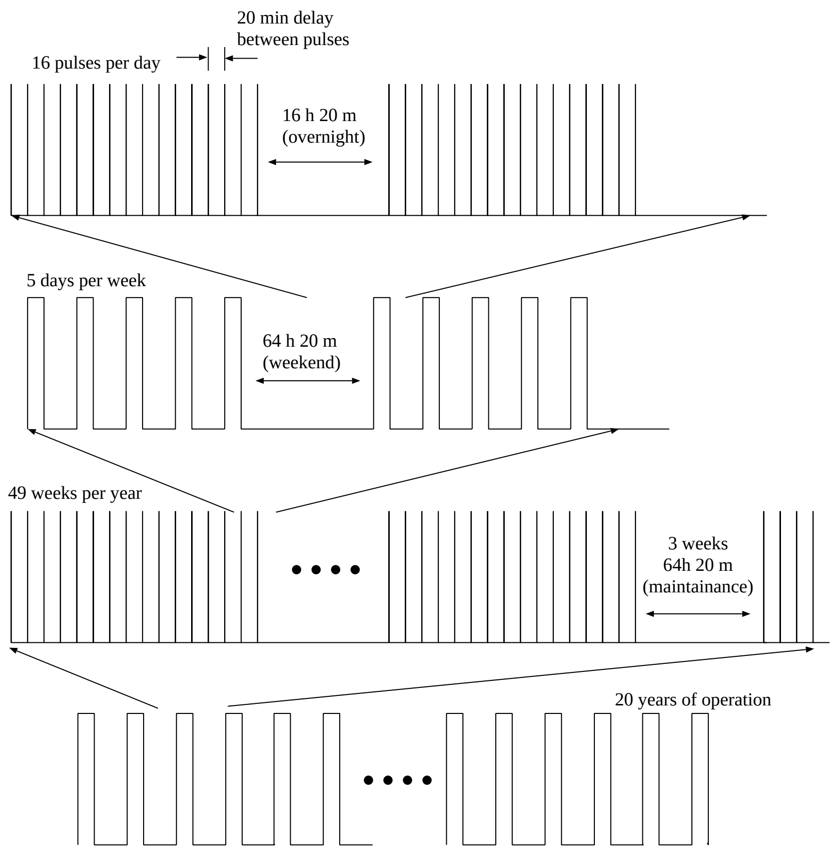


Figure 2.7: Example pulsing schedule for experimental device.

Table 2.2: Example pulsing schedule for experimental device.

Description	Time	# of Pulses
Pulse length	10 min	
Operation dwell	20 min	16 (half-hour segments)
Nightly dwell	16 h 20 min	5 (work days)
Weekend dwell	64 h 20 min	49 (weeks without maintenance)
Annual Maintenance	3 weeks 64 h 20 min	5 (years)

Finally, it may be desirable to have a number of different schedules which, as a set, are repeated at regular intervals. For example, when modeling material that is exposed to different neutron fluxes in a repeating sequence, such as the high-Z material of an inertial confinement fusion target being recycled.[24] The first time this material is introduced to the system is as part of a target, and thus receives a very large flux. Due to the explosion of this target during this initial reaction, the material is deposited on the facility's walls. It will be subjected to a number of pulses with lower flux (and different flux spectrum) before it is removed from the system, spends some time being reprocessed into a new target, and goes through the cycle again. Figure 2.8 shows such a history, which can be modeled efficiently and exactly with matrix methods. The activation equation is solved once for the primary pulse, once for the secondary pulse, once for the dwell between pulses, and once for the reprocessing dwell time. These matrices can then be multiplied and raised to a power as appropriate to model the entire history.

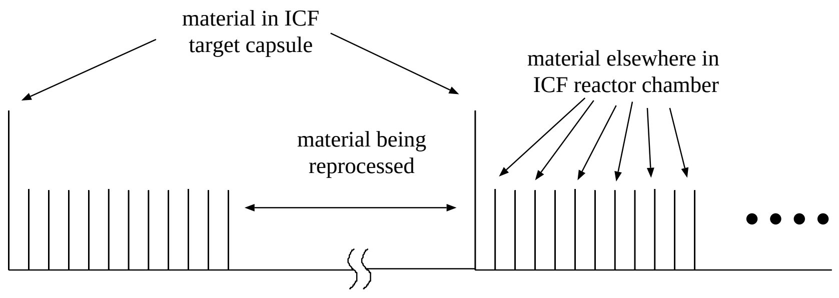

material in ICF target capsule
material elsewhere in ICF reactor chamber
material being reprocessed

Figure 2.8: Irradiation history for recycled ICF target material.

It is important to note, that the more complicated the irradiation history becomes, the slower the solution will be because of the necessity to solve more individual versions of the activation equation. In fact, in the limit of a continuously changing flux, this exact representation becomes physically identical to a time-step based method with the time-step size dictated by the characteristic time of the flux history, but will likely be computationally slower because of the mathematical method being employed at each time step.

2.3 Reverse Calculations

An activation calculation may be performed to determine the relative production of trace quantities of specific isotopes. If the production pathways for these few isotopes are very long, the tree will become very large, and the majority of the results will not be relevant to the specific target isotopes.

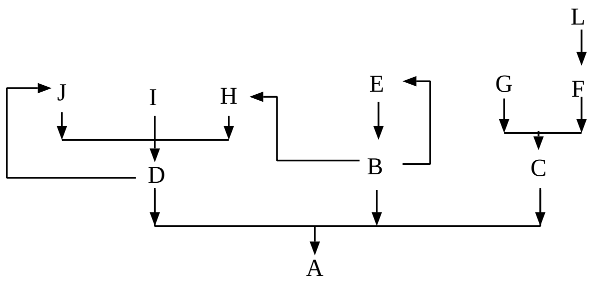


Figure 2.9: Sample reverse calculation tree.

One solution is to build the activation tree in reverse, starting with the target isotope and adding parent isotopes, as shown in figure 2.9. For the most part, if the forward calculation is wisely implemented, the physical modeling of the problem requires only a few enhancements, and the mathematical methods are identical.

The easiest way to perform a reverse calculation is to use a reversed nuclear data library. Where a normal data library is indexed by the parent isotope, giving a table of reaction paths and cross-sections for each one, a reversed library is indexed by the daughter isotope, giving a table of production paths and associated cross-sections. Once a library is reversed, extracting data for each reverse *pc*-node is similar to extracting data from a normal library for each *pc*-node of a forward calculation.

Perhaps the most significant difference, however, is the necessity to reinterpret the truncation algorithm. First, instead of maximizing the accuracy of the solution across all isotopes by minimizing the relative atom loss from the modeled system, the goal of the reverse truncation algorithm is to maximize the accuracy of the production of the target isotope. Therefore, it is more appropriate to use a relative production calculation than a relative atom loss calculation.

Second, and more important, the interpretation of the relative production result in comparison to the truncation and ignore tolerances must be reassessed. As with the forward calculation, if the relative production is larger than the tolerance at shutdown, it is clear that the chain must continue to grow. Also similar to the forward calculation, if the relative production is less than the truncation tolerance both at shutdown and at all the cooling times, the chain should be truncated. However, when the relative production is less than the truncation tolerance at shutdown and greater at some after-shutdown time, it is no longer appropriate to truncate all the transmutation branches. This result indicates that the parent of one of the decay branches in the chain accumulates during the irradiation history, after which a significant fraction decays during the cooling time. The decay branch at which this occurs may not be related to the isotope being tested at the root of the chain. (Of course, when this isotope is at the root of the chain, this will be the outcome of the truncation comparison.) It is therefore necessary to simply continue the chain in this case, since the next parent may still lead to sufficient accumulation of the relevant decay parent that the relative production of the target isotope is greater than the truncation tolerance at a certain cooling time.

The last difference is the interpretation of the final matrix solution. In a forward calculation, the first column of the solution matrix represents the relative production of each isotope from the root isotope, and these results are used to sum the total production of each isotope from each root. In a reverse calculation, the last row of the matrix is used, as it represents the relative production of the target isotope from each of the isotopes in the chain.

2.4 ALARA Implementation Summary

This section will discuss the details of the exact implementation of the physical modeling methods in ALARA.

After reading the input information, ALARA's first task is to process all the mixture definitions and create a list of unique initial isotopes, cross-referenced with the mixtures in which they exist, and sorted by increasing atomic number. The activation problem is then solved completely for each root isotope in turn.

When calculating the induced activation of a whole system, the material composition and neutron flux spectrum will vary from location to location in the device. A structural region of a problem may contain some variety of steel while a coolant region might have water and a breeding region would contain lithium. The initial isotopes, and thus the reaction trees, will therefore be very different for each region. Further, for each point of interest, the spectral distribution and magnitude of the fluxes will be different. Thus, even for identical trees from the same material composition, the production rates for each isotope will vary from point to point.

A reference flux is found by determining the group-wise maximum flux across all the spatial points which include this root isotope. Beginning with this root isotope and using the reference flux, the activation tree is modeled as linear chains created by a depth-first search of the straightened activation tree. Using this technique, ALARA must never store information about more than one chain at any time. As each chain is truncated, it is immediately solved for each spatial point, now using that point's flux, in which the root isotope exists. Various parameters of the chain are used to sum only the contributions from the *pc*-nodes which will not be part of the next chain in the depth-first search. This ensures that no *pc*-node is included more than once and also allows the chain to be completely discarded, making that memory available, as it will no longer be needed in the calculation.

Unlike many other activation codes, ALARA does not use a fixed table of nuclear reaction codes to determine the reaction type and daughter product. Rather, it expects all this information to be included in the nuclear data library. ALARA is not limited to any conventional set of nuclear reactions. This has already proved important when being used to simulate the activation in a system with intermediate neutron energies (up to 55 MeV)[25] and many more reaction channels per parent ($\lesssim$ 120) than most standard libraries ($\approx$ 35) and most standard reaction tables ($\lesssim$ 100).

The relative atom loss truncation method outlined above, including an ignore tolerance, is implemented in ALARA using the reference flux established for each root isotope. The relative atom loss calculation is performed for the exact irradiation history, including the various after-shutdown cooling times. As each *pc*-node is added to the chain during the depth first search, it is tested according to the flowchart shown in figure 2.10. It is worth noting that, contrary to some other implementations,[13,14] this truncation method allows a chain to be fully truncated on a radioactive isotope, but only if it has been determined that the relative atom loss through that isotope is less than the truncation tolerance at shutdown and all after-shutdown cooling times. It is, in fact, possible to ignore a radioactive isotope, if its relative atom loss is below the ignore tolerance at all times of interest.

The modeling of the irradiation history in ALARA allows for a very versatile hierarchy of schedules and sub-schedules limited only by the computer resources (both system memory and calculation time). Each schedule can have a list of sub-schedules, where each sub-schedule can be either a simple pulse, with an associated flux definition, or another schedule. Each sub-schedule, regardless of which type, is subjected to a pulsing hierarchy such as described in table 2.2 and separated from the next sub-schedule by

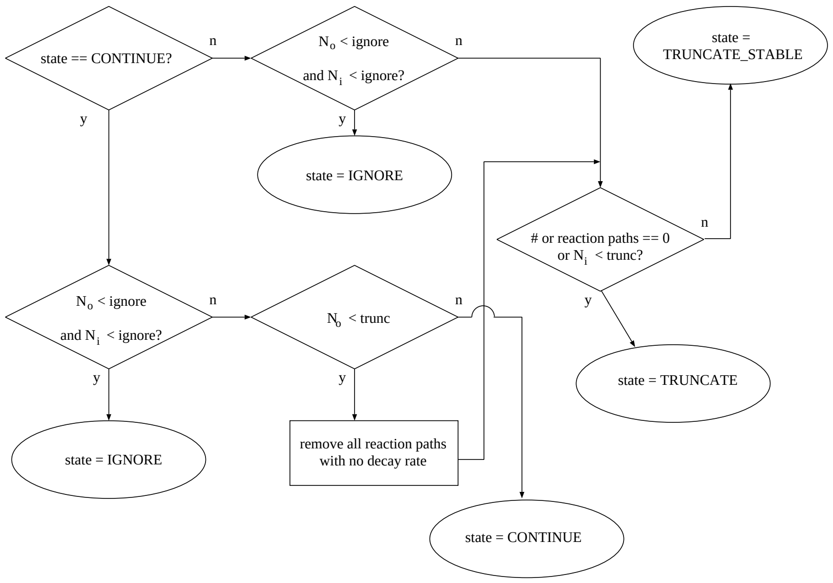

state == CONTINUE?
n
y
N_o < ignore and N_i < ignore?
n
y
state = IGNORE
state = TRUNCATE_STABLE
# or reaction paths == 0 or N_i < trunc?
n
y
state = TRUNCATE
N_o < ignore and N_i < ignore?
n
y
state = IGNORE
N_o < trunc
n
y
remove all reaction paths with no decay rate
state = CONTINUE

Figure 2.10: Flowchart for truncation algorithm.

a dwell time. When ALARA solves the problem, the schedule hierarchy, analogous to an *n*-ary tree, is traversed in a depth-first search. Thus, each schedule's list of sub-schedules is solved, and the product of all the sub-schedule solution matrices and dwell time matrices is taken as the solution matrix of the schedule.

Chapter 3

Mathematical Technique and Theory

The mathematical problem resulting from the physical system described in Chapter 2 is, at first glance, a very simple one. However, the calculation of matrix exponentials, such as the solution to the activation equation

$$\vec{N}(t) = e^{\mathbf{A}t}\vec{N}_o, \tag{1.2}$$

are known to be difficult in general. An extensive study of nineteen different methods found them all to be “dubious”, saying that “some of the methods are preferable to others, but that none are completely satisfactory.”[26]

If the original activation tree (without removing cross-links and straightening loops – Figure 2.1) is converted directly to its mathematical equivalent, the result is a compact but potentially stiff system of linear first order ordinary differential equations [ODE's],

$$\begin{aligned}\dot{\vec{N}}(t) &= \mathbf{A}\vec{N}(t)\\ &= \begin{bmatrix} -d_1 & P_{2\to1} & P_{3\to1} & \cdots & \cdots & P_{l\to1}\\ P_{1\to2} & -d_2 & P_{3\to2} & \cdots & \cdots & P_{l\to2}\\ P_{1\to3} & P_{2\to3} & -d_3 & \cdots & \cdots & P_{l\to3}\\ \vdots & \vdots & \vdots & \ddots & & \vdots\\ \vdots & \vdots & \vdots & & -d_{l-1} & P_{l\to l-1}\\ P_{1\to l} & P_{2\to l} & P_{3\to l} & \cdots & P_{l-1\to l} & -d_l \end{bmatrix} \cdot \begin{bmatrix} N_1\\ N_2\\ N_3\\ \vdots\\ \vdots\\ N_l \end{bmatrix},\end{aligned} \tag{3.1}$$

where:

$$\begin{aligned}\vec{N} &\equiv \text{number densities, } N_i\text{, of all isotopes}\\ d_i &\equiv \text{destruction rate of isotope } i\\ P_{i\to j} &\equiv \text{production rate of isotope } j \text{ from isotope } i.\end{aligned}$$

After loop straightening and cross-link removal is performed (see figure 2.2), the result is a somewhat larger, simpler set of ODE's:

$$\begin{aligned}\dot{\vec{N}}(t) &= \mathbf{B}\vec{N}(t)\\ &= \begin{bmatrix} -d_1 & 0 & 0 & \cdots & \cdots & 0\\ P_{1\to2} & -d_2 & 0 & \cdots & \cdots & 0\\ P_{1\to3} & P_{2\to3} & d_3 & \cdots & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & & \vdots\\ \vdots & \vdots & \vdots & & -d_{m-1} & 0\\ P_{1\to m} & P_{2\to m} & P_{3\to m} & \cdots & P_{m-1\to m} & -d_m \end{bmatrix} \cdot \begin{bmatrix} N_1\\ N_2\\ N_3\\ \vdots\\ \vdots\\ N_m \end{bmatrix}.\end{aligned} \tag{3.2}$$

This lower triangular matrix is quite sparse with a maximum of two entries in each row since each *pc*-node has only one production path and one total destruction rate.

Finally, if this physical model is further broken into the previously described linear chains (figure 2.3), many small sets of ODE's are created with special simplifying characteristics,

$$\begin{aligned}\dot{\vec{N}}(t) &= \mathbf{C}\vec{N}(t) \\ &= \begin{bmatrix} -d_1 & 0 & 0 & \cdots & \cdots & 0 \\ P_{1\to 2} & -d_2 & 0 & \cdots & \cdots & 0 \\ 0 & P_{2\to 3} & -d_3 & \cdots & \cdots & 0 \\ 0 & 0 & P_{3\to 4} & -d_4 & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & 0 \\ 0 & 0 & 0 & \cdots & P_{k-1\to k} & -d_k \end{bmatrix} \cdot \begin{bmatrix} N_1 \\ N_2 \\ N_3 \\ \vdots \\ \vdots \\ N_k \end{bmatrix}.\end{aligned} \tag{3.3}$$

The bidiagonal nature of these matrices is an important factor in subsequent derivations and calculations.

In all cases, the generic solution takes the form

$$\vec{N}(t) = \mathbf{T}\vec{N}_o(t), \tag{3.4}$$

where $\mathbf{T}$ is the exponential of the matrices $\mathbf{A}$, $\mathbf{B}$, or $\mathbf{C}$, depending on which method is used (*e.g.* $\mathbf{T} = e^{\mathbf{A}t}$).

It is interesting to compare the sizes of these three matrices as it gives some initial insight into the efficiency of the solution. To compare the size of the original tree, l, with that of the straightened tree, m, is not easy. Since the physical conversion is to convert cross-links and loops into pc-nodes, it is clear that $m > l$; however, the severity of this inequality is difficult to determine. Nevertheless, an approximate comparison between k and m can be made. Since $\mathbf{B}$ represents a true tree structure, it can be analyzed by assuming that it is a balanced n-ary tree, that is, assuming that every pc-node in the tree has exactly n branches. Since k is the depth of such a tree, there will be n^{k-1} chains representing the $m = \frac{n^k - 1}{n-1} \approx O(n^{k-1})$ nodes of the tree. The computational complexity of the mathematical operations performed on these matrices are generally at least of the order of the square of the matrix dimension, and often the cube. Thus, for matrix $\mathbf{B}$, the mathematical costs will be at least $O(n^{2k-2})$ and possibly $O(n^{3k-3})$ or higher. On the other hand, for the n^{k-1} matrices $\mathbf{C}$, the mathematical costs will be $O(k^2 n^{k-1})$ or $O(k^3 n^{k-1})$. This very rough analysis shows that for mathematical operations of order x, as long as $n^{1-\frac{1}{x}} > k^{\frac{1}{k-1}}$, the linear chain method will be more efficient. While it may be difficult to visualize this relationship, it can be used to define some limits.

If the mathematical operations have second order computational complexity, the linear chain method is always more efficient if $n \geq 4$ (there are at least 4 branches per pc-node). If this complexity increases to third order, the physical model only requires $n \geq 3$ for linear chains to be more efficient. From another perspective, if the number of branches per pc-node is fixed at $n = 2$, and chain depth, $k = 3$, operations with 5^{th} order complexity are required for the linear chains to be more efficient. This requirement decreases very quickly for longer chains, with only third order complexity required at $k = 4$. Finally, as with most real problems, $n \geq 4$ and the linear chain method is more efficient for all orders of complexity in the mathematical operations.

It is important to note that not only is this analysis not rigorous, but the order of the mathematical solution is itself dependent on the method which is used. Thus, the analysis gives a first glance into the comparison of efficiency, but is hardly complete.

Historically, a wide range of matrix methods, non-matrix time-step based solvers and matrix/time-step hybrid solvers have been used.

For the non-matrix methods, the time history of the problem is divided into time steps and the equations are solved using some variation of a standard differencing technique, including Runge-Kutta, Euler, or GEAR methods. By choosing small enough time steps, the numerical error of such methods can be reduced to make them sufficiently accurate. The consequence of using small enough time steps, however, is increased computational time.

Hybrid methods are variations on "scaling and squaring" where some small time step is chosen, either based on the time constants of the problem, or by taking sufficient square roots of the total time of interest. The scaled matrix exponential is solved with a rapidly converging series solution and then raised to some power, or successively squared, to reach the full operation time. These methods are both computationally inefficient, and in the general case, can be prone to round-off error.

It is also possible to use matrix decomposition methods to solve this formulation. The matrix exponential can be decomposed as, $\mathbf{A} = \mathbf{SDS}^{-1}$, giving the solution $e^{\mathbf{A}t} = \mathbf{S}e^{\mathbf{D}t}\mathbf{S}^{-1}$,. The goal of this class of methods is to find a decomposition such that the exponential $e^{\mathbf{D}t}$ is easily calculated. Some matrix decomposition methods which have been explored for this application include eigenvector decomposition,[27] generalized eigenvector decomposition[28] or Schur decomposition.[8] For some special forms of the transfer matrix, $\mathbf{A}$, these decomposition methods are accurate. However, when formulated for the general case, the decompositions can be prone to round-off error.

Within the class of matrix methods, however, implementations designed to take advantage of the special nature of the linear chain matrix $\mathbf{C}$, can be promising. The individual ODE's are solved with the assistance of Laplace transforms, filling the resultant matrix exponential one element at a time. These methods are versions of matrix decomposition methods which have been reformulated to minimize round-off errors and handle defective matrices, so much so that they are hardly recognizable as matrix decomposition methods. Previous applications have used a formulation of the analytic Bateman[12–14] solution for linear chains, but this is only applicable when no loops occur.

A new method, developed in this work, is able to replace the Bateman solution, in the event that a loop occurs in the linear chain.

One primary advantage of matrix methods is their ability to quickly solve problems with irradiation histories based on repetition, whether it be simple hierarchies of pulsing, or complex hierarchies of schedules and sub-schedules. A single matrix must be solved for each relevant flux level and/or characteristic time, and these matrices are multiplied appropriately to build the response matrix for the entire history. Time-step methods are unable to do this as they never generate a transfer matrix, but work directly with the isotopic number density vector. Since it is impossible to determine a unique transfer matrix given the initial and final vector, time-step methods are forced to simply step through the entire history, applying the whole array of calculations at each time step and modeling each pulse individually.

By transferring this system to the Laplace domain and considering each equation individually, it is possible to write the solution in a more compact form (N.B. P_i implies $P_{i-1\to i}$):

$$\tilde{N}_i = \frac{N_{i_0}}{s+d_i} + P_i \frac{\tilde{N}_{i-1}}{s+d_i} \tag{3.5}$$

$$\begin{aligned} &= \frac{N_{i_0}}{s+d_i} + \frac{N_{i-1_0}}{s+d_{i-1}}\frac{P_i}{s+d_i} + \frac{N_{i-2_0}}{s+d_{i-2}}\frac{P_{i-1}P_i}{(s+d_{i-1})(s+d_i)} + \cdots \\ &\quad + \frac{N_{2_o}}{s+d_2}\prod_{j=3}^{i}\frac{P_j}{s+d_j} + \frac{N_{1_o}}{s+d_1}\prod_{j=2}^{i}\frac{P_j}{s+d_j}, \end{aligned} \tag{3.6}$$

which can be written as

$$\begin{aligned} \tilde{N}_i &= \sum_{j=1}^{i} \tilde{N}_{ij} \\ &= \sum_{j=1}^{i} N_{j_o} \prod_{k=j+1}^{i} P_k \prod_{l=j}^{i} \frac{1}{s+d_l} \\ &= \sum_{j=1}^{i} N_{j_o} \tilde{F}_{ij}(s) \prod_{k=j+1}^{i} P_k. \end{aligned} \tag{3.7}$$

In this representation, the matrix $\mathbf{T}$ is filled by setting

$$\begin{aligned} T_{ij} &= N_{ij}/N_{j_o} \\ &= \mathcal{L}^{-1}\left[\tilde{F}_{ij}(s)\right] \prod_{k=j+1}^{i} P_k, \end{aligned} \tag{3.8}$$

and it becomes an exercise of solving for the inverse transform of the term

$$\tilde{F}_{ij}(s) = \prod_{l=j}^{i} \frac{1}{s+d_l}. \tag{3.9}$$

Normally, two such matrices, $\mathbf{T}$ and $\mathbf{D}$, are required to represent the pulse and dwell times, respectively. In the first case, all the destruction and production rates include the terms for neutron transmutation which are dependent on the flux spectrum and therefore it is only during this period in which loops can occur in the isotope tree. In the dwell period, the destruction and production rates are only those of decay, and therefore, many of the values will be zero.

3.1 The Analytical Bateman Solution

If all the destruction rates, d_i, are distinct, Equation 3.9 can be easily inverted,

$$f_{ij}(t) = \sum_{l=j}^{i} e^{-d_l t} \prod_{\substack{m=j \\ m\neq l}}^{i} \frac{1}{d_m - d_l}. \tag{3.10}$$

This leads to a compact representation of the solution to the Bateman equations:

$$N_i(t) = N_{i_o} e^{-d_i t} + \sum_{j=1}^{i-1} N_{j_o} \left[\sum_{k=j}^{i-1} \frac{P_{k+1}(e^{-d_k t} - e^{-d_i t})}{d_i - d_k} \prod_{\substack{l=j \\ l\neq k}}^{i-1} \frac{P_{l+1}}{d_l - d_k} \right]. \tag{3.11}$$

Finally, we can write the transfer matrix elements as:

$$\begin{aligned} T_{ii} &= e^{-d_i t} \\ T_{\substack{ij \\ i\neq j}} &= \sum_{k=j}^{i-1} \frac{P_{k+1}(e^{-d_k t} - e^{-d_i t})}{d_i - d_k} \prod_{\substack{l=j \\ l\neq k}}^{i-1} \frac{P_{l+1}}{d_l - d_k}. \end{aligned} \tag{3.12}$$

It can be shown that the mathematical operations performed in this method are essentially the same operations that would be carried out if an eigenvector decomposition was used to solve the matrix.

3.2 Laplace Inversion Method

When the destruction rates (eigenvalues) are not distinct, other methods of solving equation 3.9 are required. In general, however, because of the bidiagonal nature of the matrix representation (that is, because of the linear chain physical representation), this is a simple problem which, for a small system, can easily be solved on paper by hand. In particular, this Laplace space representation can be directly inverted to the time representation.

For repeated poles, one uses the residue theorem to determine the coefficients for each term in a partial fractions expansion. Each of those terms would result in an exponential, perhaps multiplied by a polynomial in time, t, when converted back to the time domain. Those residues are calculated using one of two simple rules.

If the pole is not repeated, then the residue, R_k, for the pole, $-d_k$, is calculated as

$$R_k = \lim_{s \to -d_k} (s + d_k)\tilde{F}_{ij}(s) \tag{3.13}$$

which becomes $R_k e^{-d_k t}$ in the time domain. If all poles have a singular multiplicity, the solution reduces exactly to the Bateman solution, and can be represented in many ways. This is obviously the solution with no loops.

If the pole is repeated m times, under a partial fraction expansion, this becomes m terms in that expansion:

$$\frac{R_{km}}{(s + d_k)^m} + \frac{R_{km-1}}{(s + d_k)^{m-1}} + \cdots + \frac{R_{k1}}{(s + d_k)} \tag{3.14}$$

which becomes

$$e^{-d_k t}\left(R_{km}\frac{t^{m-1}}{(m-1)!}+R_{k,m-1}\frac{t^{m-2}}{(m-2)!}+\cdots+R_{k2}\frac{t}{1!}+R_{k1}\right) \tag{3.15}$$

in the time domain. In this case, the residues are found using:

$$R_{kn}=\frac{1}{(m-n)!}\lim_{s\to -d_k}\frac{d^{m-n}}{ds^{m-n}}\left[(s+d_k)^m\tilde{F}_{ij}(s)\right]. \tag{3.16}$$

This latter rule requires the ability to evaluate derivatives of a generic function:

$$\tilde{G}^k_{ij}(s)=(s+d_k)^m\tilde{F}_{i,j}(s) \tag{3.17}$$

at values of $s=-d_k$ for all i. By examining the successive derivatives of $\tilde{G}(s)$ it can be shown (see Appendix A) that these derivatives can be recursively defined as:

$$\left[\tilde{G}^k_{ij}(s)\right]^{(n)}=\sum_{j=1}^{n}(-1)^j\frac{(n-1)!}{(n-j)!}\left[\tilde{G}^k_{ij}(s)\right]^{(n-j)}\sum_{i=1}^{I}\frac{1}{(s+d_i)^j} \tag{3.18}$$

and this, in turn, can be converted to a computational numerical algorithm, allowing the entire problem to be solved.

We will call this method the Laplace Inversion Method.

3.3 Laplace Expansion Method

Both of the above analytical solutions can be prone to round-off error when the absolute difference between two poles is small, either because the poles are nearly identical or because both poles are very small. Each term in the sum will be very large due to division by small numbers, and successive terms will differ only in the least significant digits. For these cases, an expansion in $1/s$ may be preferred.

It is apparent from equation 3.12, however, that such divisions can be eliminated by writing the solution as a difference of exponentials, using the series expansions for the exponentials, factoring out the offending term from the numerator and canceling. While this seems like a monumental task to perform on an arbitrary problem, thanks to the bidiagonal nature of the system, it is again quite simple to implement. If we start again with our function:

$$\tilde{F}_{ij}(s) = \prod_{l=j}^{i} \frac{1}{s+d_l} \tag{3.19}$$

and making no assumptions about the multiplicity of the poles, we expand this as a series in $1/s$, the result is:

$$\begin{aligned}
\tilde{F}_{ij}(s) &= \frac{1}{s^{i-j+1}} \prod_{l=j}^{i} \frac{1}{1+\frac{d_l}{s}} \\
&= \frac{1}{s^{i-j+1}} \prod_{l=j}^{i} \left(1 - \frac{d_l}{s} + \frac{d_l^2}{s^2} - \frac{d_l^3}{s^3} + \cdots\right) \\
&= \frac{1}{s^{i-j+1}} \left[1 - \frac{\sum_{l=j}^{i} d_l}{s} + \frac{\sum_{l=j}^{i} d_l \sum_{k=l}^{i} d_k}{s^2} - \frac{\sum_{l=j}^{i} d_l \sum_{k=l}^{i} d_k \sum_{m=k}^{i} d_m}{s^3} + \cdots\right].
\end{aligned} \tag{3.20}$$

If $n = i - j$, in the time domain, this becomes:

$$\begin{aligned}
f_{ij}(t) = t^n \Bigg[&\frac{1}{n!} - \frac{t}{(n+1)!} \sum_{l=j}^{i} d_l + \frac{t^2}{(n+2)!} \sum_{l=j}^{i} d_l \sum_{k=l}^{i} d_k \\
&- \frac{t^3}{(n+3)!} \sum_{l=j}^{i} d_l \sum_{k=l}^{i} d_k \sum_{m=k}^{i} d_m + \cdots \Bigg]
\end{aligned} \tag{3.21}$$

and thus:

$$\begin{aligned}
T_{ii} &= e^{-d_i t} \\
T_{\substack{ij \\ i \neq j}} &= f_{ij}(t) \prod_{k=j}^{i-1} P_k \,.
\end{aligned} \tag{3.22}$$

It is clear that this solution will only be computationally viable when the product, $\max\{d_i\} \cdot t$ is small. For arbitrary problems, this is only guaranteed when there are short operating times, but can easily be tested for a particular problem. Other representations can be formed, each providing different insight into the method (see Appendix B).

We will call this method the Laplace Expansion Method.

3.4 Adaptive Mathematical Methods

Upon examining available methods to solve such lower bidiagonal systems, there are certain easily determined matrix characteristics which can be used to adaptively choose a method for each matrix element which will optimize the speed and accuracy of the solution.

The primary criterion for choosing amongst the solution methods is the presence of degenerate eigenvalues. The eigenvalues of these matrices are simply the diagonal elements, which in turn are the destruction rates of each isotope in the linear chain being modeled. True degeneracies in these values will occur only when the chain being modeled by this matrix has straightened loops and a simple test can determine whether this is the case. Each matrix element, $T_{i,j}$ represents the transfer within a segment of the chain between nodes j and i. Even if loops exist elsewhere in the chain, if no loops occur in this chain segment, the analytical Bateman solution can be selected.

If a loop does exist within this sub-chain, the laplace expansion technique is immediately implemented. When, during the course of the expansion solution, it does not converge quickly enough, the laplace inversion technique is invoked.

For the dwell periods between pulses, the Bateman solution can always be used because the absence of transmutation reactions makes loops physically impossible.

Because the mathematical method is chosen independently and adaptively for each matrix element, the efficiency and accuracy of the entire solution is optimized. If a small loop occurs in one small portion of an activation tree, it is not necessary to perform the relatively computationally intensive Laplace Expansion and/or Inversion solution on the entire problem. On the other hand, the solution is not limited by the non-loop Bateman solution when loops do exist.

3.5 ALARA Implementation Summary

As with the physical modeling, the implementation of the mathematical solution requires some special implementation to enhance its efficiency.

The first such enhancement is to store the solution matrices from one chain to another. The value of this implementation can be seen when considering the solution of many subsequent chains of large rank, n. Without saving the previously generated matrices, all the $n(n-1)/2$ elements of the lower-triangular transfer matrices would have to be completely recalculated ($n^2/2$ calculations) for each of these subsequent chains, even if they only differ in the last rank. When the transfer matrix solution for each interval is stored after its use in the solution of one chain, only the last row (n calculations) is necessary. This savings carries to all situations throughout the activation tree. Since the chains are formed by a depth-first search, it is likely that for each chain, a significant portion of the data has already been determined for a previous calculation. Since the physical model allows a complex hierarchy of irradiation schedules and sub-schedules, the storage for these transfer matrices must mimic this hierarchy. Furthermore, as the reaction rates are a function of the neutron flux, one of these storage hierarchies is required for each interval.

This is also true for the decay matrices between pulses and after shutdown, however in this case, the matrices are not only saved from one chain to the next, but across all the intervals being solved for the same chain. Since the decay matrices are independent of flux, they need only be calculated once for each chain and then used in all the intervals for that chain. This has potentially large savings since there may be many intervals sharing the decay matrices, each recalculation of which costing n calculations even with the already implemented savings. Furthermore, since the decay matrices are likely to be much sparser than the pulse transfer matrices, a special implementation of the Bateman solution routine to intelligently fill these matrices has been implemented. If any single production rate is equal to zero in the chain segment relevant to a particular matrix element, that matrix element will be zero, and there is no point in performing the full Bateman solution.

These mathematical methods are implemented during a depth-first search of the irradiation history hierarchy. A storage block is allocated for each item in each schedule in the hierarchy, consisting of the item's operation block transfer matrix and the total transfer matrix, including the effect of the operation block, the pulsing hierarchy and the dwell time between this block and the next. For a simple pulse the operation block matrix is the transfer matrix for the pulse itself while the total transfer matrix includes the effect of the pulsing hierarchy and the inter-item dwell period. For a complex sub-schedule, the operation block is calculated each time the depth-first search retracts back up the schedule hierarchy as the product of total transfer matrices of that sub-schedule's items. The total transfer matrix again combines the effects of the operation block, the pulsing hierarchy and the inter-item dwell period.

The solution of a particular pulsing hierarchy deserves a little more attention. Given the transfer matrix of the operation block, say $\mathbf{T}_0$, and a single dwell matrix for the dwell time of the first level of pulsing, $\mathbf{D}_1$. The product of these, $\mathbf{D}_1\mathbf{T}_0$, is then raised to a power representing one less than the number of pulses in that level, n_1. Multiplying by the operation block matrix one more time becomes the transfer matrix for the next level of pulsing, $\mathbf{T}_1 = \mathbf{T}_0(\mathbf{D}_1\mathbf{T}_0)^{n_1}$. This is repeated with the single dwell matrix for the dwell time of the second level, $\mathbf{D}_2$, and so on, using the general formula[23]

$$\mathbf{T}_i = \mathbf{T}_{i-1}(\mathbf{D}_i\mathbf{T}_{i-1})^{n_i}. \tag{3.23}$$

It is important to use an efficient algorithm[29] for this matrix exponentiation process since it will be performed so often during the operation of the code. For N levels of pulsing, the matrix, $\mathbf{T}_N$, is the transfer matrix for the entire pulsing hierarchy up to the dwell time after that item.

Using figure 2.7 as an example, $\mathbf{T}_0$ represents the activation solution for one 10 minute pulse and $\mathbf{D}_1$ represents the solution for one 20 minute dwell period. Therefore, $\mathbf{T}_1 = \mathbf{T}_0(\mathbf{D}_1\mathbf{T}_0)^{15}$ represents the solution for one work-day's worth of pulsing. If $\mathbf{D}_2$ represents the solution for one dwell period of 16 hours and 20 minutes, then $\mathbf{T}_2 = \mathbf{T}_1(\mathbf{D}_2\mathbf{T}_1)^4$ is the solution for a full 5 day week of experiments. This continues until the last level where $\mathbf{T}_4 = \mathbf{T}_3(\mathbf{D}_4\mathbf{T}_3)^{19}$ represents the solution for 20 years of operation.

Chapter 4

Validation and Benchmarking

For all new computer programs, an important step in the development process is the program's validation against other computational tools. In the field of fusion activation calculations, there are many such tools. Both FISPACT-97[6] (time-step based ODE solver) and DKR[12] (no-loop Bateman solution to linear chains) have been shown in the past to agree well with analytical solutions to multi-step activation pathways.[15,16] ALARA offers improvement over DKR because it is able to accurately model loops in the activation trees and calculate the gas production. In comparison to FISPACT-97, ALARA has many advantages, including the ability to exactly model pulsed irradiation histories and simultaneously calculate the solution at many spatial points. In comparison to both codes, ALARA uses modern programming practices and data handling to increase the flexibility of operation and reduce memory requirements (see section 5.1).

4.1 Benchmark Specifications

To validate ALARA the International Atomic Energy Agency [IAEA] Fusion Evaluated Nuclear Data Library [FENDL] Calculational Activation Benchmark[30] problem was chosen. This problem is based on the reference steel/water shielding blanket design in the International Thermonuclear Experimental Reactor [ITER] outline design. This design includes:

- a copper first wall with beryllium coating
- shielding blanket with alternating layers of stainless steel (316 SS) and water
- a double wall Inconel 625 vacuum vessel with a water-cooled steel pebble bed and a back shield made of lead and boron carbide
- an inboard magnet, including conductors and insulators.

With such a wide range of materials, the design offers an extensive test of the activation code's capabilities.

The neutron fluxes have been provided by the benchmark in the VITAMIN-J 175 group energy structure for each of the 468 fine mesh intervals. These fluxes were calculated using the ONEDANT[31] deterministic neutron transport code with a 14.1 MeV isotropic neutron source normalized to inboard and outboard neutron wall loadings of 1 and 1.5 MW/m^2, respectively.

IAEA Fusion Evaluated Nuclear Data Library (ver. 2.0) for Activation (FENDL/A 2.0) and for Decay (FENDL/D 2.0) were used for all calculations, ensuring that difference in the results were due to the activation codes themselves and not the nuclear data.

The first activation calculations were performed with a steady state operation time of 3 years and cooling times of 1 hour, 1 day, 1 week, 30 days, 1 year and 100 years. A pulsed activation calculation was also performed using 94500 pulses of 1000 s with a dwell time of 1200 s between pulses.

4.2 Steady-State Problem

In ALARA this calculation built 109 reaction trees with a total of 32023 nodes and a longest chain of length 14, producing 603 different isotopes in the steel-containing intervals, of which 402 were radioactive. The truncation tolerance was 10^{-4} and the ignore tolerance, 10^{-6}.

The results have been compared by calculating the relative difference between ALARA and the other codes:

$$\text{Relative Difference} = \frac{\text{ALARA}}{\text{X}} - 1,$$

where X is either FISPACT-97 or DKR.

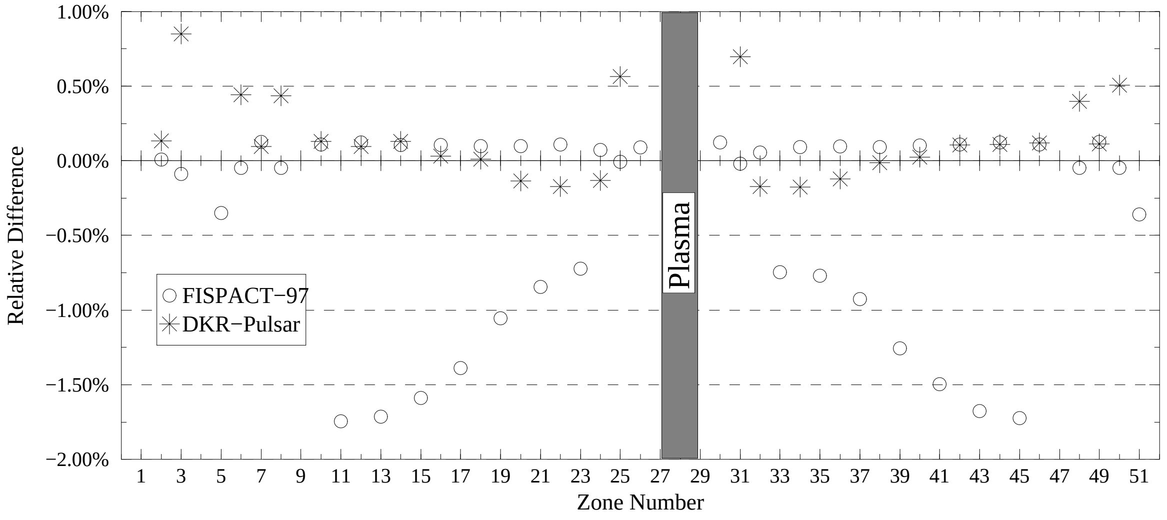


Figure 4.1: Relative difference between ALARA and other codes for steady state problem at a cooling time of 1 hour.

Figures 4.1 and 4.2 show the relative difference between the steady state problem's results from ALARA and FISPACT-97 and between ALARA and DKR at a cooling time of 1 hour and 1 century, respectively.

At a cooling time of 1 hour, the ALARA results are within 1.8% of the FISPACT-97 results throughout the entire geometry, with most zones having a difference of less than

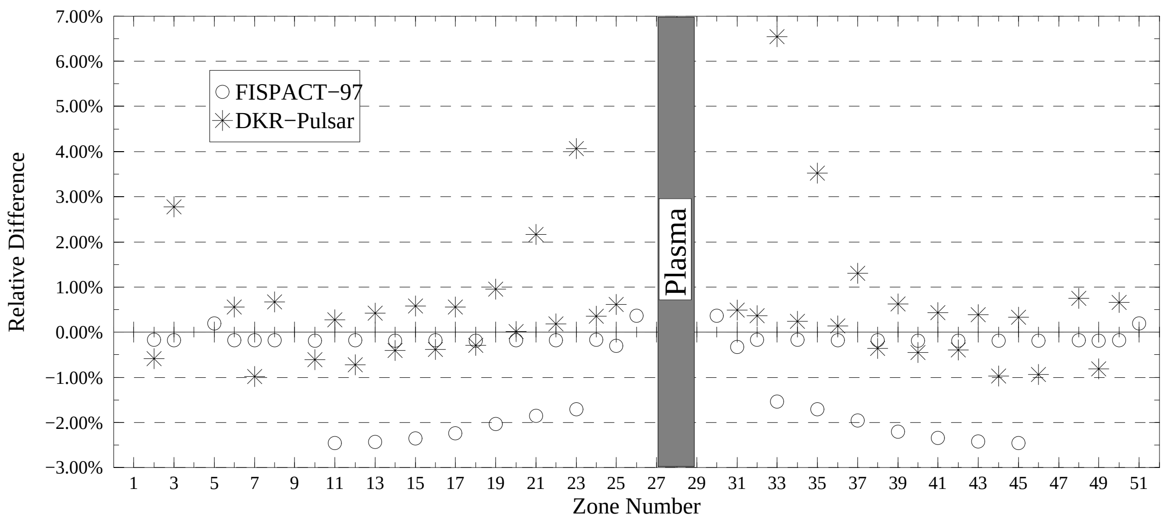


Figure 4.2: Relative difference between ALARA and other codes for steady state problem at a cooling time of 1 century.

0.4%. The largest differences occur in the blanket's 14 water-filled zones, increasing in directions away from the plasma as flux becomes lower and softer.

At a cooling time of 1 century, the differences between ALARA and FISPACT-97 are as high as 2.5%, with the largest differences still occurring in the water-filled zones where, after more than 8 tritium half-lives, the dominant isotope is now ^{14}C. The differences in the other zones remain below 0.4%.

The results from these comparisons immediately demonstrate the impact of the various activation codes different methods. In most zones, the ALARA results are within a fraction of a percent of the FISPACT-97 values. In the water-filled zones, alternating with the steel in the inboard and outboard blanket, the total activity was greatly underestimated. Further investigation shows that the activity is dominated by tritium and ^{14}C, and that their relative production levels are as low as 5×10^{-12} and 3×10^{-9}, respectively. It is clear that this level of accuracy is not sufficient for these zones. On the other hand, increasing the accuracy for the whole problem would dramatically increase the activation tree size in all the other zones. While this demonstrates the limitations of a problem-wide truncation tolerance, it requires little effort to perform a second ALARA calculation. A second calculation was performed for this problem using 10^{-10} for both the truncation and ignore tolerances. Figures 4.1 and 4.2 are based on these values in the odd numbered zones between 11 and 23 and between 33 and 45, inclusively.

DKR had an even more drastic problem with zones where the specific activity was dominated by tritium. Since the methods implemented in this version of DKR were not designed to compute the accumulation of light ions (^{1}H, ^{2}H, ^{3}H, ^{3}He, and ^{4}He) emitted by nuclear reactions, the major source of tritium, its total activity for all zones with dominant tritium inventories is much too small. Since these differences were up to a few orders of magnitude, the relative difference between ALARA and DKR in these zones has not been shown in figure 4.1 in order to compare the differences in the other zones. The magnitude of these differences can be inferred, however, from figure 4.2, where it must be noted that 1 century is over 8 tritium half-lives. The undercalculation of tritium in these zones is so great that even after more than 99.6% of the tritium has decayed, the error is still a few percent.

In those zones with insignificant tritium inventories, the differences between ALARA and DKR are less than 0.2% throughout the geometry at a cooling time of 1 hour. After 1 century, even though the relative contribution of tritium has increased in some of those same zones, the relative differences are still less than 1%. DKR's inability to account for the light ion production results in tritium inventories as much as 6 orders of magnitude too low in the first wall's Be coating, and up to 3 times too low in the blanket's water cooled zones. Even after more than 8 tritium half-lives (1 century), when tritium is responsible for less than 10% of the total activity, the difference between ALARA and DKR in the water can be as high as 7% (zone #33). This discrepancy demonstrates why the modeling of light ion accumulation is a base features for any new activation code.

A single fine mesh interval was chosen in which to compare the activity of various nuclides in detail. The 1 mm thick stainless steel (SS316) inboard first wall back plate is modeled as a single interval (#242) in zone #24. The large number of initial isotopes in steel and the high flux due to its proximity to the plasma make this a good choice for comparison.

Table 4.1 shows the seven most dominant isotopes at a cooling time of 1 hour, together accounting for over 95% of the total activity. Table 4.2 shows the five most dominant isotopes at a cooling time of 1 century, accounting for more than 99.7% of the total activity.

Table 4.1: Detailed differences in interval #242 at 1 hour.

Isotope	ALARA	Relative Difference [%]	
	[10^{16}Bq/m^3]	FISPACT-97	DKR-Pulsar 2.0
^{56}Mn	114.6	-0.051	0.15
^{55}Fe	85.5	0.24	-0.10
^{51}Cr	75.7	0.019	-0.041
^{57}Co	24.4	-0.081	3.0
^{54}Mn	20.0	0.97	0.15
^{58m}Co	10.3	-0.053	0.30
^{58}Co	8.32	-0.048	-13.74

The agreement between ALARA and FISPACT-97 is seen to be within 1% in all cases. DKR, on the other hand, has relative differences of up to 16%. These discrepancies are most probably caused by the inability of DKR to model certain kinds of loops in the decay chains and the influence this has on the decay chain creation calculations.

Table 4.2: Detailed differences in interval #242 at 1 century.

Isotope	ALARA	Relative Difference [%]	
	[10^{13}Bq/m^3]	FISPACT-97	DKR-Pulsar 2.0
^{63}Ni	27.8	-0.17	0.40
^{59}Ni	3.80	-0.18	-1.2
^{91}Nb	3.37	-0.21	1.3
^{14}C	0.86	-0.22	-0.19
^{93}Mo	0.69	-0.21	16

4.3 Pulsing Problem

The results of ALARA and DKR for the pulsing problem are compared in Figure 4.3 for both 1 hour and 1 century. There are no results for FISPACT-97 as it was unable to solve the exact pulsing problem in a reasonable amount of time. Based on the time (14 minutes) required to solve the first 100 pulses at a single spatial point, FISPACT-97 was determined to be unsuited to such pulsed operation calculations. Assuming that the total computation time scales linearly with the number of pulses, the full problem would require more than 220 days for each of the 317 spatial points – a total computation time of over 190 years!

Once again, tritium plays an important role in the discrepancies which are nearly identical to the discrepancies between ALARA and DKR for the steady state problem. In TF coil's glass insulator (zone #3), the discrepancy in the pulsing problem is twice as high as in the steady state problem at 1 hour, but the same at 1 century. This demonstrates the true physical effect of pulsing on the tritium inventory's importance at relatively short cooling times, even though the tritium inventory itself is not significantly affected by the pulsed history.

The pulsed operation will tend to reduce the inventory of isotopes with half-lives of the same order of magnitude as the dwell time between pulses: very long-lived isotopes will decay little between pulses and slowly reach their saturation level while very short-lived isotopes will decay completely between pulses, but can reach their saturation level in a single pulse.[21] The dominant isotopes in the glass at 1 hour are ^{64}Cu and ^{24}Na, both with half-lives slightly longer than half a day. Their inventories in the pulsed problem are 50% less than in the steady state problem, while the tritium inventory is reduced by less than 10%. Thus, the fraction of the total activity from tritium is increased, even though the tritium inventory itself goes down.

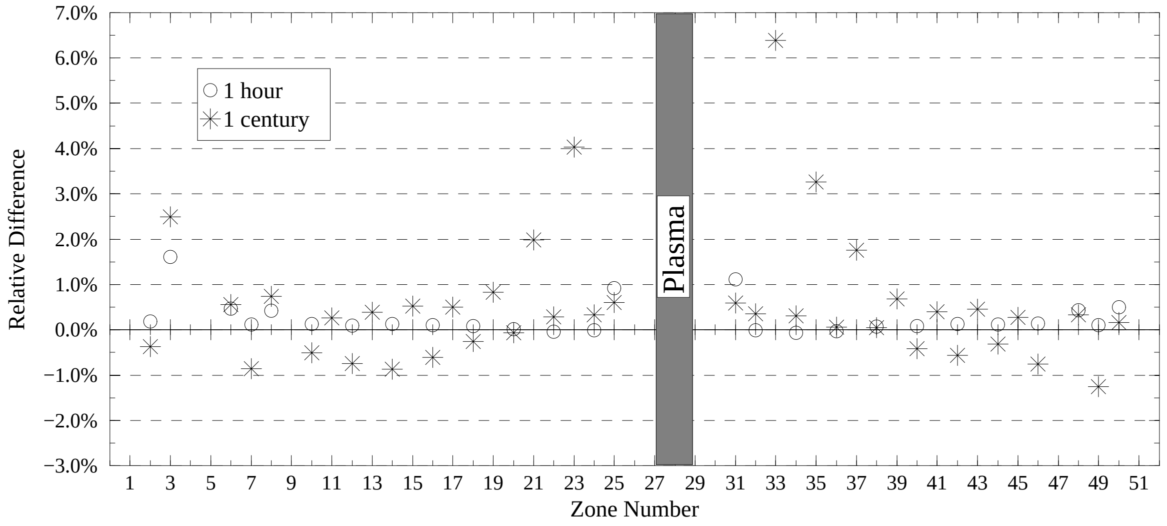


Figure 4.3: Relative difference between ALARA and DKR for the pulsing problem at cooling times of 1 hour and 1 century.

One method of modeling a pulsed problem as a steady state problem is to preserve both the total fluence and the total operating time.[21,32,33] Using ALARA, the results of such an approximate calculation with a flux scaling factor of 1/2.2 and total operation time of 2.079×10^8 are compared to the exact solution in Figure 4.4, represented as

$$\text{Relative Difference} = \frac{\text{Pulsing}}{\text{Steady State}} - 1.$$

In this case, due to the nature of the pulsing history, the effect can only be seen at short cooling times. As discussed above, the pulses' effect is greatest for isotopes with half-lives of roughly the same order of magnitude as the dwell time between pulses. Thus the effect decays away within a few "dwell times" of shutdown.

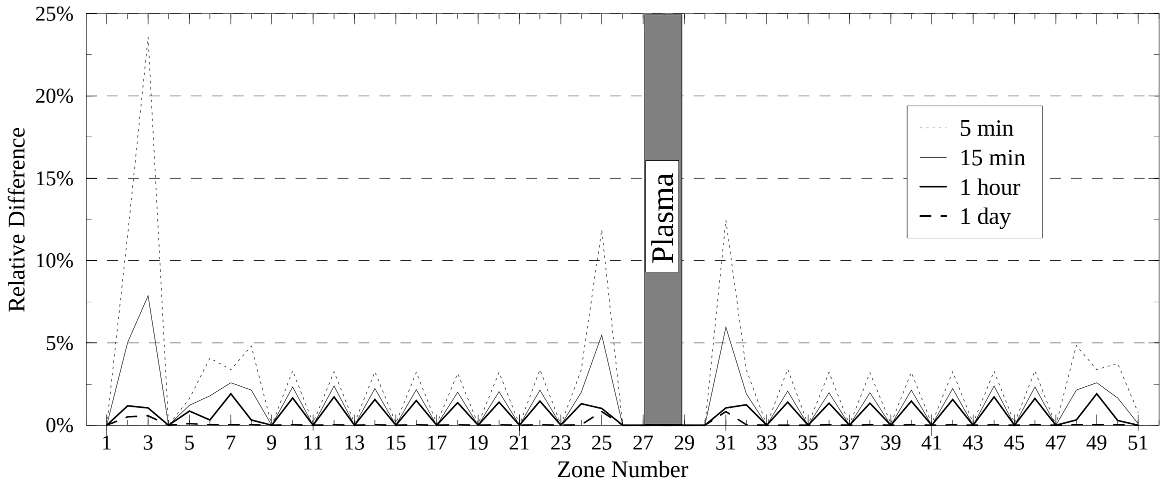


Figure 4.4: Relative difference between exact pulsed solution and steady state approximation at various cooling times.

The two materials with largest discrepancies are the glass insulator (zone #3) and the first wall heat sink (Cu-Be-Ni in zones #25 and #31). In the former, the activity of ^{28}Al ($t_{1/2}$ = 2.25 m), responsible for over 25% of the activity at a cooling time of 1 minute, is under-calculated in the steady-state approximation by 50%. The same is true of ^{66}Cu ($t_{1/2}$ = 5.10 m), responsible for just under 10% of the activity in the first wall.

4.4 Advanced Features

It is not possible to validate ALARA's complex schedule modeling or reverse calculation mode through comparison to other codes because these features are unique to ALARA.

It is possible to reformulate these benchmark problems into test problems for ALARA's advanced features, comparing the results to those for the standard benchmark problem.

4.4.1 Complex Schedule Modeling

With the standard single pulsing history capabilities validated against DKR in section 4.3, it is possible to construct a number of complex pulsing schedules, each reducible to the same pulsing schedule as the IAEA Benchmark problem. Four such reformulations, with increasing complexities, have been tested and compared to the normal pulsing problem's results.

The first formulation splits the 94500 pulses into two groups of 47250 pulses, and a delay of 1200 s after the first group of pulses. The next formulation uses four groups of pulses of 23625 pulses with 1200 s delay after each of the first 3 groups. The third formulation increases the number of levels in the schedule hierarchy. The top level schedule has two sub-schedules. Each sub-schedule is repeated 100 times, with 1200 s delay after the first sub-schedule. The first sub-schedule has two pulsing groups, each with 250 pulses and separated by a 1200 s delay. The second sub-schedule has two groups of pulses with 225 and 220 pulses respectively. The total number of pulses is again $[100 \times (250 + 250) + 100 \times (225 + 220)] = 94500$. The last formulation is the most complicated. The top level schedule again has two sub-schedules. The first sub-schedule is repeated 10 times and consists of a block of 2500 pulses followed by a 1200 s delay and then another sub-schedule, consisting, in turn, of two blocks of 125 pulses, and is repeated 10 times. The second sub-schedule is repeated 100 times and has one block of 445 pulses. Summing the entire schedule gives $[10 \times (2500 + 10 \times (125 + 125)) + 100 \times 445] = 94500$ pulses.

After the complete solution of all four problems, the only difference amongst the results was the length of time taken to calculate them (using the unix program 'diff' on both the output files and tree files showed no difference). This is logical since increasing the schedule complexity increases both the number of matrices created and the number of matrix calculations required to combine their effects into the whole schedule.

4.4.2 Reverse Calculation

In order to test the reverse calculation mode, it is sufficient to formulate a problem targeting isotopes known to be created by the forward calculation. Since steel is the most prevalent material in the IAEA Benchmark, the 4 most significant (in terms of specific activity) isotopes were chosen as targets in the steel mixtures: ^{51}Cr, ^{54}Mn, ^{55}Fe, and ^{57}Co. With the goal being to determine the production of these isotopes from steel, all zones without stainless steel 316 were changed to "void" to reduce the computational time.

The shutdown inventories for these four isotopes were identical to those of the forward calculation and table 4.3 shows the source of each target isotope.

4.5 Computing Resources

For the steady-state and pulsing code comparison, all three codes were used on the same IBM RS/6000 Model 595 P2SC workstation. The full steady-state problem was solved by ALARA in 3425 s (57^m5^s) and by DKR in 5253 s ($1^h27^m33^s$). The same problem required 20715 s ($5^h36^m25^s$) for a previously developed shell-script system sequentially running FISPACT-97 for each of the intervals. The pulsed problem was solved by ALARA in 5736 s ($1^h35^m36^s$), with 44591 nodes and a longest chain of 14. DKR needed 10855 s ($3^h0^m55^s$). FISPACT-97 was unable to solve the pulsed problem.

Source isotope	**Target isotopes**			
	^{51}Cr	^{54}Mn	^{55}Fe	^{57}Co
^{51}V	1.9 appm	–	–	–
^{50}Cr	69%	–	–	–
^{52}Cr	29%	–	–	–
^{53}Cr	221 appm	–	–	–
^{54}Cr	–	101 appm	–	–
^{55}Mn	–	31%	830 appm	–
^{54}Fe	2.6%	68%	24%	–
^{56}Fe	–	0.65%	72%	–
^{58}Ni	–	238 appm	3.9%	100%

Table 4.3: Sources of primary radioactive isotopes in stainless steel first wall layer.

For the steady state problem, ALARA requires a maximum of 35 MB of RAM and, other than the binary library of just over 11 MB, uses no hard drive space. DKR required as much as 107 MB of RAM and up to 250 MB of temporary hard drive space in addition to its 10 MB text library. Other than the 38 MB data libraries, FISPACT-97 uses negligible quantities of RAM and hard drive space since it solves each interval sequentially.

For the advanced feature validation, a complete set of problems was run on the development platform, an Intel P166MMX based machine running the Linux 2.0.30 operating systme. Table 4.4 shows the time required for each problem.

steady state	17.5 m
basic pulsing	24.7 m
complex pulsing 1	40.3 m
complex pulsing 2	68.8 m
complex pulsing 3	70.2 m
complex pulsing 4	74.1 m
reverse problem	1.7 m

Table 4.4: Runtimes for benchmark suite.

4.6 Conclusions

The ALARA activation code has been validated for use in calculating the activation of fusion power systems. The results for a steady state activation problem have been compared to the results from two standard codes whose accuracy has been well documented: FISPACT-97 and DKR. Discrepancies between the total activities calculated by ALARA and the other codes are always less than 2.5%, except where DKR is unable to calculate the tritium production from emitted light ions. The results of a pulsing problem have been compared to DKR (FISPACT-97 was unable to perform such a calculation in a reasonable time). The discrepancies in this case are once again primarily due to the lack of tritium production in DKR, and are otherwise less than 1%.

Based on this validation and its faster and less memory-intensive operation, ALARA is recommended for the solution of fusion activation problems.

Chapter 5

Summary and Future Development

5.1 Modern Implementation Features

One advantage to writing an absolutely new activation code is the ability to implement modern computing techniques, including algorithms and data structures, data handling, software design and user-oriented features. This section will describe some of those implementations in ALARA.

5.1.1 Software Design

ALARA's source code, written in *C++*, takes advantage of object-oriented programming techniques. This offers the primary benefit of improved extensibility and maintainability of the code. Strictly designed classes with appropriate inheritance, information hiding and well-documented interfaces minimize the unpredictable effects of isolated modifications and extensions. The code is also made more readable by using short functions with specific purposes and overloaded operators to perform standard operations on complex classes (such as matrix operations: A = B * C). See the class header files (distributed with the source code) for the documentation and definition of the data classes and their interfaces.

Many of these classes are implemented as linked lists, permitting the problem's computational representation to grow freely as required. During the input processing, this permits the user to add an arbitrary number of each input element, such as mixture descriptions, schedule definitions or flux declarations. During the next phase, the linked lists for each input element is then conveniently cross-referenced to accelerate future searches and linking in the problem. The linked list structures are also important for accumulating the solution. Each interval has a linked list of solution vectors where each item in the list represents an isotope generated in the tree.

5.1.2 Data Handling

Due to the size of the individual activation tree and the number of initial isotopes, ALARA must frequently access the nuclear data library. Past codes have developed various mechanisms to optimize this access to sequentially read data files, but many methods are based on the usage of magnetic tapes. For example, DKR performs a breadth-first search algorithm for chain creation, sorting the isotopes at each generation before reading their nuclear data. This requires it to simultaneously store all the chains for a whole activation tree. In order to minimize the required computational resources for physical modeling in ALARA, the data library access is not restricted to optimized sequential access. Instead, an indexed binary data file format has been designed to permit frequent random access. Built into ALARA is the ability to convert other popular non-binary library formats to the proprietary ALARA format. If a library formated to the European Activation File[18] conventions, for example, is offered as the data library, it is automatically converted to an ALARA binary library for the immediate problem and then saved under a default name for future problems.

Similarly, to minimize memory usage throughout the problem, ALARA makes use of a binary file to log the solution of each root isotope in each interval. In essence, this so-called dump file records the concentration of all the isotopes resulting from a single root isotope. This data is loaded again during post-processing after the calculation to be summed and processed into the type of output requested by the user. This dump file could also serve as the interface to a graphical post-processor, should one be developed. Since all the engineering responses based on an activation calculation are generated from the scaling and summing of these individual interval results, the input file and dump file can be used to reconstruct the solution to the problem.

5.1.3 User-Oriented Design

When choosing new software, one primary factor considered by users is the ease of use. Two aspects of this factor are the input file's simplicity and readability and the code's flexibility to a variety of applications.

The first of these has been addressed by creating a freely-formatted input file format based on natural language (rather than a seemingly random array of numbers and parameters). The various sections of the input can be given in any order and are cross-referenced using symbolic names defined by the user. Input file sections can be included from other files, permitting frequently re-used segments to be archived in a kind of library. Finally, the input file permits comments and blank lines to improve its readability. Once read, the input is tested extensively for completeness and self-consistency, with a long list of error and/or warning messages when it is found to be incomplete of inconsistent.

The flexibility of ALARA for different problems has already been demonstrated. One main sources of flexibility is ALARA's reliance on its data library to provide all the information needed for each nuclear reaction. ALARA has already been implemented to study a nuclear system with intermediate energy fluxes (up to 55 MeV),[25] where no other major activation code was able to solve the problem. Furthermore, this application demonstrated the importance of the spectrum's higher energy tail to the activation results compared to previous approximations using other activation codes. It is theoretically possible to analyze systems with fissile material if the fission product production terms were included as a type of "transmutation" cross-section. Finally, ALARA is not limited to neutron activation problems, but since it is library driven, can solve activation problems with any kind of incident particle (*e.g.* proton activation).

The reverse calculation mode also enhances the flexibility, allowing ALARA to be used in completely new ways and applications. For example, assuming that small changes in the material composition have little impact on the neutron transport solution, the reverse calculation can be used to tailor a material to minimize its activation levels in a particular system. Similarly, a reverse calculation could be used to determine the best composition to irradiate in order to produce a desired radioactive isotope such as those used in medical applications.

5.1.4 General Program Flow

The program flow of ALARA can be divided into three main phases, as shown in figure 5.1. The most important of these phases is the solution phase, in which all the physical modeling methods and mathematical techniques described in earlier chapters are implemented.

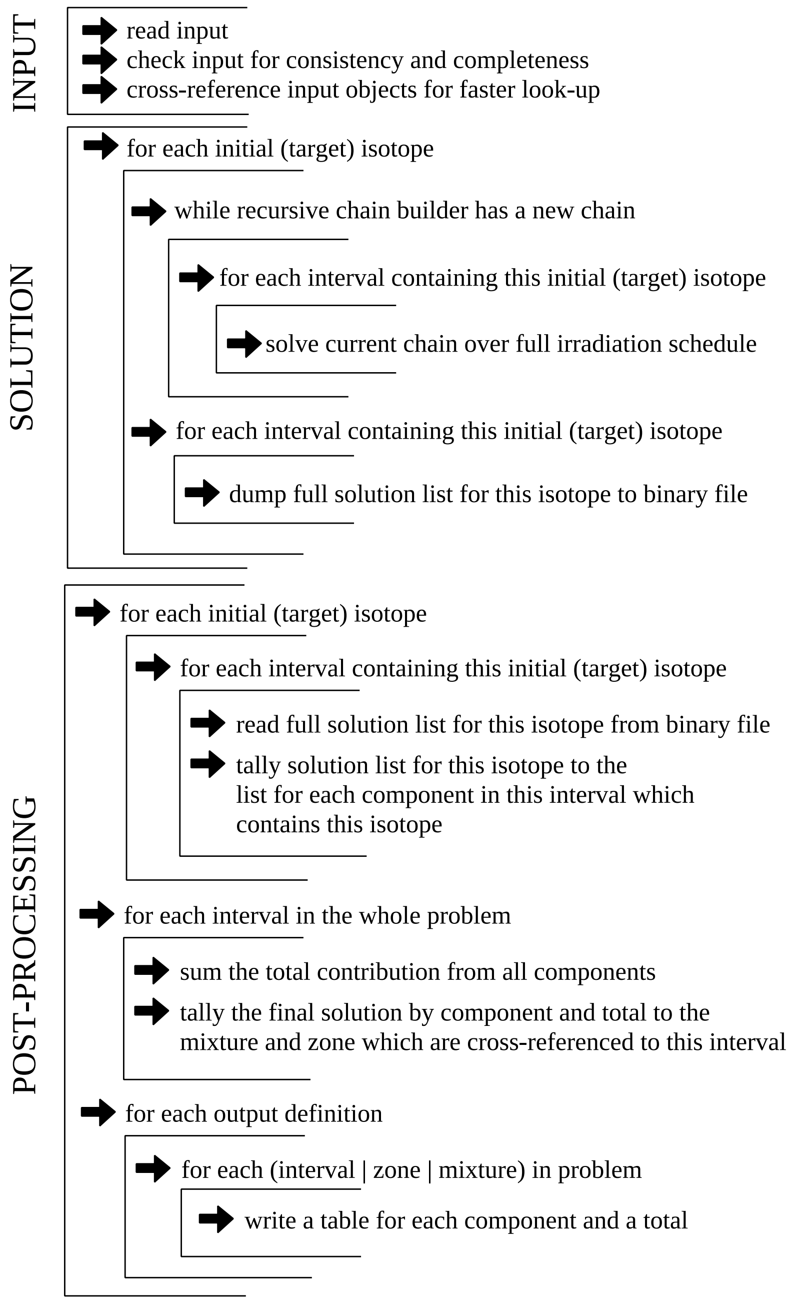

INPUT
read input
check input for consistency and completeness
cross-reference input objects for faster look-up
SOLUTION
for each initial (target) isotope
while recursive chain builder has a new chain
for each interval containing this initial (target) isotope
solve current chain over full irradiation schedule
for each interval containing this initial (target) isotope
dump full solution list for this isotope to binary file
POST-PROCESSING
for each initial (target) isotope
for each interval containing this initial (target) isotope
read full solution list for this isotope from binary file
tally solution list for this isotope to the list for each component in this interval which contains this isotope
for each interval in the whole problem
sum the total contribution from all components
tally the final solution by component and total to the mixture and zone which are cross-referenced to this interval
for each output definition
for each (interval | zone | mixture) in problem
write a table for each component and a total

Figure 5.1: The general flow of ALARA can be described as follows:

5.2 Future Developments

A variety of extensions and features have already been considered for implementation in ALARA. This section will describe just a few possible extensions for future versions in ALARA's continuing development.

5.2.1 Sequential Charged-Particle Reactions

The physical model as described in Chapter 2 only allows for the production of different isotopes from the original ones by the neutron flux experienced in the system. However, many of these neutron reactions result in emitted light ions, which are themselves able to induce transmutation reactions. Recent work by Cierjacks, *et al*,[34] has identified the importance of such reactions and developed a data library for the inclusion of these reactions within an activation code.

Some development of this method will be necessary for inclusion in ALARA because the original library was designed for use with the 0-D code FISPACT. As such, the equations used to determine the effective cross-sections for these reactions require full knowledge of the neutron flux and isotopic composition at a point in space. To use the equations directly in ALARA would require either greatly reduced speed or greatly increased internal storage. Instead, it is possible to modify the equations into a matrix format, creating a transformation from neutron flux to charged particle flux, dependent only on the initial isotopic composition. Then, for each interval having that mixture definition, the charged particle fluxes can be determined and stored for later use in calculating production/destruction rates. A group-wise maximum charged particle flux will also be used in truncation calculations.

Beginning with the equation for charged particle flux from Cierjacks, generalized to an N group neutron spectrum,

$$\begin{aligned}\Phi_{x_k} = \sum_A \sum_{i=1}^{N} \overline{\Phi_{n_i}} \sigma_{nx}^A(E_{n_i}) N_A \Delta E_{n_i} \\ \times \sum_{j=k}^{24} f_{nx}^A(E_{n_i}, E_{x_j}) \Delta E_{x_j} \Delta R_{x_k},\end{aligned} \qquad \text{(Eqn (4) in Ref. 34)}$$

where

Φ_{x_k}	$\equiv$	Charged particle flux of type x and energy group k,
$\overline{\Phi_{n_i}}$	$\equiv$	Neutron flux of energy group i,
$\sigma_{nx}^A(E_{n_i})$	$\equiv$	Group cross-section for production of charged particle x from bombardment of isotope A by neutrons of energy group i,
N_A	$\equiv$	Initial atom density of isotope A,
$f_{nx}^A(E_{n_i}, E_{x_j})$	$\equiv$	Energy distribution of charged particle x caused by reactions between isotope A and neutrons of energy i given in discrete bins of charged particle energy j,
ΔR_{x_k}	$\equiv$	Incremental range of charged particle x of energy group k in material.

To convert this to a transformation matrix between the neutron flux and the charged particle flux, it is first necessary to recognize that the energy increment, ΔE_{n_i} is usually included in the group-wise flux,

$$\begin{aligned}\sigma_{nx}(E_{n_i}) &= \frac{\int_{E_{n_i-}}^{E_{n_i+}} \sigma_{nx}(E_n)\phi_n(E_n)dE_n}{\int_{E_{n_i-}}^{E_{n_i+}} \phi_n(E_n)dE_n} \\ \Phi_{n_i} &= \overline{\Phi_{n_i}}\Delta E_{n_i} = \int_{E_{n_i-}}^{E_{n_i+}} \phi_n(E_n)dE_n\end{aligned} \tag{5.1}$$

and thus, this term will be dropped from the following calculations.

If the above equation from Cierjacks[34] is rewritten as:

$$\Phi_{x_k} = \sum_{i=1}^{N} \Phi_{n_i} \sum_{j=k}^{24} \Delta R_{x_k} \sum_A N_A \sigma_{nx}^A(E_{n_i}) f_{nx}^A(E_{n_i}, E_{x_j}) \Delta E_{x_j}, \tag{5.2}$$

and the summations are considered as matrix multiplications, the following substitutions can be made:

$$\begin{aligned}
&\mathbf{f}^A \text{ where } f^A_{ji} = f^A_{nx}(E_{n_i}, E_{x_j})\Delta E_{x_j} \\
&\sigma^A \text{ where } \sigma^A_{ij} = \sigma^A_{nx}(E_{n_i})\delta_{ij} \\
&\mathbf{R} \text{ where } R_{ij} = \Delta R_{x_i},\ \forall\, i \leq j \\
&\mathbf{T} = \mathbf{R}\sum_A N_A \mathbf{f}^A \sigma^A,
\end{aligned} \tag{5.3}$$

and the final formula to calculate the charged particle flux from the neutron flux would be

$$\vec{\Phi}_x = \mathbf{T}\vec{\Phi}_n. \tag{5.4}$$

It should be noted that this method will only calculate the charged particle fluxes originating from interactions with the initial isotopes in the mixture, and not with those induced by neutron interactions. In most cases this is very reasonable since only a small fraction of the initial isotopes is actually converted to something else.

The implementation of this feature will require the creation, indexing and inclusion of another set of libraries to contain the various data required for this operation, in particular, the incremental ranges, ΔR, and the charged particle production distributions, $f^A_{nx}(E_{n_i}, E_{x_j})$.

5.2.2 Sensitivity Analysis

By performing a sensitivity analysis, it is possible to determine how an activation problem's solution depends on the transmutation and decay data used in the problem.[35] This analysis can be used in a number of ways including pathway analysis (see Ref. 35), data evaluation, and error estimates of solutions. By understanding how the solution depends on the input data, evaluators can determine which cross-sections and decay constants need to have the most accurate determination. For example, if the solution depends more strongly on the decay constant for tritium than on the transmutation rate of deuterium to tritium, more effort should be spent on improving the evaluation of that decay constant that on refining the $D(n,\gamma)T$ cross-section. Furthermore, the sensitivity coefficients obtained by this method define how the data uncertainties should be weighted when they are summed to find the uncertainty in the final solution.

The method developed by Khursheed[35] and based on work by James[36] is particularly suited to a code based on a time-step ODE solver. To find the sensitivity to a particular datum, x, one differentiates the initial equation:

$$\frac{\partial \dot{\vec{N}}(t)}{\partial x} = \frac{\partial \mathbf{A}}{\partial x}\vec{N}(t) + \mathbf{A}\frac{\partial \vec{N}(t)}{\partial x}.$$

Assuming that the derivatives can be exchanged and rewriting with $N' = \frac{\partial \vec{N}(t)}{\partial x}$:

$$\dot{N'} = \mathbf{A}N' + \frac{\partial \mathbf{A}}{\partial x}\vec{N}(t) \tag{5.5}$$

having almost the same form as the original equation. In this case, if using a time-step based ODE solver, this solution can be easily calculated concurrently with the main solution.

Also, given the solution,

$$\vec{N}(t) = \mathbf{T}\vec{N}_o(t) = e^{\mathbf{A}t}\vec{N}_o(t), \tag{3.4}$$

it is possible to directly define,

$$\frac{\partial \vec{N}(t)}{\partial x} = \frac{\partial \mathbf{T}}{\partial x}\vec{N}_o(t) + \mathbf{T}\frac{\partial \vec{N}_o(t)}{\partial x} = \frac{\partial \mathbf{T}}{\partial x}\vec{N}_o(t) = \frac{\partial \left[e^{\mathbf{A}t}\right]}{\partial x}\vec{N}_o(t), \tag{5.6}$$

since the initial composition is completely independent of the data.

However, the partial derivative of $\mathbf{T} = e^{\mathbf{A}t}$ is very difficult to find. Instead, beginning with the form of solution used to fill each matrix position,

$$\tilde{T}_{ij}(s) = \tilde{F}_{ij}(s)\prod_{k=j+1}^{i} P_k, \tag{5.7}$$

the partial differential can be directly computed,

$$\begin{aligned}\mathcal{L}\left[\frac{\partial T_{ij}}{\partial x}\right] &= \prod_{k=j+1}^{i} P_k \sum_{l=j}^{i} \frac{-\tilde{F}_{ij}}{s+d_l}\frac{\partial d_l}{\partial x} + \tilde{F}_{ij}\prod_{k=j+1}^{i} P_k \sum_{l=j+1}^{i} \frac{1}{P_l}\frac{\partial P_l}{\partial x} \\ &= \tilde{N}_{ij}\left[\sum_{l=j+1}^{i} \frac{1}{P_l}\frac{\partial P_l}{\partial x} - \sum_{l=j}^{i} \frac{1}{s+d_l}\frac{\partial d_l}{\partial x}\right] \\ &= \prod_{k=j+1}^{i} P_k \left[\sum_{l=j+1}^{i} \frac{1}{P_l}\frac{\partial P_l}{\partial x} - \sum_{l=j}^{i} \frac{1}{s+d_l}\frac{\partial d_l}{\partial x}\right]\tilde{F}_{ij}(s).\end{aligned} \tag{5.8}$$

Of all the P_k and d_l, at most two may depend on any given input data value, so this equation can be used to fill the matrix, $\frac{\partial \mathbf{T}}{\partial x}$, with at most one extra calculation to invert $\frac{1}{s+d_l}\tilde{F}_{ij}$ (since the inversion of $\tilde{F}_{ij}$ is performed for the main solution anyway) due to its similar form to $\tilde{F}_{ij}$. This inversion would have to be performed using the Laplace methods because there is a guaranteed multiplicity of pole d_l.

On the other hand, it is important to note that in a 3-D problem, there will be a distinct set of data, x, for each spatial point, and thus, a full sensitivity analysis would require many more computations even though each part of the analysis does not itself involve a significant number of extra calculations.

5.2.3 Relational Databases and Advanced Data Handling

Given the large volumes of nuclear data used by an activation calculation there may be some benefit in using a relational database engine for data lookup. While the implications of database implementation on the code's performance would have to be considered, it would avoid the use of proprietary binary formats for nuclear data, such as that used by ALARA to maximize its performance. Furthermore it would allow for the more straightforward selection of nuclear data to be used in a problem. A variety of different indexes could be used to describe a piece of nuclear data, such as its source, the number of energy groups in the cross-section data, its energy range, its date of evaluation, and so on. The user could then provide a set of criteria by which data would be chosen to solve a problem. These criteria would ensure that the user has access to the best data, possible mixing data from different sources and different distributions.

Similarly, relational databases could be used to store the data created by an activation problem. The current version of ALARA creates a hierarchical binary dump file with one block per initial isotope and one record per block for each interval containing that root isotope. This is another proprietary data format requiring special tools to access it for more advanced post-processing. If this data were to be stored in a standardized database, standard tools could be used to extract and combine in various arrangements required by the user. It may also provide the possibility to increase the archived data's resolution, storing the exact solution of every pc-node in an activation tree for each interval. This volume of data would be of the order of 300 MB for a problem similar to the benchmark problems in chapter 4. While creating and storing a file of such size may not be a problem, randomly accessing individual records to be summed into a single response may be inefficient.

All of these advanced data handling techniques suggest the development of a post-processor. By upgrading the ALARA dump file with an index, a graphical tool could be built to randomly access the file as requested by the user, calculating a wider range of responses than built in to ALARA itself and with more flexibility. For example, the specific activity could be averaged across a user-defined set of zones in order to lower the bulk activity to some acceptable level. A graphical application could be created to allow tailoring of materials, with "slides" and "dials" being used to adjust impurity concentrations in order to lower the material's activation.

5.3 Summary

The ALARA activation code has been developed and validated for the solution of activation problems. The set of basic features has been implemented and a selection of advanced features are also operational.

A physical modeling method has been developed to permit the use of modified linear chains without truncating loop in the activation trees. When combined with a carefully designed truncation algorithm, this permits the use of transfer matrix mathematical techniques. Such techniques are particular important for ensuring efficient solutions when modeling arbitrary hierarchical irradiation schedules exactly. Another new feature of ALARA is the ability to perform so-called reverse calculations.

In order to permit the solution of the straightened linear chains, new mathematical methods have been implemented. The best method is adaptively chosen based on the sub-problem's characteristics at the time of the solution. This ensures an optimum balance of accuracy and speed throughout the problem.

ALARA has been validated against the performance of other activation codes known to perform accurately. The validation benchmarks demonstrated that ALARA's accuracy equals that of the state-of-the-art activation codes, with significantly shorter runtimes. Once the basic features had been validated, ALARA was also able to validate its advanced features by comparing the solutions to a suite of benchmark problems.

With its modern computational techniques and continuing development, it is hoped that ALARA will become a widely used code for the activation analysis of nuclear systems. With its library driven handling of transmutation reactions, it can be used for the analysis of a wide variety of systems, based on energy domain, particle type and possible reaction channels.

References

[1] White, A., "Transmutation and Activation Analysis of Fusion Power Plants," Doctoral Thesis, University of Wisconsin-Madison, (May 1985).

[2] Barstall, R.F., "FISPIN - A Computer Code for Nuclide Inventory Calculations," ND-R-328(R), (October 1979).

[3] Bell, M., "ORIGEN–The ORNL Isotope Generation and Depletion Code," Oak Ridge National Laboratory report ORNL-4628, (May 1973).

[4] England, T.R., "CINDER - A One-Point Depletion and Fission Product Program," Bettis Atomic Power Laboratory report WAPD-TM-334(Rev), (1964).

[5] Bateman, H., *Proc. Cambridge Phil. Soc.* **15** 423 (1910).

[6] Forrest, R.A., and J-Ch. Sublet, "FISPACT3 - User Manual," AEA/FUS 227, (April 1993).

[7] Jung, J., "RACC: Theory and Use of the Radioactivity Code RACC," Argonne National Laboratory report ANL/FPP/TM-122, (May 1979).

[8] Attaya, H., "Input Instructions for RACC-P," Argonne National Laboratory report ANL/FPP/TM-270, (September 1994).

[9] Wang, Q., and D.L. Henderson, “Summary Report for ITER Design Task D10: Updating the Activation Code RACC for ITER Design Analysis,” University of Wisconsin Fusion Technology Institute report UWFDM-977, (1995).

[10] Mann, F.M., “REAC*2: Users Manual and Code Description,” Westinghouse Hanford Company report WHC-EP-0282, (December 1989).

[11] Sanz, J., *et al*, “ACAB, Activation Code for Fusion Applications: User’s Manual V 2.0,” Lawrence Livermore National Laboratory report UCRL-MA-122002, (September 1995).

[12] Sung, T.Y., and W.F. Vogelsang, “DKR: A Radioactivity Calculation Code for Fusion Reactors,” University of Wisconsin Fusion Technology Institute report UWFDM-170, (September 1976).

[13] Henderson, D.L., and O. Yasar, “DKRICF: A Radioactivity and Dose Rate Calculation Code Package: Vols. I & II,” University of Wisconsin Fusion Technology Institute report UWFDM-714, (November 1986). This code package is available from the Radiation Shielding Information Center (RSIC) at Oak Ridge National Laboratory as Computer Code Collection entry CCC-323-DKR.

[14] DKR-PULSAR is a new version of the DKR-ICF code which implements methods from Reference 21 for the exact treatment of pulsed history irradiation. It is being developed by D.L. Henderson and H. Khater at the University of Wisconsin–Madison.

[15] Taylor, N., *et al.*, “Experimental validation of calculations of decay heat induced by 14 MeV neutron activation of ITER materials”, *Fus. Eng. and Design*, **45**, (March 1999).

[16] Cheng, E.T., R.A. Forrest, and A.B. Pashchenko, "Report on the Second International Activation Calculation Benchmark Comparison Study," International Atomic Energy Agency report INDC(NDS)-300, (February 1994).

[17] Wilson, P.P.H., and D.L. Henderson, "Expanding Towards Excellence: Ironing out DKR's Wrinkles," University of Wisconsin Fusion Technology Institute report UWFDM-995, (1995).

[18] Sublet, J.-Ch., J. Kopecky and R.A. Forrest, "The European Activation File, EAF-97 - Cross section library," United Kingdom Atomic Energy Agency Fusion report UKAEA-FUS-351, (June 1997).

[19] Pashchenko, A.B., *et al.*, "FENDL/A-2.0 Neutron activation cross section data library for fusion applications," International Atomic Energy Agency report IAEA(NDS)-173 (October 1998). Data library retrieved online from the IAEA Nuclear Data Section.

[20] Mann, F.M., and D.E. Lessor, "REAC*3 Nuclear Data Libraries," Proceedings of an International Conference on Nuclear Data entitled: Nuclear Data for Science and Technology, held at the Forschungszentrum Juelich, Fed. Rep. of Germany, 13-17 May 1991.

[21] Sisolak, J.E., S.E. Spangler, and D.L. Henderson, "Pulsed/Intermittent Activation in Fusion Energy Reactor Systems," *Fusion Tech.* **21**, 2145 (1992).

[22] Spangler, S.E., "A Numerical Method for Calculating Nuclide Densities in Pulsed Activation Studies," Master of Science Thesis, University of Wisconsin-Madison, (August 1991).

[23] Spangler, S.E., J.E. Sisolak, and D.L. Henderson, "Calculational Models for the Treatment of Pulsed/Intermittent Activation Within Fusion Energy Devices" *Fus. Eng. and Design*, **22**, 349 (July 1993).

[24] de Hoon, M.L.J., E. Greenspan and M.D. Lowenthal, "A Model for Pulsed Activation Accounting for Circulation, Extraction and Makeup," Abstracts of the Thirteenth American Nuclear Society Topical Meeting on the Technology of Fusion Energy, Nashville, TN, (June 1998).

[25] Wilson, P.P.H., "Neutronics of the IFMIF Neutron Source: Development and Analysis," Forschungszentrum Karlsruhe report FZKA-6218 (1999).

[26] Moler, C. and C. Van Loan, "Nineteen Dubious Ways to Compute the Exponential of a Matrix," *SIAM Review*, **20**, 801 (Oct. 1978).

[27] Fukumoto, H., "New Approach to Neutron-Induced Transmutation, Radioactivity and Afterheat Calculations and Its Application to Fusion Reactors," *Nuc. Sci. and Tech.*, **23**, 97, (February 1986).

[28] Wilson, P.P.H., J.E. Sisolak, and D.L. Henderson, "GERAPH: A Novel Approach to the General Solution of Pulsed History Activation Problems," *Fusion Tech.* **26**, 1092 (November 1994).

[29] Lee, E.S., "Computer Engineering: Computer Algorithms, Data Structures, and Languages," Prepared Notes, University of Toronto, (1989).

[30] Sawan, M.E., "FENDL Activation Benchmark: Specifications for the Calculational Activation Benchmark," International Atomic Energy Agency report INDC(NDS)-318, (October 1994).

[31] O'Dell, R.D., *et al.*, "User's Manual for ONEDANT - A Code Package for One-Dimensional, Diffusion-Accelerated, Neutral-Particle Transport," Los Alamos National Laboratory report LA-9184-M, Rev., (February 1989).

[32] Attaya, H., "Radioactivity Computation of Steady State and Pulsed Fusion Reactor Operation," *Fusion Eng. and Design*, **28** 571 (1995).

[33] Wang, Q., and D.L. Henderson, "Pulsed Activation Analyses of the ITER Blanket Design Options Considered in the Blanket Trade-off Study," *Fusion Eng. and Design*, **28** 579 (1995).

[34] Cierjacks, S.W., et al, "Development of a Novel Algorithm and Production of New Nuclear Data Libraries for the Treatment of Sequential (x, n) Reactions in Fusion Material Activation Calculations," *Fus. Tech.*, **24**, 277 (November 1993)

[35] Khursheed, A., "Neutron-Induced Activation of Materials for the First Wall of Conceptual Fusion Reactors," Doctoral Thesis, Imperial College of Science and Technology, London, England, (1989).

[36] James, M.F., "The Calculation of Sensitivities of Nuclide Inventories and Decay Power," Proceedings of the NEA Specialist Meeting on Data for Decay Heat Predictions, Studsvik, Sweden, 7-10 September 1987.

Acknowledgements

At this point I would like to thank all those people who have supported me professionally, technically and morally throughout the development of this research. While I cannot list them all here, certain groups and individuals are deserving of special acknowledgement.

This work was carried out in the Fusion Technology Institute [FTI] of the University of Wisconsin-Madison's Engineering Physics [EP] Department, with partial support from the United States Department of Energy. I gratefully acknowledge the support and feedback of all my FTI and EP colleagues.

Despite the short opportunity I had to know him, the late Prof. Emeritus Charles Maynard provided much inspiration for this work. He always insisted that loops were usually not important, and when they were, the solution could be easily found. I am happy to have proven him correct.

Special thanks are due to Professor Douglass Henderson for presiding over this research. We have both learned much more about the nature of this problem through many hours of debate and discussion over the last six years. ALARA would never have been completed without his guidance.

In addition, Jim Sisolak, Jeff Crowell and Prof. Jake Blanchard have all, at times, been good listeners to help me focus my ideas and develop my methods.

Finally, a very special thanks and recognition for my wife, Laurie Nagus for putting up with the long hours, for listening to my complaining, and most of all, for always believing that I could do it.

Appendix A

Derivation of Recursive Derivative Definition

$$G(s) = \prod_{i=1}^{N} \frac{1}{s+d_i} \tag{A.1}$$

$$\begin{aligned} G'(s) &= \sum_{j=1}^{N} \frac{-1}{s+d_j} \prod_{i=1}^{N} \frac{1}{s+d_i} \\ &= -G(s) \sum_{j=1}^{N} \frac{1}{s+d_j} \end{aligned} \tag{A.2}$$

$$G''(s) = -G'(s) \sum_{j=1}^{N} \frac{1}{s+d_j} + G(s) \sum_{j=1}^{N} (s+d_j)^{-2} \tag{A.3}$$

$$\begin{aligned} G'''(s) &= -G''(s) \sum_{j=1}^{N} \frac{1}{s+d_j} + G'(s) \sum_{j=1}^{N} (s+d_j)^{-2} \\ &\quad + G'(s) \sum_{j=1}^{N} (s+d_j)^{-2} - 2G(s) \sum_{j=1}^{N} (s+d_j)^{-3} \\ &= -G''(s) \sum_{j=1}^{N} \frac{1}{s+d_j} + 2G'(s) \sum_{j=1}^{N} (s+d_j)^{-2} \\ &\quad - 2G(s) \sum_{j=1}^{N} (s+d_j)^{-3} \end{aligned} \tag{A.4}$$

$$\begin{aligned}
G''''(s) &= -G'''(s)\sum_{j=1}^{N}\frac{1}{s+d_j} + G''(s)\sum_{j=1}^{N}(s+d_j)^{-2} \\
&\quad + 2G''(s)\sum_{j=1}^{N}(s+d_j)^{-2} - 4G'(s)\sum_{j=1}^{N}(s+d_j)^{-3} \\
&\quad - 2G'(s)\sum_{j=1}^{N}(s+d_j)^{-3} + 6G(s)\sum_{j=1}^{N}(s+d_j)^{-4} \\
&= -G'''(s)\sum_{j=1}^{N}\frac{1}{s+d_j} + 3G''(s)\sum_{j=1}^{N}(s+d_j)^{-2} \\
&\quad - 6G'(s)\sum_{j=1}^{N}(s+d_j)^{-3} + 6G(s)\sum_{j=1}^{N}(s+d_j)^{-4}
\end{aligned} \tag{A.5}$$

Thus, for n=4,

$$\begin{aligned}
G^{(n)}(s) &= -\frac{(n-1)!}{(n-1)!}G^{(n-1)}(s)\sum_{j=1}^{N}\frac{1}{s+d_j} \\
&\quad + \frac{(n-1)!}{(n-2)!}G^{(n-2)}(s)\sum_{j=1}^{N}(s+d_j)^{-2} \\
&\quad - \frac{(n-1)!}{(n-3)!}G^{(n-3)}(s)\sum_{j=1}^{N}(s+d_j)^{-3} \\
&\quad + \frac{(n-1)!}{(n-4)!}G^{(n-4)}(s)\sum_{j=1}^{N}(s+d_j)^{-4} \\
&= \sum_{i=1} n(-1)^i\frac{(n-1)!}{(n-i)!}G^{(n-i)}(s)\sum_{j=1}^{N}(s+d_j)^{-i}
\end{aligned} \tag{A.6}$$

A.1 Induction Proof

$$G^{(n)}(s) = \sum_{i=1}^{n}(-1)^i\frac{(n-1)!}{(n-i)!}G^{(n-i)}(s)\sum_{j=1}^{N}(s+d_j)^{-i} \tag{A.7}$$

given

$$G^{(0)}(s) = G(s) = \prod_{j=1}^{N}(s+d_j)^{-1} \tag{A.8}$$

First, we solve for n=1:

$$\begin{aligned} G'(s) &= (-1)\frac{0!}{0!}G(s)\sum_{j=1}^{N}(s+d_j)^{-1} \\ &= -G(s)\sum_{j=1}^{N}(s+d_j)^{-1} \end{aligned} \tag{A.9}$$

which matches Equation A.2.

Now, we solve for n=2:

$$\begin{aligned} G''(s) &= (-1)\frac{1!}{1!}G'(s)\sum_{j=1}^{N}(s+d_j)^{-1} + \frac{1!}{0!}G(s)\sum_{j=1}^{N}(s+d_j)^{-2} \\ &= -G'(s)\sum_{j=1}^{N}(s+d_j)^{-1} + G(s)\sum_{j=1}^{N}(s+d_j)^{-2} \end{aligned} \tag{A.10}$$

which matches Equation A.3.

Now, given $G^{(k)}(s)$, we take the derivative, $G^{(k+1)}(s)$, and see if it matches the correct form:

$$G^{(k+1)}(s) = \sum_{i=1}^{k}(-1)^i\frac{(k-1)!}{(k-i)!}\left[G^{(k-i+1)}\sum_{j=1}^{N}(s+d_j)^{-i} - iG^{(k-i)}\sum_{j=1}^{N}(s+d_j)^{-(i+1)}\right] \tag{A.11}$$

letting $l = k+1$:

$$\begin{aligned} G^{(l)}(s) = \quad & \sum_{i=1}^{l-1}(-1)^i\frac{(l-2)!}{(l-i-1)!}G^{(l-i)}\sum_{j=1}^{N}(s+d_j)^{-i} \\ & - \sum_{i=1}^{l-1}(-1)^i\frac{(l-2)!}{(l-i-1)!}iG^{(l-i-1)}\sum_{j=1}^{N}(s+d_j)^{-(i+1)} \end{aligned} \tag{A.12}$$

Now, letting $m = i + 1$ in the second sum:

$$\begin{aligned} G^{(l)}(s) = & \sum_{i=1}^{l-1}(-1)^i \frac{(l-2)!}{(l-i-1)!} G^{(l-i)} \sum_{j=1}^{N}(s+d_j)^{-i} \\ & + \sum_{m=2}^{l}(-1)^m \frac{(l-2)!}{(l-m)!}(m-1) G^{(l-m)} \sum_{j=1}^{N}(s+d_j)^{-m} \end{aligned} \tag{A.13}$$

and recombining the sums:

$$\begin{aligned} G^{(l)}(s) = & -\frac{(l-2)!}{(l-2)!} G^{(l-1)}(s) \sum_{j=1}^{N}(s+d_j) \\ & + \sum_{i=2}^{l-1}(-1)^i \left[\frac{(l-2)!}{(l-i-1)!} + (i-1)\frac{(l-2)!}{(l-i)!}\right] G^{(l-i)} \sum_{j=1}^{N}(s+d_j)^{-i} \\ & + (-1)^l (l-2)!(l-1) G(s) \sum_{j=1}^{N}(s+d_j)^{-l} \end{aligned} \tag{A.14}$$

$$\begin{aligned} & = -G^{(l-1)}(s) \sum_{j=1}^{N}(s+d_j)^{-1} \\ & + \sum_{i=2}^{l-1}(-1)^i \left[(l-i)\frac{(l-2)!}{(l-i-1)!(l-i)} + (i-1)\frac{(l-2)!}{(l-i)!}\right] G^{(l-i)} \sum_{j=1}^{N}(s+d_j)^{-i} \\ & + (-1)^l (l-1)! G(s) \sum_{j=1}^{N}(s+d_j)^{-l} \end{aligned} \tag{A.15}$$

$$\begin{aligned} & = -G^{(l-1)}(s) \sum_{j=1}^{N}(s+d_j)^{-1} \\ & + \sum_{i=2}^{l-1}(-1)^i \frac{(l-1)!}{(l-i)!} G^{(l-i)} \sum_{j=1}^{N}(s+d_j)^{-i} \\ & + (-1)^l (l-1)! G(s) \sum_{j=1}^{N}(s+d_j)^{-l} \end{aligned} \tag{A.16}$$

$$= \sum_{i=1}^{l}(-1)^i \frac{(l-1)!}{(l-i)!} G^{(l-i)} \sum_{j=1}^{N}(s+d_j)^{-i} \tag{A.17}$$

QED.

Appendix B

Other Forms of $1/s$ Expansion

The $1/s$ expansion from section 3.3 can take on many slightly different forms providing different methods for determining the coefficients. First, it is instructive to relate the expansion as shown in equation 3.21 to a simple difference of exponentials. Starting with the Bateman solution (equation 3.12) for a single matrix element,

$$T_{31} = \frac{P_2(e^{-d_1 t} - e^{-d_3 t})}{d_3 - d_1}\frac{P_3}{d_2 - d_1} + \frac{P_3(e^{-d_2 t} - e^{-d_3 t})}{d_3 - d_2}\frac{P_2}{d_1 - d_2} \tag{3.12}$$

and using the standard expansion for the exponential, we get

$$\begin{aligned}
&= P_2P_3\left[\frac{1 - d_1t + \frac{(d_1t)^2}{2} - \frac{(d_1t)^3}{6} - 1 + d_3t - \frac{(d_3t)^2}{2} + \frac{(d_3t)^3}{6} + \dots}{(d_3 - d_1)(d_2 - d_1)}\right. \\
&\qquad\left. + \frac{1 - d_2t + \frac{(d_2t)^2}{2} - \frac{(d_2t)^3}{6} - 1 + d_3t - \frac{(d_3t)^2}{2} + \frac{(d_3t)^3}{6} + \dots}{(d_3 - d_2)(d_1 - d_2)}\right] \\
&= P_2P_3\left[\frac{(d_3 - d_1)[t - (d_3 + d_1)\frac{t^2}{2} + (d_3^2 + d_3d_1 + d_1^2)\frac{t^3}{6} + \dots]}{(d_3 - d_1)(d_2 - d_1)}\right. \\
&\qquad\left. + \frac{(d_3 - d_2)[t - (d_3 + d_2)\frac{t^2}{2} + (d_3^2 + d_3d_2 + d_2^2)\frac{t^3}{6} + \dots]}{(d_3 - d_2)(d_1 - d_2)}\right] \qquad \text{(B.1)} \\
&= P_2P_3\left[\frac{(d_2 - d_1)\frac{t^2}{2} + [d_3(d_1 - d_2) + (d_1^2 - d_2^2)]\frac{t^3}{6} + \dots}{d_2 - d_1}\right] \\
&= P_2P_3\left[\frac{t^2}{2} - (d_3 + d_2 + d_1)\frac{t^3}{6} + \dots\right] \\
&= P_2P_3\, t^2\left[\frac{1}{2} - (d_3 + d_2 + d_1)\frac{t}{6} + \dots\right]
\end{aligned}$$

which has the form of Equation 3.21.

Whether in the Laplace transform domain or the time domain, there is a necessity to calculate coefficients of the form:

$$\{c_i\} = \left\{\sum_{j=1}^{N} d_j\ ,\ \sum_{j=1}^{N} d_j \sum_{k=j}^{N} d_k\ ,\ \sum_{j=1}^{N} d_j \sum_{k=j}^{N} d_k \sum_{l=k}^{N} d_l\ ,\ \dots\right\}. \qquad \text{(B.2)}$$

A different form for these coefficients becomes apparent when $N = 2$ or $N = 3$. The coefficients, $\{c_i\}$, are:

$$\{c_i\} = \left\{d_1 + d_2\ ,\ d_1(d_1 + d_2) + d_2^2\ ,\ d_1\left[d_1(d_1 + d_2) + d_2^2\right] + d_2^3\ , \dots\right\} \qquad \text{(B.3)}$$

or

$$\begin{aligned}
\{c_i\} = \big\{&d_1 + d_2 + d_3,\ d_1(d_1 + d_2 + d_3) + d_2(d_2 + d_3) + d_3^2, \\
&d_1\left[d_1(d_1 + d_2 + d_3) + d_2(d_2 + d_3) + d_3^2\right] + d_2\left[d_2(d_2 + d_3) + d_3^2\right] + d_3^3\ , \dots\big\}.
\end{aligned} \qquad \text{(B.4)}$$

This shows the following pattern, assuming $\{\lambda_{0,j}\} = 1;\ j = [1, N]$:

$$\lambda_{ij} = \sum_{k=j}^{N} d_k \lambda_{i-1,k} \tag{B.5}$$

$$c_i = \lambda_{i1}. \tag{B.6}$$

This last form leads to an efficient way to calculate these coefficients using matrix multiplications. If we form a matrix, M, with elements $m_{ij} = d_j; j \geq i$:

$$\mathbf{M} = \begin{bmatrix} d_1 & d_2 & d_3 & \dots & d_N \\ 0 & d_2 & d_3 & \dots & d_N \\ 0 & 0 & d_3 & \dots & d_N \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & d_N \end{bmatrix}, \tag{B.7}$$

it is clear that $\lambda_1 = [\mathbf{M} \cdot \vec{1}]$ and that $\lambda_i = [\mathbf{M}^i \cdot \vec{1}]$. Therefore,

$$c_i = \lambda_{i1} = \left[M^i \cdot \vec{1}\right]_1. \tag{B.8}$$

Since the direct calculation of

$$\sum_{j_1=1}^{N} d_{j_1} \sum_{j_2=j_1}^{N} d_{j_2} \sum_{j_3=j_2}^{N} d_{j_3} \cdots \sum_{j_{n-1}=j_{n-2}}^{N} d_{j_{n-1}} \sum_{j_n=j_{n-1}}^{N} d_{j_n} = \prod_{l=n}^{1} \sum_{j_l=j_{l-1}}^{N} d_{j_l} \tag{B.9}$$

tends to require $O(N^n)$ calculations, the matrix method above will be highly advantageous since it requires only $O(nN^3)$ calculations.

Once these coefficients have been calculated, they are then used to calculate the time response using Equation 3.21:

$$\begin{aligned} f(t) = t^n \Bigg[\frac{1}{n!} - \frac{t}{(n+1)!} \sum_{l=j}^{i} d_l + \frac{t^2}{(n+2)!} \sum_{l=j}^{i} d_l \sum_{k=l}^{i} d_k \\ - \frac{t^3}{(n+3)!} \sum_{l=j}^{i} d_l \sum_{k=l}^{i} d_k \sum_{m=k}^{i} d_m + \cdots \Bigg]. \end{aligned} \tag{3.21}$$

Appendix C

Users' Guide

The usage of ALARA is fairly straightforward, requiring little knowledge of the code's inner workings. Of course, to ensure that ALARA is well-suited to the problems that you are trying to solve, you are encouraged to read chapters 2 and 3 and understand the physical and mathematical modeling characteristics of ALARA.

This chapter will describe the command-line options of ALARA and then describe the basic support files necessary to run ALARA.

C.1 Command-line Options

`alara [-t` ***tree_output_filename***`] [-h] [-v` ***[n]***`]` ***input_filename***

ALARA currently supports 4 command-line options:

`-h` **Help**

This option will print a short help message describing the command-line options and their usage.

`-t` ***tree_output_filename*** **Tree File**

This option allows you to define the file name for the tree file containing tree creation and truncation information. This information can later be used for basic pathway analysis. The default is to create no tree file. The format of this file is described in section C.4.2.

`-v` ***[n]*** **Verbose**

This option alters the output verbosity. Without this option, only the final results will be displayed. By using this option, details of the calculation are included in the output. The level of detail is controlled by the optional value, n, having a value between 1 (least detail) and 7 (most detail). If no value is given, it defaults to 1.

input_filename **Input File**

This option allows you to define the input filename to be used by ALARA. If no name is specified, the input will be read from `stdin`.

Output from ALARA is written to the `stdout` stream. To capture the output in a file, simply use the standard method of your operating system.

C.2 Input File Description

The input file for ALARA has been designed to ensure that the input information is easy to understand, edit and comment. This is possible by using a very free format permitting comments, blank lines, inclusion of other files, and arbitrary ordering of the input information. After reading the full input file, ALARA performs various cross-checks and cross-references to ensure that the input data is self-consistent. It then goes on to pre-process the data for the calculation. Every attempt has been made to give useful error messages when the data is not consistent.

C.2.1 General

There are 19 possible input block types. These blocks can appear in any order and many blocks can occur more than once, if needed. One block type, `convert_lib`, is only used to convert data library formats, and will cause ALARA to halt if used in an input file. The other 18 input blocks are:

1. `geometry`
2. `dimension`
3. `major_radius`
4. `minor_radius`
5. `volumes`
6. `mat_loading`
7. `mixture`
8. `flux`
9. `spatial_norm`
10. `schedule`
11. `pulsehistory`
12. `truncation`
13. `output`
14. `cooling`
15. `material_lib`
16. `element_lib`
17. `data_library`
18. `dump_file`

Not all input blocks are required, with some having default values and others being unnecessary for certain problems. There are also some input blocks which are incompatible with each other. While superfluous input blocks may go unnoticed (there are occasional warnings), incompatible input blocks will create an error.

Most input blocks allow the user to define their own symbolic names for cross-referencing the various input data. Any string of characters can be used as long as its does not contain any whitespace (spaces, tabs, new-lines, etc.). It is considered dangerous, however, to use a keyword as a symbolic name. If the input file is correct, it will function properly, but if there are errors in the input file, the usage of keywords as symbolic names may make the error message irrelevant. The keywords include those listed in the above list and the keyword "end". While many input blocks of fixed length require nothing to indicate the end of the block, some blocks have a variable length and require the keyword "end" to terminate the block.

Some input elements represent times and can be defined in a number of different units. When this is the case, the floating point time value should be followed by a single character representing the following units:

`[s]econd`	`[m]inute` = 60 seconds	`[h]our` = 60 minutes
`[d]ay` = 24 hours	`[w]eek` = 7 days	`[y]ear` = 52 weeks
`[c]entury` = 100 years		

One input file can be included in another with the `#include` directive, similar to the C programming language. Any number of files can be included, and included files can also contain directives to include other files. The only restriction is that the inclusion must not occur within an input block!

All other lines in which the first non-space character is the pound sign (or number sign) (#) are considered as comments. Comments can also be used after any single word input (an input value with no whitespace) by using the same comment character (#). Such comments extend to the end of the current line. Blank lines are permitted anywhere in the input file.

When length units are implied in the input for sizes and dimensions, it is only important that all implied units be consistent but not what unit is implied.

C.2.2 `geometry`

This input block is only necessary when defining a geometry using the `dimension` input block, but may always be included. It should only occur once. This input block takes a single argument which must be one of the following:

```
point | rectangular | cylindrical | spherical | torus
```

This input block should not be terminated.

```
# problem geometry
geometry spherical
```

If using the `dimension` input block to define the geometry and the type is `torus`, the `major_radius` input block is required and the `minor_radius` block may also be required.

C.2.3 dimension

This input block is used to define the geometry layout, and should be included once for each dimension needed in the problem. The dimension block's first element indicates which dimension is being defined and should be one of the following:

```
x | y | z | r | theta | phi
```

ALARA will check to ensure that only dimensions relevant to the defined geometry are included. For example, defining the 'x' dimension in a spherical problem will generate an error.

The dimension block's next element is the first zone's lower boundary, expressed as a floating point number. This is followed by a list of pairs, one pair for each zone: an integer specifying the number of intervals in this zone in this dimension and a floating point number indicating the zone's upper boundary. This list is terminated with the `end` keyword.

```
# sample dimension for spherical problem
dimension r 1.0      # inside radius
         10 2.0      # 10 intervals in the first zone (1.0,2.0)
         15 3.0      # 15 intervals in the next zone (2.0,3.0)
end
```

Since this method of defining the geometry calculates the the fine mesh intervals' zone membership and volume from the `dimension` data, it is incompatible with the `volumes` input block. Including both will generate an error message.

C.2.4 major_radius and minor_radius

These two input blocks are used to define the major and minor radii of toroidal geometries. They are only needed if defining a toroidal geometry with `dimension` input blocks, and each should only be included once. Furthermore, if the minor radius dimension is defined with a `dimension` block, the `minor_radius` input block is not required. In both cases, these input blocks have a fixed size, with a single argument specifying the radius as a floating point number.

```
# major and minor radii of torus
major_radius     2.0
minor_radius     0.5
```

C.2.5 `volumes`

This input block is used to define the fine mesh intervals' volumes and zone membership. This block can be used instead of the `dimension` method of defining the geometry. If both are used, an error will result. This block should only occur once. Multiple occurrences will result in undefined behavior.

This input block should be a list of pairs, one pair for each interval. Each pair consists of a floating point value for the volume of that interval and the symbolic name of the zone containing that interval. These symbolic names should correspond with the symbolic names given to the zones in the `mat_loading` input block (see section C.2.6). This list must be terminated with the keyword `end`.

```
# list of fine mesh intervals
volumes
   0.5     vacuum_vessel
   1.5     shield_zone
   2.34    blanket_zone
   1.92    first_wall
end
```

C.2.6 `mat_loading`

This input block is used to indicate which mixtures are contained in each zone. This block is a list with one pair of entries for every zone. Each pair consists of a symbolic name for the zone and a symbolic name for the mixture contained in that zone. This list is terminated by the keyword end. This block should only occur once. Multiple occurrences will result in undefined behaviour.

If the geometry is defined using the dimension input blocks, there number of zones defined here must match the number of zones defined in the dimension blocks exactly; if not, an error results. If the volumes method is used to define the geometry, this block uniquely determines the number of zones. The symbolic name for the mixture must match one of the mixture definitions exactly, or be the keyword 'void', indicating that this zone is empty of material.

```
# material loadings for all zones
mat_loading
   vacuum_vessel  VV_materials
   shield_zone    shield_mixture
   blanket_zone   breeding_blanket
   first_wall     liquid_FW
   scrapeoff      void
end
```

C.2.7 mixture

This kind of block is used to define the composition of a mixture. This block can occur as many times as necessary to define all the mixture compositions in the problem. Any mixtures that are defined, but not used in the problem will generate a warning and be removed from the list of mixtures.

The first element of a mixture block is the symbolic name used to refer to this mixture elsewhere in the input file. Following this is a list of triplets with one triplet for each mixture component. The list must be terminated with the keyword 'end'. The first element of each triplet describes the type of that component and should be one of:

```
material | element | isotope | like | target
```

The remaining elements in each triplet are interpreted as follows, based on this first element:

`material` The second element in this triplet is the symbolic name of a material definition existing in the material library (see section C.6.1). The final element is a floating point value representing the relative density of this material. This value, usually between 0 and 1, will be multiplied by the density found in the material library to define the density of this component. It can also be interpreted as the volume fraction of this material in this mixture.

`element` The second element in this triplet is the element's chemical symbol. This element will be expanded into a list of isotopes using the natural isotopic abundances found in the element library (see section C.6.2). The final element is a floating point value representing the relative density of this material. This value, usually between 0 and 1, will be multiplied by the standard theoretical density found in the element library to define the density of this component. It can also be interpreted as the volume fraction of this element in this mixture.

`isotope` The second element in this triplet is a symbolic name for the isotope in the format ZZ-AAA, where ZZ is the chemical symbol and AAA is the mass number, for example, `i-127, ca-40` or `pb-207`. The final element of this triplet is the number density of this isotope in the mixture.

`like` This type of entry is provided as a convenience and indicates that this component is like another user-defined mixture, with a potentially different density. The second element of this triplet is the symbolic name of another mixture definition. If the other mixture definition is not found, an error will result. The triplet's final element is a relative density, used to normalize the density as defined in that mixture's own definition. This might be used when a user-defined mixture makes up part of another mixture. [Hint: it is permissible to define a mixture that is not used in any zones, but only used as part of another mixture.]

`target` This type of entry is used to initiate a reverse calculation (see section 2.3) and define the target isotopes for the reverse calculation. The user can define an arbitrary number of target isotopes. The second element of this triplet is one of the keywords `element` or `isotope`, indicating what kind of target this is. The final element is the symbolic name of either the `element` or `isotope`, as defined in the corresponding entries in this list above. There is no density in this type of entry. If a target is of type `element`, the element will be expanded using the element library to create a list of isotopes.

```
# definition of vacuum vessel
mixture VV_materials
   material SS316-L 1.0
end

# definition of fusion shield
mixture shield_mixture
   like     vacuum_vessel  0.6
   element  he             0.4
end

# definition of fusion breeding blanket
mixture breeding_blanket
   element  li      0.95
   isotope  li-6    6.02e23
   target   isotope h-3
end
```

Note that even if a target is defined in only one mixture, it will cause the whole problem to be run as a reverse problem. There is therefore little purpose in having mixture definitions without targets (such as in this example).

C.2.8 `flux`

This input block defines a set of flux spectra. Since different parts of the irradiation history can have different flux spectra, this block may occur as many times as necessary to represent all the different necessary flux definitions. The first element of this block is a symbolic name, used to refer to this flux spectra definition. The other elements of this block are a filename, a floating point scalar normalization, an integer skip value (see below), and flux type indicator string, respectively.

The flux filename should indicate which file contains this flux information, including path information appropriate to find the file from the directory in which ALARA will be run.

The scalar normalization permits uniform flux scaling at all spatial points (as opposed to the `spatial_norm` information in the next section). All groups of all fluxes in this definition will be multiplied by this value.

The skip value indicates how many N-group flux entries to skip in this file before reading the first flux. This permits the user to have one file with many different flux spectra. For example, if the schedule requires two different flux spectra for N different fine mesh points, the data for the first one may be at the beginning of the file, with a skip of 0, while the data for the second flux definition would be after these first fluxes, with a skip of N.

The last element is a character string indicating the flux file's format. Currently the only supported format is `default`. The default flux file format consists of one list of group fluxes per spatial point. There are no other entries and this can be freely formatted, although comments are not permitted.

```
# flux for first part of irradiation schedule
flux high_yield_flux ../n.transport/machine.flux 1.0 0 default

# flux for second part of irradiation schedule
flux low_yield_flux ../n.transport/machine.flux 1.0 432 default

# flux for final part of irradiation schedule
flux attenuated_high_yield ../n.transport/machine.flux 0.5 0 default
```

[Hint: Different flux definitions might use exactly the same flux values (same flux file and skip value) but a different scaling value.]

C.2.9 `spatial_norm`

This input block allows the user to specify a scalar flux normalization for each fine mesh interval, such as might be required to re-normalize the results of a transport calculation on an approximated geometry. The number of normalizations must be at least as many as the number of defined intervals, regardless of how the intervals are defined (`dimension` vs. `volumes`). If there are too few, an error will result; if there are too many, a warning will result.

This block consists of a list of floating point normalization values, one value for each interval, and requires the `end` keyword to terminate the list.

```
# normalizations to convert cylindrical model
# to toroidal equivalent
spatial_norm
   0.8
   0.85
   0.9
   1.0
   1.03
   1.08
   1.1
end
```

[Hint: if these values are purely a function of problem geometry, and not mixture composition, it is possible that many problems have the same spatial normalization. Put this data in a separate file and `#include` it when you need it.]

C.2.10 `schedule`

This kind of block is used to define a single schedule in the full irradiation history hierarchy. Since the hierarchy may be composed of many schedules, this block might occur many times. The first element in this input block is a symbolic name by which this schedule can be referred to. Following this is a list of items occurring in this schedule. There are two possible types for each item, and their may be an arbitrary list of items in a schedule. This list must be terminated with the keyword '`end`'. See section C.3 for more information about defining schedules.

The first type of item is a simple pulse and the entries for this kind of item are a floating point operating time, a single character defining the units of that operating time, a symbolic flux name, a symbolic pulsing definition name, a floating point post-item delay time, and a single character defining the units of that delay time.

The second type of item is a sub-schedule and the entries for this kind of item are a symbolic name for the sub-schedule, a symbolic pulsing definition name, a floating point post-item delay time, and a single character defining the units of that delay time.

In both cases, if the symbolically named items (flux, pulsing definition, or schedule) are not found during cross-referencing, an error results.

```
# top level schedule
schedule top
    # a schedule of operation defined by 'phase_1_sched'
    #     repeated with a cycle defined by 'phase_1_cycle'
    #     after which there is a 10 week delay
  phase_1_sched phase_1_cycle 10 w
    # a schedule of operation defined by 'phase_2_sched'
    #     repeated with a cycle defined by 'phase_2_cycle'
    #     after which there is a 5 week delay
  phase_2_sched phase_2_cycle 5 w
end

# phase 1 schedule
schedule phase_1_sched
    # a pulsing regimen with 1800 s pulses at flux 'high_yield_flux'
    # pulsed with '5_week_plan_short' followed by a 1 week delay
  1800 s high_yield_flux 5_week_plan_short 1 w
    # a special schedule for cleaning the facility, 'cleaning_sched'
    # with a pulsing history 'cleaning_cycle' followed by no delay
  cleaning_sched cleaning_cycle 0 s
end

# cleaning sched
schedule cleaning_sched
    # two consecutive pulsing regimen, each with one day pulses and
    # pulsing history 'daily_week_long' followed by a 1 day delay
    # the first uses 'low_yield_flux' and the other 'low_energy_flux'
  1 d low_yield_flux daily_week_long 1 d
  1 d low_energy_flux daily_week_long 1 d
end

# phase 2 schedule
schedule phase_2_sched
    # a pulsing regimen with 1 hour pulses at flux
    # 'high_yield_flux' pulsed with '5_week_plan_long' after which
    # there is a 1 week delay
  1 h high_yield_flux 5_week_plan_long 1 w
    # same cleaning phase as in phase 1 schedule
  cleaning_sched cleaning_cycle 0 s
end
```

C.2.11 `pulsehistory`

This kind of input block defines the multi-level pulsing histories referenced in the `schedule` definitions. See section C.3 for more information about defining schedules and histories. Since many different pulsing histories may be used throughout the hierarchy of schedules, this block may occur many times.

The first element of each block is a symbolic name for referring to this pulsing schedule. Following this is a list of pulsing level definition triplets, each consisting of an integer number of pulses, a floating point delay time between pulses, and a single character defining the units of that delay time. Since an arbitrary number of pulsing levels is allowed, this list must be terminated with the keyword '`end`'.

```
# define 5 weeks of the short pulses (1800 s = 1/2 hour)
pulsehistory 5_week_plan_short
   4  90 m   # 4 pulses each day with one every 2 hours
   5  17.5 h # 5 days with 16 hours + 90 minutes delay (overnight)
   5  2.73 d # 5 weeks with 2 days + 17.5 hours delay (weekends)
end

# define 5 weeks of the long pulses (1 h)
pulsehistory 5_week_plan_long
   4  1 h    # 4 pulses each day with one every 2 hours
   5  17 h   # 5 days with 16 hours + 1 h delay (overnight)
   5  2.71 d # 5 weeks with 2 days + 17 hours delay (weekends)
end
```

C.2.12 `truncation`

This fixed sized input block defines the parameters used in truncating the activation trees. See section 2.1.2 for a detailed discussion of the tree truncation issue. The first element of this block is the truncation tolerance and the second is an ignore tolerance. When testing the relative atom loss (or relative production in reverse calculations), any value higher than the truncation tolerance will result in continuing the tree while lower values will result in truncation. If the value is also lower than the ignore tolerance, that node is completely ignored.

```
# defined a 1/10000 tolerance, ignoring 1e-10
truncation   1e-4    1e-10
```

C.2.13 `output`

This kind of input block allows the user to define the output's resolution and format. The first element of an output format block indicates the resolution and should be one of:

```
interval | zone | mixture
```

This is followed by a list of output types and modifiers described in the following table:

keyword	function
component	component breakdown in addition to total response
number_density	number density result of all produced isotopes
specific_activity	specific activity of all radioactive isotopes
total_heat	total decay heat
alpha_heat	total alpha heating
beta_heat	total beta heating
gamma_heat	total gamma heating

```
# output only total zone number densities (no component breakdown)
output zone
   number_density
end

# now output activities and decay heats for zones
#   with component breakdown
output zone
   component
   specific_activity
   total_heat
end

# now output total activities for mixtures
output mixture
   specific_activity
end
```

See section C.4 for information on interpreting the output files generated by ALARA.

C.2.14 cooling

This input block is used to define the after-shutdown cooling times at which the problem will be solved. Multiple occurrences will result in undefined behavior. This block is simply a list of times, where each time consists of a floating point time followed by a single character defining the time's units. Since an arbitrary number of cooling times can be solved, this list must be terminated with the keyword '`end`'.

```
# a wide array of cooling times
cooling
     1 m
     1 d
     1 w
   0.5 y
     1 y
    10 y
     1 c
end
```

C.2.15 material_lib and element_lib

These two input blocks are used to specify the libraries to be used for looking up the definitions of materials and elements when they are given as mixture components (see section C.2.7). Each block has a single element consisting of the filename to be used in each case, including appropriate path information to find that file from the directory where ALARA is being run. For more information on the format of these libraries, see section C.6.

```
material_lib  /alara/data/matlib/magnetic_fusion
element_lib   /alara/data/std.elelib
```

C.2.16 data_library

This input block is used to define the type and location of the nuclear data library. The first element of this block is character string defining the type of library. The subsequent elements indicate the file's location. Currently accepted library types are:

alaralib Standard ALARA v.2 binary library. This library type requires a single filename indicating the library's location.

adjlib Standard ALARA v.2 reverse library This library type requires a single filename indicating the library's location.

eaflib Data library following EAF conventions (ENDF/B). This library type requires two filenames, the transmutation library and the decay library, respectively. These libraries will be read and processed, creating an ALARA v.2 binary library with the name 'alarabin' for use in subsequent calculations. Alternatively, this library could be converted to an ALARA v.2 binary library as a separate process using the convert_lib function.

For both types of ALARA v.2 library, the extension ".lib" will be added to the filename indicated in this input block. Otherwise, all filenames should include appropriate path information to find the file from the directory in which ALARA will be run.

```
# convert and use an EAF formated FENDL2 library
data_library eaflib /alara/data.src/FENDL2/fendlg-2.0
       /alara/data.src/FENDL2/fendld-2.0
```

C.2.17 dump_file

This input block defines the filename to use for the binary data dump produced during a run of ALARA. This is currently used to store the intermediate results during the calculation, and will be extended in the future to allow sophisticated post-processing of the data. This filename should be a valid name for a new file, including path information appropriate for the directory where ALARA will be run. Note that if the dump file already exists, it will be overwritten with no warning. If this input block is omitted, the default name 'alara.dump' will be used.

```
# define a dump file name
dump_file testing/test_problem.dump
```

C.3 Defining Irradiation Schedules

With the added flexibility in irradiation schedule definition comes added complication. All attempts have been made for this to be a straightforward process, and this section should help make it clearer.

Since the hierarchical systems of schedules are based on repetition, the simplest way to develop the input representing any given irradiation schedule is to identify repeated blocks within the schedule (or some portion). Each of these blocks is then given the same treatment, with looking for repetition within each block and so on, until the last stage, where the repeated element is a single pulse.

C.4 ALARA Output File Formats

C.4.1 Output File

As described in section C.1 on command-line arguments, various levels of output are available during the calculation. The part of the output file will contain this verbose output, including confirmation of the input data and details of the cross-referencing and preprocessing of the input.

The second part of the output file shows details on the tree building process, ranging from a simple list of the root isotopes being solved and statistics on the size and speed of the solution, to details on the chain growth and truncation calculations (depending on the verbosity specified on the command-line).

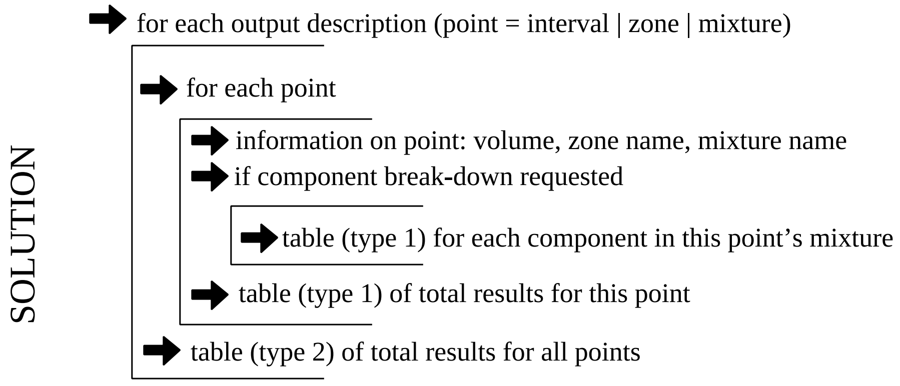

SOLUTION
for each output description (point = interval | zone | mixture)
for each point
information on point: volume, zone name, mixture name
if component break-down requested
table (type 1) for each component in this point's mixture
table (type 1) of total results for this point
table (type 2) of total results for all points

Figure C.1: Output file structure.

The final part of the output file are the results, as requested by the user in the input file. This output will include one section for each output format description given by the user. Each of these sections will be divided into blocks as shown in figure C.1.

There are two types of tables. The first type has a row for each isotope produced in the problem that has a non-zero response. For example, the specific activity in a water filled zone of the benchmark problem of chapter 4 might be:

```
isotope  shutdown         1 h         1 d        30 d         1 y
====================================================================
h-3     1.9419e+08  1.9419e+08  1.9416e+08  1.9330e+08  1.8361e+08
c-14    1.3073e+08  1.3073e+08  1.3073e+08  1.3073e+08  1.3072e+08
c-15    3.9876e+10  0.0000e+00  0.0000e+00  0.0000e+00  0.0000e+00
n-16    7.9376e+13  0.0000e+00  0.0000e+00  0.0000e+00  0.0000e+00
n-17    2.3761e+10  0.0000e+00  0.0000e+00  0.0000e+00  0.0000e+00
n-18    1.5607e+09  0.0000e+00  0.0000e+00  0.0000e+00  0.0000e+00
o-19    1.7727e+09  0.0000e+00  0.0000e+00  0.0000e+00  0.0000e+00
====================================================================
total   7.9443e+13  3.2492e+08  3.2490e+08  3.2403e+08  3.1433e+08
```

The second type of table has a row for each point in the requested resolution, giving the total response at that point. The specific activity in all the benchmark problem's zones might be:

```
Totals for all zones.
zone     shutdown        1 h         1 y
============================================
1       central_zone (void)
2       1.8628e+09  1.3445e+09  5.3928e+07  magnet_coil (TF_Coil)
3       6.7569e+08  2.1433e+08  2.1405e+06  magnet_ins (Insulator)
4       gap1 (void)
5       2.8115e+09  1.4657e+08  4.2568e+07  i_VV_shield1 (Pb_B4C)
6       3.7343e+10  3.3576e+10  3.5317e+09  i_VV_wall1 (Inconel)
.
.
.
25      2.0427e+16  1.4648e+16  1.0754e+14  i_fw_cube (Cu-Be-Ni)
26      5.8139e+15  9.3164e+13  8.8089e+13  i_fw_Be (Be)
27      i_Scrape (void)
28      plasma (void)
29      o_scrape (void)
30      7.6731e+15  1.2204e+14  1.1539e+14  o_fw_Be (Be)
31      2.3041e+16  1.6188e+16  1.4785e+14  o_fw_cube (Cu-Be-Ni)
.
.
.
45      2.3003e+11  2.8462e+06  2.7750e+06  o_b_water1 (water)
46      9.6034e+12  8.1892e+12  1.2897e+12  o_b_steel1 (steel)
47      gap3 (void)
48      6.5203e+12  5.7575e+12  6.0297e+11  o_VV_wall2 (Inconel)
49      4.3277e+11  3.6355e+11  4.6945e+10  o_VV_shield2 (Steel_Water)
50      5.3731e+09  4.8643e+09  5.1452e+08  o_VV_wall1 (Inconel)
51      4.1986e+08  2.0902e+07  6.2052e+06  o_VV_shield1 (Pb_B4C)
============================================
```

If this is a reverse calculation, the entire structure defined above will be repeated for each target isotope.

C.4.2 Tree File

ALARA also produces a so-called tree file to allow some rudimentary pathway analysis (see Section C.1). The tree file contains much information about the creation and truncation of the trees and chains used to calculate the transmutation and activation in the problem.

One tree will be created for each initial isotope. All the information given for this isotope is based on the flux chosen for the truncation calculations of this isotope, namely, the group-wise maximum flux across all the intervals in which the initial isotope exists. An entry for an isotope in the tree will look like this:

```
-(na)->h-3 - (0.00306937)
```

The level of indentation indicates the rank of this isotope in the tree. This can be best seen by viewing the whole file and noting the line's relative indentation. The information given in such an entry is as follows:

reaction type: (na) This indicates the reaction type(s). If multiple reactions lead to this product, the reactions will be separated by commas. The information indicates the emitted particles only. Therefore, in this example, the reaction is an (n,na) reaction. Generally, standard symbols are used, such as 'n' for neutrons, 'a' for alpha particles, 'p','d','t' for the three isotopes of hydrogen, respectively, and 'h' for helium-3. For all neutron reactions, an additional '*' is used to indicate that the product is in an excited isomeric state. Finally, for decay reactions the symbol '*D' is used.

product nuclide: h-3 The product isotope's chemical symbol and atomic number. In cases where the product is in an isomeric state, this will be followed by a letter (m,n,...) indicating which isomeric state.

truncation mode: - This single character indicates the result of the truncation calculation at this node. There are five possible results as follows (see Chapter 2):

- **-** This code indicates that the chain continues normally because this isotope passed all the tests.

* This code indicates that only the radioactive decays of the chain will be followed after this node. This arises when the production does not pass the truncation tolerance test, but ensures that the result includes all the radioactive products. Stable products which are descendants of this node may be calculated if they themselves pass the ignore tolerance test.

/ This code indicates that the chain will be fully truncated at this node, and the result will include this node. This arises when the node is a stable isotope and does not pass the truncation tolerance test, but does pass the ignore tolerance test.

< This code indicates that the chain will be fully truncated at this node and will *not* be included in the result. This arises when the production of this nuclide does not pass either the truncation or the ignore tolerance test.

truncation production: (0.00306937) This indicates the relative production at the end of operation of this nuclide from the initial isotope during the truncation calculation. As explained in Chapter 2, this represents the total production of this nuclide during the whole problem, assuming that none of it is transmuted or decays further. If this production is not calculated, for example, because the chain is only being followed on radioactive reactions and this nuclide is stable, then this entry will be ‘ - ’.

C.5 Binary Reaction Library Format

Because the reaction schemes/chains are created by a depth first search using the data from the transmutation and decay libraries, these libraries need to be accessed extensively and randomly. In the past, such random access was not possible due to limits on mass storage devices. Currently, in a text format, such random access would still be very tedious. To ensure that this random access does not create a drag on ALARA, it is necessary to either store the entire library in memory or use a binary file format. Because the libraries are often quite large (many MB) a simple binary format was designed. This section will describe the formats for the binary files and their indexes, generated in a text format and then appended in binary format to the end of the binary library.

Note that all cross-sections have one more group than the number of neutron groups in the library. This last "group" is used to store the decay rate for this reaction: the product of the total decay rate and the branching ratio (zero for many cases).

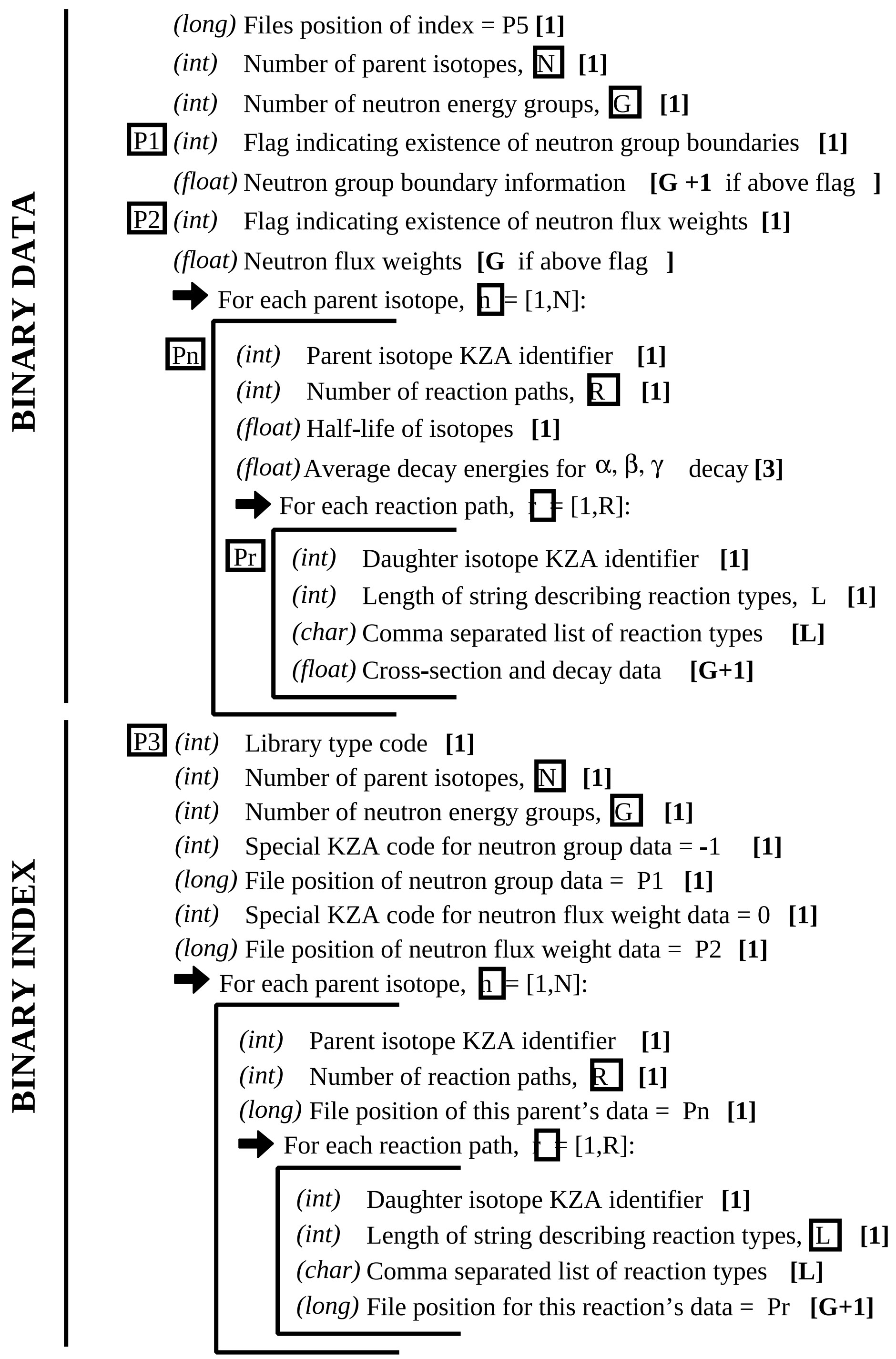
The binary file format is described in figure C.2 by listing, in order, the data written to the file using the format: *(data type)*Description[**size**].

The library used for a reverse problem is identical in format, but for the addition of one entry. Immediately following the average decay energy data for the parent isotope is an entry for the parent isotope's total destruction rate:

(float) Total destruction cross-section [**G+1**]

C.6 Material and Element Library Formats

For the user convenience, ALARA uses both a material and element library. The element library simply contains the natural isotopic breakdown of all the elements. The material library contains the elemental breakdown (using natural elemental compositions) of well-known materials.



BINARY DATA

(long) Files position of index = P5 **[1]**
(int) Number of parent isotopes, $\boxed{N}$ **[1]**
(int) Number of neutron energy groups, $\boxed{G}$ **[1]**
$\boxed{P1}$ *(int)* Flag indicating existence of neutron group boundaries **[1]**
(float) Neutron group boundary information **[G +1 if above flag]**
$\boxed{P2}$ *(int)* Flag indicating existence of neutron flux weights **[1]**
(float) Neutron flux weights **[G if above flag]**
➡ For each parent isotope, $\boxed{n}$ = [1,N]:
- $\boxed{Pn}$ *(int)* Parent isotope KZA identifier **[1]**
- *(int)* Number of reaction paths, $\boxed{R}$ **[1]**
- *(float)* Half-life of isotopes **[1]**
- *(float)* Average decay energies for α, β, γ decay **[3]**
- ➡ For each reaction path, $\boxed{r}$ = [1,R]:
  - $\boxed{Pr}$ *(int)* Daughter isotope KZA identifier **[1]**
  - *(int)* Length of string describing reaction types, L **[1]**
  - *(char)* Comma separated list of reaction types **[L]**
  - *(float)* Cross-section and decay data **[G+1]**

BINARY INDEX

$\boxed{P3}$ *(int)* Library type code **[1]**
(int) Number of parent isotopes, $\boxed{N}$ **[1]**
(int) Number of neutron energy groups, $\boxed{G}$ **[1]**
(int) Special KZA code for neutron group data = -1 **[1]**
(long) File position of neutron group data = P1 **[1]**
(int) Special KZA code for neutron flux weight data = 0 **[1]**
(long) File position of neutron flux weight data = P2 **[1]**
➡ For each parent isotope, $\boxed{n}$ = [1,N]:
- *(int)* Parent isotope KZA identifier **[1]**
- *(int)* Number of reaction paths, $\boxed{R}$ **[1]**
- *(long)* File position of this parent's data = Pn **[1]**
- ➡ For each reaction path, $\boxed{r}$ = [1,R]:
  - *(int)* Daughter isotope KZA identifier **[1]**
  - *(int)* Length of string describing reaction types, $\boxed{L}$ **[1]**
  - *(char)* Comma separated list of reaction types **[L]**
  - *(long)* File position for this reaction's data = Pr **[G+1]**

Figure C.2: The format description of an ALARA V. 2 binary library.

C.6.1 Material Library

```
For each material in library, m = [1,M]:
    (char) Name of material [no white space allowed]
    (float) Nominal density of material [g/cm$^3$]
    (int)   Number of elements in material, E
    For each element,  e  = [1,E]:
        (char) Chemical symbol of element
        (float) Weight fraction of element in material [%]
        (int)   Atomic number of element
```

C.6.2 Element Library

```
For each element in library, e = [1,E]:
    (char) Chemical symbol of element
    (float) Nominal molar mass [g/mol]
    (int)   Atomic number
    (float) Nominal atomic density [g/cm$^3$]
    (int)   Number of naturally occuring isotopes, I
    For each element,  e  = [1,E]:
        (int)   Mass number of isotope
        (float) Natural abundance of isotope in element [%]
```

C.7 Error Messages

-1: Memory allocation error: <*string*>

An error in the runtime allocation of memory occured. "<*string*>" reports the function and variable where the error occurred.

0: Option <*string*> **is not implemented yet.**

An unsupported command-line option was specified: *string*.

1: Only one input filename can be specified: <*string*>**.**

There appears to be more than one input filename on the command-line. This may be caused by an error in the other command line options, or a missing option.

C.7.1 Input Phase

Note that all error messages which occur during the input phase may not report the accurate cause of the error. If there is an error in the input file, ALARA may not immediately recognize the error and then report an error during some later input block. This is particularly true during the first step, reading the input file itself.

Read Input File

100: Invalid token in input file: <*string*>
There is an error in the input file causing it to read an invalid keyword.

101: Unable to open included file: '<*string*>**'.**
The file *string* included in one of the input files can not be openned.

110: Unable to open material library: <*string*>
The file *string* specified in the `material_lib` input block cannot be openned.

111: Unable to open element library: <*string*>
The file *string* specified in the `element_lib` input block cannot be openned.

120: Invalid units in cooling time: %10g %c
The specified cooling time does not have one of the supported time units.

121: No after-shutdown/cooling times were defined.
The `cooling` input block contains no information before the `end` keyword.

130: Invalid dimension type: <*string*>
The type of dimension, *string*, declared in the `dimension` block is not supported.

131: Dimension has no boundaries
The `dimension` block has no zone boundary information before the `end` keyword.

140: Invalid flux type: <*string*>
The flux type, *string*, specified in the `flux` block in not supported.

150: Invalid geometry type: <*string*>
The geometry type, *string*, specified in the `geometry` block is not supported.

160: History <*string*> **is empty**
The `history` input block, *string*, contains no information before the `end` keyword.

170: Material Loading is empty.
The `mat_loading` input block contains no information before the `end` keyword.

180: Target materials for reverse calculations can only be elements or isotopes and not '<*string*>'
The component type, *string*, given for this target material is not supported. It must be either "element" or "isostope".

181: Invalid material constituent: <*string*>
The component type, *string*, specified for this mixture component is not supported.

182: Mixture <*string*> has no components
The `mixture` input block, *string*, contains no information before the `end` keyword.

190: Invalid units in pulse level: %10g %c
The specified pulse level decay time does not have one of the supported time units.

200: Schedule <*string*> is empty
The `schedule` input block, *string*, contains no information before the `end` keyword.

210: Invalid units in schedule item delay time: %10g %c
The specified inter-schedule delay time does not have one of the supported time units.

211: Invalid units in single pulse time: %10g %c
The specified pulse length does not have one of the supported time units.

230: Output type '<*string*>' is not currently supported.
The output type, *string*, specified for this output format is not supported.

240: Unable to open dump file <*string*>
The output "dump" file could not be openned.

Input Checking

300: Cannot define both zone dimensions and interval volumes.
ALARA does not permit the geometry to be defined with both the `dimension` input block and the `volumes` input block. This would result in redundant and possibly inconsistent input.

301: A material loading is given for more zones (%d) than are defined by the zone dimensions (%d). Those extra zones are being ignored.
The number of zones as defined by the `mat_loading` input block does is larger than the number defined by the `dimension` blocks. This is permissible, but may lead to dubious results. The extra zones from the `mat_loading` block will be ignored.

302: Number of zones defined by zone dimensions (%d) matches number of material loadings defined.(%d)"
The number of zones as defined by the `mat_loading` input block does is smaller than the number defined by the `dimension` blocks. This is NOT permissible as it would leave some zones unfilled.

303: Must define either zone dimensions or interval volumes for multi-point problems.
ALARA requires a definition of the geomery using either the `dimension` input block or the `volumes` input block for problems in more than 0 dimensions.

310: Could not find element <*string*> in element library.
The element *string* was not found in the element library. This could be due to an error in the material library, incorrect user input, or an omission in the element library.

311: Could not find material <*string*> in material library.
The material *string* was not found in the material library. This could be due to incorrect user input or an omission in the element library.

330: Duplicate dimensions of type <*string*>.
The dimension *string* was defined more that once in the input file.

331: <*string*1> geometries don't have dimensions of type <*string*2>.
The dimension type *string*2 was defined for geometry type *string*1, which does not allow this kind of dimension.

340: Unable to open flux file <*string*1> for flux <*string*2>.
In the flux definition *string*2 the given flux file *string*1 cannot be openned.

350: Toroidal problems with zone dimensions require a major radius.
All problems defined as having toroidal geometries must define a major radius with the `major_radius` input block.

351: Toroidal problems with zone dimensions require either a minor radius or a radius dimension.
All problems defined as having toroidal geometries must define a minor radius with either a `dimension` block or the `minor_radius` input block.

370: Zone <*string*1> is loaded with a non-existent mixture: <*string*2>
The mixture *string*2 specified to fill zone *string*1 in the `mat_loading` block is not defined in the input file. Either add a new mixture definition or change the name of the mixture to be used for this zone.

380: Component type 'l' of mixture <*string*1> references a non-existent mixture: <*string*2>

The mixture *string*2 specified in the "similar" component of mixture *string*1 is not defined in the input file. Either add a new mixture definition or change the name of the mixture to be used for this definition.

400: Unable to find top level schedule. A top level schedule must not used as a sub-schedule.

All of the defined schedules are referenced as sub-schedules of other schedules. This means that there is no top to the hierarchical schedule system, as required.

410: Flux <*string*1> **for simple pulse item of schedule** <*string*2> **does not exist.**

The flux *string*1 required to calculate the simple pulsing schedule item of schedule *string*2 is not defined.

411: Bad flux file for flux <*string*> **for simple pulse item of schedule** <*string*>**.**

The file for flux *string*1 required to calculate the simple pulsing schedule item of schedule *string*2 cannot be openned.

412: Schedule recursion: <*string*>**.**

There is a loop in the schedule hierarchy. This implies an infinitely long and infinitely complex total irradiation history, which is unphysical. Check the definition of the schedules.

413: Schedule <*string*1> **for subschedule item of schedule** <*string*2> **does not exist.**

The sub-schedule *string*1 defined as a schedule item of schedule *string*2 has not been defined.

414: Pulse history <*string*1> **for item of schedule** <*string*2> **does not exist.**

The pulsing history *string*1 required to calculate a schedule item of schedule *string*2 has not been defined.

420: Zone <*string*> **specified in interval volumes was not found in the material loading.**

The zone *string* specified to contain one of the volumes in the `volumes` input block does not exist.

440: ALARA now requires a binary dump file. Openning the default file 'alara.dmp'.

ALARA uses a binary file to store intermediate results. You can set the name of this file using the `dump_file` input block. Otherwise, the default is used.

441: Unable to open dump file 'alara.dmp'.
The default output dump file could not be openned.

Input Cross-referencing

580: Removing mixture <*string*> not used in any zones.
Mixture *string* was defined in the input file, but is not used in any zones. It's definition is being removed.

620: You have specified too few normalizations. If you specifiy any normalizations, you must specify one for each interval.
The `spatial_norm` input block must contain an entry for each of the fine mesh intervals. It is not permissible to have too few.

621: You have specified too many normalizations. Extra normalizations will be ignored.
It permissible to define too many spatial normalizations, but the results may by dubious. The extra normalizations will be ignored.

622: Flux file <*string*> does not contain enough data.
The flux file *string* does not contain enough data to provide a flux for each of the fine mesh intervals.

C.7.2 Data Library Errors

1000: Data library type <*string*> (%d) is not yet supported.
The specified library type *string* is not supported.

1001: Conversion from <*string*1> (%d) to <*string*2> (%d) is not yet supported.
Conversion between the specified library types *string*1 and *string*2 is not supported.

1001: Conversion from <*string*> (%d) to (%d) is not yet supported.
Conversion between the specified library types *string*1 and *%d* is not supported.

1100: You have specified library type 'alaralib' but given the filename of an 'adjlib' library.
The type of library specified in the input block must match the internally recorded library type.

1101: You have specified library type 'alaralib' but given the filename of an unidentified library.

The type of library specified in the input block must match the internally recorded library type.

1102: You have specified library type 'adjlib' but given the filename of an 'alaralib' library.
The type of library specified in the input block must match the internally recorded library type.

1103: You have specified library type 'adjlib' but given the filename of an unidentified library.
The type of library specified in the input block must match the internally recorded library type.

C.7.3 Programming Errors

In some places, if ALARA reaches that point in the program, it implies an error in the logic of the code. Please report such errors to the code author.

9000: Programming Error:...