M.Eng (Nuclear) Minor Dissertation EEE5004Z

Numerical simulation of nuclear reactor isotope depletion

Tinus Keyser
KYSTIN001

November 2017

Abstract

A program was written in Python to simulate nuclide reactions and burnup in a thermal fission reactor numerically. The program focused on the depletion calculations and used a simplified neutron flux equation. Nuclide data like cross-sections and fission products were read in from ENDF format files that have undergone pre-processing. To solve the more than 500 simultaneous differential equations that describe the varying isotopic concentrations, short-lived decay isotopes and their decay chains were identified and solved with a modified Bateman solution and then the long-lived isotopes concentrations were solved with matrix exponentiation. The flux was calculated to keep the heat output of the reactions constant. The simulation calculations were validated by comparing the output of decay chains with known analytical solutions. The output of the reactor burnup simulation was compared to that of ORIGEN (The Oak Ridge National Laboratory Isotope Generation And Depletion Code) for a Light Water Reactor at constant load to a burnup of 33GWd/ton. The output of the simulation was relatively similar to that of ORIGEN, but differed in some marked ways, e.g. plutonium breeding, which suggested that the neutron flux calculations and neutron absorption by U238 was not similarly modelled as in ORIGEN. By slightly adjusting the neutron absorption of U238 in the simulation, the correspondence between the simulation and the reference output was improved.

Declaration

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signed by candidate

Tinus Keyser PrEng

Acknowledgements

My thanks to Emeritus Professor David Aschman for assisting and supervising this dissertation.

Contents

1 Introduction **1**

2 Isotope depletion codes **2**

2.1 Methods for calculating radioactive growth and decay 2

2.1.1 Bateman solution 3

2.1.2 Matrix exponentiation 4

3 Constructing an isotope depletion code **6**

3.1 Evaluated Nuclear Data Files 6

3.1.1 Reading an ENDF tape 7

3.1.2 Symbol Nomenclature 7

3.1.3 Precision 8

3.1.4 Types of Records 8

3.2 Retrieving data from an ENDF file 9

3.2.1 ENDF interpolations 9

3.2.2 Resonance parameters 10

3.3 Preprocessing 10

3.4 Extracting data from ENDF 11

3.4.1 Reading and storing ENDF cross section data 11

3.4.2 Reading and storing ENDF decay data 12

3.4.3 Reading and storing fission product data 12

3.4.4 Reading and storing isotope mass data 12

3.5 Isotope depletion calculations 12

3.6 Evaluation of short-lived decay chains 13

3.7 Matrix exponentiation 17

3.8 Flux calculation 17

3.9 Validation of simulation code 18

3.9.1 Problem 1: Parent decay to stable daughter 19

3.9.2 Problem 2: Parent decay to stable daughter 19

3.9.3 Problem 3: Long-lived parent decay to short-lived daughter 21

3.9.4 Problem 4: Short-lived parent decay to long-lived daughter 22

4 Results of simulation of isotope depletion **24**
4.1 Simulation output 25
4.2 Effects of resonance escape factor on U238 27
4.3 Increasing U238 capture cross section 29
4.4 Definition of spectral indices used in ORIGEN 29
4.5 Fission product buildup 30
4.5.1 Strontium-90 30
4.5.2 Caesium-137 30
4.5.3 Xenon-135 32
4.5.4 Samarium-149 32
4.6 Decay heat 33

5 Conclusions **35**

Appendices **36**

A Definition of spectral indices used in ORIGEN **37**

B Python code Read_ENDF_data **39**

C Python code Read_ENDF_decay_data **45**

D Python code Read_ENDF_neutron_fission_product_data **49**

E Python code Read_ENDF_mass **52**

F Python code Isotope depletion **57**

List of Figures

3.1 Na24 decay to Mg24 20
3.2 Na24 decay to Mg24 20
3.3 Mg28 decay to Al28 and Si28 22
3.4 La144 decay to Ce144 23

4.1 Neutron flux from simulation 26
4.2 U238 neutron cross sections 27
4.3 Neutron flux from simulation for $\sigma_c^{U238} = 5$b and $\sigma_c^{U238} = 10$b 30
4.4 Sr90 buildup 31
4.5 Cs137 buildup 31
4.6 Xe135 buildup and decay 32
4.7 Sm149 buildup and decay 33
4.8 Decay heat calculation 34

List of Tables

3.1 Output for Na24 decay after 15h 19
3.2 Output for Na24 decay after 150h 21
3.3 Output for Mg28 decay after 3h 22
3.4 Output for La144 decay after 30 min 23

4.1 Comparison of ORIGEN outputs 25
4.2 Comparison of ORIGEN output vs simulation 26
4.3 Comparison of actinide concentrations for different U238 capture cross sections at 33000 MWd/t burnup 29

Chapter 1

Introduction

In a nuclear reactor, the isotopic composition of the nuclear fuel changes constantly. Fissioning of the original nuclides produces lighter nuclides with secondary particles and energy. In addition, nuclides transform to other nuclides through other neutron-induced transmutation reactions and spontaneous radioactive decay. In many applications, it is essential to be able to predict the changes in the nuclear fuel composition. For instance when refuelling a reactor, the makeup and type of the fresh fuel rods will depend on the changes in nuclide concentrations and how these changes are compensated for. This is relevant when designing new reactor concepts and when optimising the reactor core loading of existing reactors alike. Also, it is important to assess the material decomposition of spent fuel after removing it from the reactor and at any time afterwards.

In practice, the changes in nuclear fuel material composition are evaluated by dedicated burnup calculation codes. It is very difficult to simulate the problem in the true time-dependent form because there are many interactions between nuclide concentrations and neutron density distribution. The nuclide concentrations change with different neutron distributions and the neutron density is affected by the composition of the fuel.

The burnup equations can be simplified to a system of simultaneous first order differential equations, which can be formally solved by computing the matrix exponential of the burnup matrix. Since the half-lives of different nuclides vary a lot, the system is extremely stiff. Various methods can be employed to solve the system and the selection of the method depends on the computer memory and processor capabilities.

For this dissertation assumptions were made to simplify calculations and to keep the computer processor requirements within the capabilities of a basic laptop computer. The output of the program was then compared to the output of a well known depletion code, *i.e* ORIGEN (Bell, 1973).

Chapter 2

Isotope depletion codes

There are many different codes used for various nuclear processes, e.g. criticality safety, radiation shielding and isotopic deplection. For instance, in Japan a code called FPGS90 is used by the Japan Atomic Energy Research Institute to calculate nuclides, decay heat and the spectrum of emitted γ-rays (Mara *et al*, 1995). Another example is ISOGEN: Interactive isotope generation and depletion code. (Subbaiah, 2016)

A widely used software system is the Standardised Computer Analyses for Licensing Evaluation (SCALE), developed at Oak Ridge National Laboratory (Bowman, 2011). The module in SCALE used for isotope depletion is the ORNL Isotope Generation and Depletion Code, or ORIGEN. It has been validated against actual data (Ilas *et al*, 2014).

2.1 Methods for calculating radioactive growth and decay

Isotope depletion codes use several approaches to solve depletion problems. As laid out by Gauld *et al* (2011), the amount of a radioactive nuclide present at a given time in an irradiated material can be described by a balance equation:

$$rate\ of\ change = production\ rate - removal\ rate \tag{2.1}$$

The production terms include generation by radioactive decay of a parent nuclide, generation by fission of actinides, and generation by neutron-induced reactions on other nuclides, leading to the formation of the nuclide of interest. The removal terms include radioactive decay and neutron-induced reactions of the nuclide considered. A general expression for the rate at which a nuclide i changes as a function of time may be written as follows:

$$\frac{dN_i}{dt} = \sum_{j=1}^{m} I_{ij}\lambda_j N_j + \bar{\Phi}\sum_{k=1}^{m} f_{ik}\sigma_k N_k - (\lambda_i + \bar{\Phi}_i\sigma_i + r_i)N_i + F_i\,, \quad (i = 1, \ldots, m)\,, \tag{2.2}$$

where

N_i = atom density of nuclide i

λ_i = radioactive disintegration constant of nuclide i

$\bar{\Phi}$ = space- and energy-averaged neutron flux

I_{ij} = branching fractions of radioactive disintegrations from other nuclides j

f_{ik} = branching fractions for neutron absorption by other nuclides k that lead to the formation of species i

σ_i = spectrum averaged absorption cross section

r_i = continuous removal rate of nuclide i from the system

F_i = continuous feed rate of nuclide i

The terms r_i and F_i in Eq.(2.2) are only applicable to various phases of fuel reprocessing or other systems that involve external removal of feed processes that can be described with rate constants. The neutron flux and neutron absorption cross sections in Eq.(2.2) are considered constant over small intervals of time such that the equations describing the production and removal of nuclides may form a set of simultaneous first-order ordinary differential equations with constant coefficients.

There are various methods generally employed to solve the set of equations: the Bateman solution formula, numerical integration of the ordinary differential equations and matrix exponentiation. There are also other methods like exponential moments functions (Harr, 2007).

2.1.1 Bateman solution

A solution for linear decay chains was given by Bateman (1910). For the radioactive decay case of m-nuclide series in a linear chain describing nuclide concentration are as follows:

$$\frac{dN_1}{dt} = -\lambda_1 N_1 \tag{2.3}$$

$$\frac{dN_i}{dt} = \lambda_{i-1} N_{i-1} - \lambda_i N_i \qquad (i = 2, m) \tag{2.4}$$

where λ_i is the decay constant of the ith nuclide.

Assuming zero concentrations of all daughters at time zero
$N_1 \neq 0$ and $N_i = 0$ when $i > 1$

the concentration of the nth nuclide after time t was given by Bateman

$$N_n(t) = \frac{N_1(0)}{\lambda_n} \sum_{i=1}^{n} \lambda_i \alpha_i e^{-\lambda_i t} \tag{2.5}$$

where

$$\alpha_i = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{\lambda_j}{(\lambda_j - \lambda_i)} \tag{2.6}$$

2.1.2 Matrix exponentiation

Analytical depletion codes generally use a solution of simplified linearised buildup and decay chains in which a nuclide concentration is independent of any of its progeny. This approach is taken to avoid difficulties in solving the complete set of equations for a few specific types of transmutations in which an isotope decays to produce one of its precursors, as is encountered in alpha decay of the actinides. The primary interest of this paper involve fuel burnup and decay only, in which case Eq.(2.2) is homogeneous, as no external source term is present $F_i = 0$. In this case, the matrix exponential technique is used to solve the homogeneous system of equations.

For the homogeneous case, the set of equations may be written in matrix notation as

$$\dot{\mathbf{N}}(t) = \mathbf{A}\mathbf{N}(t) \tag{2.7}$$

where

$\mathbf{N}$ = vector of nuclide atom densities

$\mathbf{A} = N \times N$ transition matrix containing the rate coefficients for radioactive decay and neutron absorption.

Each element of the matrix contains the total transition rate constant $\mathbf{A} = -\lambda + \bar{\Phi}\sigma$. Equation (2.7) has the formal solution

$$\mathbf{N}(t) = e^{\mathbf{A}t}\mathbf{N}(0) \tag{2.8}$$

where $\mathbf{N}(0)$ is a vector of initial nuclide atom densities and $e^{\mathbf{A}t}$ is defined as the power series expansion

$$e^{\mathbf{A}t} = \sum_{k=0}^{\infty} \frac{1}{k!}(\mathbf{A}t)^k \tag{2.9}$$

with the additional definition $\mathbf{A}^0 = \mathbf{I}$.

If one can generate this function accurately from the transition matrix, then the solution of the nuclide chain reactions is readily obtained. However, in solving the matrix for

nuclide depletion equations inaccuracies can occur due to the addition of very large and small numbers.

One method of avoiding the inaccuracies is to only use the matrix exponential method for nuclides with a sufficiently small decay constants and then to solve the remaining nuclide concentrations with other methods, e.g. the Bateman solution. The criterion used for ORIGEN (Gauld *et al*, 2011) uses the norm of the transition matrix, defined as the smaller of the maximum-row absolute sum and the maximum-column absolute sum given by Equation 2.10:

$$\|\mathbf{A}\| = \min \left\{ \max_j \sum_i |a_{ij}|, \max_i \sum_j |a_{ij}| \right\} \tag{2.10}$$

The maximum term in the summation for any element in the matrix exponential function cannot exceed $(\|\mathbf{A}\|t)^n/n!$ where n is the largest integer not greater than $\|\mathbf{A}\|t$. To ensure that 16 digit significant precision is maintained, ORIGEN restricts $\|\mathbf{A}\|t$ to

$$\|\mathbf{A}\|t \leq -2\ln(0.001) = 13.8155 \tag{2.11}$$

Nuclides with matrix elements larger than this is solved with the Bateman solution.

Chapter 3

Constructing an isotope depletion code

The isotope depletion code was written in Python 3.5 and executed on a Windows operating system with a Intel i7-3687U CPU operating at 2.10 GHz. The first step for a depletion code is to acquire nuclear data libraries. The completeness and accuracy of these libraries are crucial to ensure good results. The ENDF data used for the simulation was ENDF/B-VI, Release 8 and it was downloaded from the Nuclear Energy Agency website (OECD, 2017).

3.1 Evaluated Nuclear Data Files

Nuclear data like cross sections and isotope masses are stored in a format called ENDF, or Evaluated Nuclear Data Files. ENDF-format libraries are computer-readable files of nuclear data that describe nuclear reaction cross sections, the distributions in energy and angle of reaction products, the various nuclei produced during nuclear reactions, the decay modes and product spectra resulting from the decay of radioactive nuclei and the estimated errors in these quantities (MacFarlane, 2000).

An ENDF-format nuclear data library has a hierarchical structure by tape, material, file and section. (When the first libraries were compiled, the data was stored on punch cards and magnetic tape, hence the term tape and the 80-column data width of ENDF records.) Each of these levels has a characteristic numerical identifier:

- An ENDF 'tape' is a file that contains one or more ENDF materials.
- MAT labels an ENDF material. It is computed from the target Z and A. It equals 100×Z plus an isotope indicator which starts at 25 for the lightest of the common isotopes. The MAT number step by threes to allow for isomers. Thus, 125 is H1, 128 is H2, 2625 is Fe54, 6153 is Pm148m and 9228 is U235.

- MF labels an ENDF file. Files are usually used to store different types of data:
  - **MF=1** contains descriptive and miscellaneous data
  - **MF=2** contains resonance parameter data
  - **MF=3** contains reaction cross sections vs energy
  - **MF=4** contains angular distributions
  - **MF=5** contains energy distributions
  - **MF=6** contains energy-angle distributions
  - **MF=7** contains thermal scattering data
  - **MF=8** contains radioactivity data
  - **MF=9-10** contains nuclide production data
  - **MF=12-15** contains photon production data, and
  - **MF=30-36** contains covariance data
- MT labels an ENDF section. Sections are used to hold different reactions. For example, MT=1 is the total cross section, MT=2 is elastic scattering, MY=16 is the (n,2n) reaction, MT=18 is fission and MT=102 is radiative capture. Full details can be found in the ENDF format manual (Herman and Trkov, 2009)

3.1.1 Reading an ENDF tape

An ENDF tape contains one or more materials in increasing order by MAT. Each material contains several files in increasing order by MF. Each file contains several sections in increasing order by MT. An ENDF 'tape' is built up from a small number of basic structures called 'records', such as TPID, TEND, CONT, TAB1, and so on. These 'records' normally consist of one or more 80-character FORTRAN records. It is also possible to use binary mode, where each of the basic structures is implemented as a FORTRAN logical record. (Herman and Trkov, 2009) The structure of an ENDF data tape (file) is as follows. The tape contains a single record at the beginning that identifies the tape. The major subdivision between these records is by material (identified by the MAT number). The data for a material is divided into files, and each file (identified by the MF number) contains the data for a certain class of information. A file is subdivided into sections, each one containing data for a particular reaction type (identified by the MT number). Finally, a section is divided into records. Every record on a tape contains three identification numbers: MAT, MF, and MT. These numbers are always in increasing numerical order, and the hierarchy is MAT, MF, MT. The end of a section, file, or material is signaled by special records called SEND, FEND, and MEND, respectively.

3.1.2 Symbol Nomenclature

ENDF data use an internally consistent notation based on the following rules derived from FORTRAN.

1. Symbols starting with the letter I, J, K, L, M, or N are integers. All other symbols refer to floating-point (real numbers).
2. The letter I or a symbol starting with I refers to an interpolation code.
3. Letters J, K, L, M, or N when used alone are indices.
4. A symbol starting with M is a control number. Examples are MAT, MF, MT.
5. A symbol starting with L is a test number.
6. A symbol starting with N is a count of items.

All numbers are given in fields of 11 columns. In character mode, floating-point numbers are in one of the following forms:

$$\pm 1.234567 \pm n$$
$$\pm 1.23456 \pm nn,$$

where nn $\leq$ 38 depending on the size of the exponent. Both of these forms can be read by the "E11.0" format specification of FORTRAN.

3.1.3 Precision

The floating point format allows for 7 significant digits. If larger exponents are needed, the format drops back to 6 or 5 significant digits. This is sufficient for the calculations done in this work. For some evaluations like fission product yields, the 1σ uncertainty is also given, although for this work uncertainties were not calculated.

It should also be noted that different versions of ENDF files may contain differing values for instance, of decay half lives. The source of the evaluations used in a specific ENDF evaluation is given in File 1, Section 451.

3.1.4 Types of Records

All records on an ENDF tape are one of six possible types, denoted by TEXT, CONT, LIST, TAB1, TAB2, and INTG. The CONT record has six special cases called DIR, HEAD, SEND, FEND, MEND, and TEND. The TEXT record has the special case TPID. Every record contains the basic control numbers MAT, MF, and MT, as well as the sequence number NS. The definitions of the other fields in each record will depend on its usage as described as described by Herman and Trkov (2009).

3.2 Retrieving data from an ENDF file

There are various repositories for ENDF files, e.g. ENDF/B - United States Evaluated Nuclear Data File, CENDL - China Evaluated Nuclear Data Library or JENDL - Japan Evaluated Nuclear Data Library. Simple cross sections are an example of the general one-dimensional tabulation, or TAB1, record. For example, here is the section for the (n,2n) reaction in U235 from ENDF/B-VI.8:

```
 9.223501+4 2.330250+2          0          0          0          09228 3 16 1
-5.297781+6-5.297781+6          0          0          1         549228 3 16 2
        54          2                                              9228 3 16 3
 5.320521+6 0.000000+0 5.500001+6 5.591713-3 5.750001+6 6.393142-29228 3 16 4
 6.000001+6 1.403257-1 6.250001+6 2.321997-1 6.500001+6 3.126039-19228 3 16 5
 6.750002+6 3.708208-1 7.000002+6 4.051608-1 7.250002+6 4.266690-19228 3 16 6
 7.500002+6 4.477299-1 7.750002+6 4.744135-1 8.000002+6 5.034515-19228 3 16 7
 8.250002+6 5.367261-1 8.500002+6 5.729155-1 8.750002+6 6.113210-19228 3 16 8
 9.000002+6 6.501345-1 9.250002+6 6.831664-1 9.500002+6 7.146096-19228 3 16 9
 9.750002+6 7.435087-1 1.000000+7 7.708144-1 1.025000+7 7.968839-19228 3 16 10
 1.050000+7 8.184777-1 1.075000+7 8.271063-1 1.100000+7 8.267420-19228 3 16 11
 1.125000+7 8.354444-1 1.150000+7 8.380148-1 1.175000+7 8.340168-19228 3 16 12
 1.200000+7 8.191227-1 1.219440+7 7.909012-1 1.225000+7 7.827817-19228 3 16 13
 1.250000+7 7.456549-1 1.275000+7 7.077953-1 1.300000+7 6.704240-19228 3 16 14
 1.325000+7 6.265299-1 1.350000+7 5.840038-1 1.375000+7 5.430612-19228 3 16 15
 1.400000+7 5.035570-1 1.425000+7 4.756169-1 1.450000+7 4.480091-19228 3 16 16
 1.475000+7 4.212566-1 1.500000+7 3.951591-1 1.525000+7 3.686654-19228 3 16 17
 1.550000+7 3.434944-1 1.575000+7 3.200375-1 1.600000+7 2.983213-19228 3 16 18
 1.650000+7 2.623509-1 1.700000+7 2.320431-1 1.750000+7 2.076704-19228 3 16 19
 1.796240+7 1.891968-1 1.800000+7 1.878874-1 1.850000+7 1.689812-19228 3 16 20
 1.900000+7 1.538259-1 1.950000+7 1.405132-1 2.000000+7 1.300024-19228 3 16 21
 0.000000+0 0.000000+0          0          0          0          09228 3 099999
```

The first line is the HEAD record; it contains the ZA value (1000*Z+A) and the AWR value (ratio of target mass to neutron mass). The second card starts the TAB1 record and contains the reaction Q value (-5.297781 MeV), the number of interpolation ranges and the number of energy points in the tabulation (54). The third line gives the interpolation data which is 54 pairs linearly interpolated. The rest of the record contains the tabulation given as energy, cross-section pairs with energies in eV and cross sections in barns. The last line is the SEND record.

3.2.1 ENDF interpolations

Most ENDF data is given as a table of values on a defined grid with an interpolation law to to define the values between the grid points. Simple one 0-dimensional 'graph paper' interpolation schemes and a special Gamow interpolation law for charged-particle cross-

sections are provided. The limitation to 'graph paper' interpolation schemes causes some problems for reactions that are a sum of processes with different characteristic shapes.

The classic example of this is the total cross section at low energies. At zero temperature, the elastic cross section tends to be constant for many materials, and it can be represented well using linear-linear interpolation. But the radiative capture cross section usually goes like $1/v$, and it is best described using log-log interpolation. Clearly, the sum of these two reactions will be all right at the grid points, but values intermediate between the grid points cannot be calculated with either linear-linear or log-log interpolation.

For this reason, summation cross-sections such as MT=1 (total cross section), MT=4 (total inelastic) and sometimes MT=18 (total fission), must not be considered fundamental. They must always be reconstructed from the sum of their parts.

There are various software suites to convert evaluations in the NDF format into forms useful for practical applications, e.g. NJOY Nuclear Data Processing System, developed by the Los Alamos National Laboratory (MacFarlane and Kahler, 2010) and the PREPRO system from the IAEA (Cullen, 2015).

3.2.2 Resonance parameters

To include all energy points for cross sections so that the sharp peaks and valleys due to resonances would lead to very large files for the heavier isotopes. ENDF uses resonance cross section formulas to calculate the elastic, capture and fission cross sections over a defined 'resonance range'. The format allows for Single-level Breit-Wigner, Multi-Level Breit-Wigner, Reich-Moore, etc. The preprocessor will reconstruct pointwise (energy-dependent) cross sections from ENDF resonance parameters and interpolation schemes.

3.3 Preprocessing

As explained, ENDF data cannot be immediately used, but need some preprocessing to do interpolations, calculate resonances and to do Doppler broadening. PREPRO was obtained from the International Atomic Energy Agency website (IAEA, 2017) for this purpose.

The preprocessing programs were run in the order suggested by Cullen (2015). First was ENDF2C which converts the data from the old FORTRAN 'E-less' numbering to modern standards, e.g. 1.234567+3 to 1234.567. The output from ENDF2C was then run through LINEAR. This preprocessor converts all tables to a linear interpolation. ENDF format allows cross sections to be represented as tables of data points using a number of different interpolation laws between tabulated points; in order to obtain accurate results it is important to interpret the data using these interpolation laws. The interpolation

laws are very useful during evaluation, but can present problems when they are used in applications. The subsequent use of the data can be greatly simplified and the accuracy of results improved by first linearising all of the cross sections, i.e., replace the original tabulated data points and interpolation law by a new table where one can use linearly interpolation between tabulated points to within any required accuracy.

Next the preprocessor RECENT is applied. It adds the contribution of resonances to the cross sections. ENDF format allows cross sections to be represented as a contribution of resonance parameters and tabulated background corrections. This code will add the resonance contribution to the background cross sections in order to define the cross sections as linearly interpolable tables at 0 Kelvin (cold). Therefore subsequent codes need only deal with tabulated, linearly interpolable, 0 Kelvin cross sections.

After RECENT is applied, the cross sections need to be Doppler broadened to the appropriate temperature, in this case taken as 600 Kelvin to represent the average fuel temperature, by using SIGMA1. As in the case of LINEAR and RECENT all cross sections read and written by this code are tabulated, linearly interpolable. All subsequent codes need not explicitly consider temperature effects and need only deal with tabulated, linearly interpolable cross sections at a given temperature.

3.4 Extracting data from ENDF

The complete ENDF tape is over 100MB in size and almost two million lines in length. Searching for data in it while doing depletion calculations would place a heavy toll on memory and processor resources. Therefore, only the required data like cross sections and fission yields were extracted and stored in separate files that are used by the depletion code.

3.4.1 Reading and storing ENDF cross section data

Since the numerical simulation was going to be for an Light Water Reactor, to simplify matters only thermal neutron data was used to simply calculations. A program, *Read_ENDF_data* (see Appendix B), was written that would read through the linearised ENDF data and output a file with isotopes and their cross sections at a specified neutron energy for different reactions. The program takes for input MF, MT, the Doppler broadened temperature and the neutron energy, in this case 0.0253 eV. It then reads sequentially through the file looking for the first occurrence of MF MT. When found, it extracts the isotope ZA and the number of data pairs. The data pairs are read into an array and are searched for the required energy. If the required energy is not one of the data points, the program will interpolate to find the cross section. The values are then written to an output file. This is repeated for all isotopes.

The following cross-sections in MF=3 had non-zero values for thermal neutrons:

MT=18 (n,f)
MT=101 (n,γ)
MT=107 (n,α)

3.4.2 Reading and storing ENDF decay data

Radioactive decay data is stored in MT=8 and MF=457. It contains the decay type, energy, half-life, branching fractions if there is more than one type of decay and the isomeric state of the daughter product. A program, *Read_ENDF_decay_data* (see Appendix C), was written to extract the decay data. It searches through the ENDF tape for the MT=8 and MF=457 records. It then extracts for each isotope the half-life, number of decay modes, the reaction type, isomeric state for the daughter product, the total decay energy and the branching fraction for each decay mode. It then determines the daughter products from the decay mode and outputs all the data to the output file.

3.4.3 Reading and storing fission product data

Independent fission product yield data is stored in MT=8 and MF=454. A program, *Read_ENDF_neutron_fission_product_data* (see Appendix D), was written to extract the fission product data. It searches through the ENDF tape for the MT=8 and MF=454 records. It then extracts for each fissionable isotope the fission product, the fission yield and the 1σ uncertainty for the yield, all of which is written to an output file.

3.4.4 Reading and storing isotope mass data

Isotope masses were obtained from MT=1 and MF=451. In ENDF the mass is stored in AWR, the ratio of the mass of the material to that of the neutron. A program, *Read_ENDF_mass* (see Appendix E), was written to extract the isotope masses. It searches through the ENDF tape for the MT=1 and MF=451 records. It then extracts for each isotope the AWR value and writes it to an output file.

3.5 Isotope depletion calculations

A program, *Isotope_depletion* was written to do isotope depletion calculations for a LWR (see Appendix F). The program algorithm was loosely based on ORIGEN - The Oak Ridge National Laboratory Generation and Depletion code (Bell, 1973). The basic sequence of the program is as follows:

- Initialise variables
- Read in isotopes and their masses
- Read in decay data and construct dictionaries with decay products
- Read in cross-section data
- Read in fission product data
- Initialise four production and loss vectors for each isotope
  - Parents for decay
  - Parents for neutron absorption
  - Radioactive decay
  - Neutron absorption
- Initialize the concentration vector, either from a given fuel composition or read from a file
- Initialise the transformation matrix
- Start the main loop
  - Calculate the flux
  - Calculate the transformation matrix
  - Find short-lived decay chains
  - Evaluate short-lived decay chains
  - Create new concentration and transmutation matrices by removing short-lived chain concentrations and decay constants
  - Calculate new concentration matrix
  - Return short-lived results to concentration matrix
  - Calculate heat release from fission, radioactive decay and neutron capture
  - Write output to file

3.6 Evaluation of short-lived decay chains

Short-lived isotopes cannot be successfully be evaluated with the matrix exponential methods, as the norm of the transition matrix must be less than $2ln(1000)$ (Bell, 1973). The isotope depletion program calculates the quantity of a short-lived isotope using a generalised form of the Bateman equation as suggested by Strenge (1995). The program searches for short-lived isotopes. When one is found, a recursive routine generates all the decay chains forming from the short-lived isotope until a sufficiently long-lived daughter is found to terminate the chain. For a decay chain $A \rightarrow B \rightarrow C$, the algorithm will also return $B \rightarrow C$, so the unique chains are selected after all the isotopes have been evaluated.

Then for each long-lived daughter, all the chains leading to its formation is evaluated. The general solution for evaluating the remaining concentrations of the short-lived chains

can be summarised by the following four equations:

$$A_c(t) = \sum_{i=1}^{c} K_{ci} e^{-\lambda_i t}$$

$$K_{11} = A_1(0)$$

$$K_{cn}(n = 1 \to c-1) = \frac{\sum_{i=n}^{c-1} f_{ic} \lambda_i K_{in}}{\lambda_c - \lambda_n}$$

$$K_{cc} = A_c(0) - \sum_{n=1}^{c-1} K_{cn}$$

where

$A_c(t)$ = quantity of chain member c at time t (atoms)

λ_c = radioactive decay constant for chain member c

f_{nc} = fraction of precursor (chain member n) that results in production of chain member c

λ_n = radioactive decay constant for chain member n

K = array to store recursive values

When using these equations to determine isotope depletion, another problem arises. Take for example the decay chains leading to Strontium-90. The decay chain search yields five unique decay chains:

$$^{90}\mathrm{Kr} \xrightarrow{\beta-} {}^{90}\mathrm{Rb} \xrightarrow{\beta-} {}^{90}\mathrm{Sr} \quad (1)$$

$$^{90}\mathrm{Kr} \xrightarrow{\beta-} {}^{90m}\mathrm{Rb} \xrightarrow{\beta-} {}^{90}\mathrm{Sr} \quad (2)$$

$$^{90}\mathrm{Kr} \xrightarrow{\beta-} {}^{90m}\mathrm{Rb} \xrightarrow{\gamma} {}^{90}\mathrm{Rb} \xrightarrow{\beta-} {}^{90}\mathrm{Sr} \quad (3)$$

$$^{91}\mathrm{Kr} \xrightarrow{\beta-} {}^{91}\mathrm{Rb} \xrightarrow{\beta-,n} {}^{90}\mathrm{Sr} \quad (4)$$

$$^{92}\mathrm{Kr} \xrightarrow{\beta-,n} {}^{91}\mathrm{Rb} \xrightarrow{\beta-,n} {}^{90}\mathrm{Sr} \quad (5)$$

To determine the final concentrations of all the isotopes involved in the decay chains, one cannot just evaluate the five decay chains and add up the resulting concentrations. (Note that for this thesis concentration means number of atoms and has g-atoms as a unit. In some literature concentration is given as number of atoms per volume, but since the density of nuclear fuels differ and for computational ease, the first approach is used.) For instance, $^{90}\mathrm{Kr} \xrightarrow{\beta-} {}^{90m}\mathrm{Rb}$ will be evaluated twice. The problem is then what initial concentrations to use when it is evaluated for the second time. As $^{90m}\mathrm{Rb}$ (excited state of $^{90}\mathrm{Rb}$) decays to both $^{90}\mathrm{Rb}$ and $^{90}\mathrm{Sr}$. Also, only radioactive decay is considered for the short-lived isotopes in the decay chain, but the long-lived daughter at the end of the

decay chain may have a significant cross section for neutron interactions as well and that have to be calculated with the matrix exponential method. One approach to handle this problem is to set the concentration of the long-lived daughter to zero when evaluating the decay chain and adding the result to the concentration after evaluation by the matrix exponential method of the initial concentration.

Each evaluation of the decay chains will yield the concentration for the parent isotope, so it must only be taken once. The intermediate products present more of a challenge where a parent-daughter pair is evaluated more than once. For the first evaluation, the initial concentrations are used for both the parent and daughter. For subsequent evaluations, the resultant concentrations must not be added to the final results. The parent-daughter concentrations are kept at the initial concentration so to yield the correct concentrations further down the decay chain. For an example, assume the initial concentrations for ^{90}Kr, ^{90}Rb, ^{90m}Rb, ^{91}Kr, ^{91}Rb and ^{92}Kr as say 10 gram-atom each. Taking a decay interval of 100 seconds, if we evaluate chain 1, the resultant concentrations in g-atom are:

Isotope	Resultant concentration
^{90}Kr	1.1711
^{90}Rb	12.143
^{90}Sr	5.6268

Note that for ^{90}Sr *an initial concentration of zero was used.*

After evaluation of chain 2:

Isotope	Resultant concentration
^{90}Kr	1.1711
^{90m}Rb	8.5320
^{90}Sr	2.4693

When evaluating decay chain 3, one must note that the parent-daughter decay chain $^{90}\text{Kr} \xrightarrow{\beta-} {}^{90m}\text{Rb}$ and 37090 $\xrightarrow{\beta-} {}^{90}\text{Sr}$ have already been evaluated, but $^{90m}\text{Rb} \xrightarrow{\gamma} {}^{90}\text{Rb}$ has not. To obtain the correct value for ^{90}Rb, the initial concentration for ^{90m}Rb is set at 10 g-atom and that of ^{90}Rb at zero, but the result for ^{90m}Rb is not added to the resultant concentration. Then we find:

Isotope	Resultant concentration
^{90}Kr	1.1711
^{90m}Rb	8.5320
^{90}Rb	0.0465
^{90}Sr	0.0117

Chain 4 has a new parent, ^{91}Kr, so the chain is evaluated with the original concentrations.

Isotope	Resultant concentration
^{91}Kr	3.0718E−03
^{91}Rb	6.6246
^{90}Sr	1.3372E−05

Chain 5 has a new parent, ^{92}Kr. However, ^{91}Rb has already been evaluated as a parent for ^{90}Sr, so its concentration is set at zero.

Isotope	Resultant concentration
^{92}Kr	5.3470E−16
^{91}Rb	1.0400E−03
^{90}Sr	2.2600E−09

Now we can calculate the final concentrations. For the parent, ^{90}Kr, the value from the first evaluation can be used, i.e. 1.1711. For ^{90}Rb, since it has two different parents, the decay result from ^{90}Kr and ^{90m}Rb is added to give 12.189. The isotope ^{90m}Rb has only one parent, we we take its value from decay chain 2, 8.5320. For ^{91}Kr and ^{92}Kr we can take the result from chain 4 and 5, 3.0718-03 and 5.3470E-16. Also from chain 4 and 5, we can add up to 6.6256 for isotope ^{91}Rb. Finally, for the long-lived daughter ^{90}Sr, the results from all the decay chains are added up to give a concentration of 8.1078.

Isotope	Final concentration
^{90}Kr	1.1711
^{90}Rb	12.189
^{90m}Rb	8.5320
^{91}Kr	$3.0718E-03$
^{92}Kr	$5.3470E-16$
^{91}Rb	6.6256
^{90}Sr	8.1078

So to summarise the rules for evaluating a set of short-lived decay chains:

- Find all decay chains that end in a common long-lived daughter.
- Evaluate the first decay chain. The initial concentrations are used for all isotopes except the long-lived daughter, which is set at zero. The parent result is stored as the final value and the daughters are stored in a temporary array.
- Repeat for the following chains:
    - Use the initial concentrations for the parent.

- If a daughter has been evaluated already, its concentration is set to zero.
- The long-lived daughter concentration is set to zero.
- Add the results for the subsequent daughters to those in the temporary array.

3.7 Matrix exponentiation

To do matrix exponentiation, ORIGEN uses a recursion relation that does require storage of the entire matrix (Bell, 1973). The expression for one nuclide in Equation 2.2 is given by

$$x_i = \sum_{n=0}^{\infty} C_i^n \tag{3.1}$$

where C_i^n is generated by a recursive relation

$$C_i^0 = x_i(0) \tag{3.2}$$

$$C_i^{n+1} = \frac{t}{n+1} \sum_{j=1}^{N} a_{ij} C_j^n \tag{3.3}$$

where a_{ij} is an element in the transition matrix that represents the first-order rate constant for the formation of species i from species j and t is a user specified time step.

The advantage of this method is that only the vector $\mathbf{C}^n$ needs to be stored, which was important in the time when ORIGEN was originally developed when computer memory was at a premium. With modern computers memory is not such a constraint anymore, so in the program matrix exponentiation was done with the `expm` function in the NumPy package for scientific computing in Python. It computes the matrix exponential using Padé approximation.

3.8 Flux calculation

The neutron flux calculation in the simulation code for this work was done by assuming a constant power output. Then the flux was the quantity necessary to achieve the power output taking into consideration the concentrations of all fissionable isotopes multiplied by the fission energy yields, similarly for neutron capture energy release and adding the radioactive decay heat release. The flux obtained in this way was on average 15% less than than the flux reported by ORIGEN. For i isotopes:

$$P_{th} = \Phi(H_f + H_c + H_{na}) + P_{rd}$$
$$\Phi = \frac{P_{th} - P_{rd}/t}{H_f + H_c + H_{na}}$$

where

P_{th} = Reactor thermal power output

$P_{rd} = \sum_i E_i \lambda_i N_i(t)$ = Energy released by radioactive decay

$H_f = \sum_i \sigma_f^i E_f^i N_i$ = Radiant fluence for fission

$H_c = \sum_i \sigma_c^i E_c^i N_i$ = Radiant fluence for neutron capture

$H_{na} = \sum_i \sigma_\alpha^i E_\alpha^i N_i$ = Radiant fluence for (n,α) reactions

t = Time interval

According to Gauld *et al* (2011), ORIGEN used a similar methodology to calculate flux. The neutron flux is determined as follows:

$$\Phi(t) = P/(1.6 \times 10^{-19} \sum_i \sum_{\substack{j \\ j \neq i}} N_i(t) \sigma_{ij} Q_{ij}) \,, \tag{3.4}$$

where

P = specific power in MW
N_i = number of atoms for nuclide i
σ_{ij} = microscopic cross section for nuclide i and reaction type j (e.g. fission and capture) in square centimeters
Q_{ij} = recoverable energy released from fission and neutron capture in MeV
Φ = instantaneous value of the neutron flux in neutrons/(cm^2·s)
1.6×10^{-19} = unit conversion constant (MeV/s·MW^{-1}).

Values of the recoverable energy for fission and (n, γ) reactions, which are tabulated for 24 fissile isotopes and other important neutron-absorbing nuclides, are based on data from ENDF/B evaluations. A value of 5 MeV is used for the capture energy of all other nuclides, which are generally of lower importance. In the simulation, the actual energy values from the ENDF/B evaluations were used for increased accuracy. The energy carried off by neutrinos is not included.

3.9 Validation of simulation code

To validate the accuracy of the different methods, the simulation code was tested against sample problems where the analytical solutions are available. The results were given to ten decimals to confirm that the incremental calculation approach that the code uses can provide sufficient accuracy.

3.9.1 Problem 1: Parent decay to stable daughter

Solution method: Matrix exponential method.
Estimate the concentration of Na24 and Mg24 after time interval of 15h for the decay chain $^{24}\text{Na} \xrightarrow{\beta-} {}^{24}\text{Mg}$.
Initial Na24 concentration = 1.0 g.atom
$T_{\frac{1}{2}}$ of Na24 = 15h and Mg24 is stable.
Answer:
By inspection, Na24 = 0.5 g.atom and Mg24 = 0.5 g.atom Solution from simulation:
Since $T_{\frac{1}{2}} = 15h \geq \frac{-ln(2)}{ln(0.001)} \times T_{step}$ where $T_{step} = 1h$, Na24 and Mg24 (stable) are considered to be long-lived in this case and hence the matrix exponential method was used by the code to solve this problem. The code output is listed in Table 3.1 and graphed in Figure 3.1.

T (h)	Na24	Mg24
0	1.0000000000E+00	0.0000000000E+00
1	9.5484160391E-01	4.5158396090E-02
2	9.1172248856E-01	8.8277511442E-02
3	8.7055056330E-01	1.2944943670E-01
4	8.3123789614E-01	1.6876210386E-01
5	7.9370052598E-01	2.0629947402E-01
6	7.5785828326E-01	2.4214171674E-01
7	7.2363461872E-01	2.7636538128E-01
8	6.9095643998E-01	3.0904356002E-01
9	6.5975395539E-01	3.4024604461E-01
10	6.2996052495E-01	3.7003947505E-01
11	6.0151251804E-01	3.9848748196E-01
12	5.7434917750E-01	4.2565082250E-01
13	5.4841248985E-01	4.5158751015E-01
14	5.2364706141E-01	4.7635293859E-01
15	5.0000000000E-01	5.0000000000E-01

Table 3.1: Output for Na24 decay after 15h
The code yielded the correct answer to a very good accuracy.

3.9.2 Problem 2: Parent decay to stable daughter

Solution method: Matrix exponential method.
Estimate the concentration of Na24 and Mg24 for $^{24}\text{Na} \xrightarrow{\beta-} {}^{24}\text{Mg}$ after time interval of 150h.
Initial Na24 concentration = 1.0 g.atom
$T_{\frac{1}{2}}$ of Na24 = 15h and Mg24 is stable.
Answer:

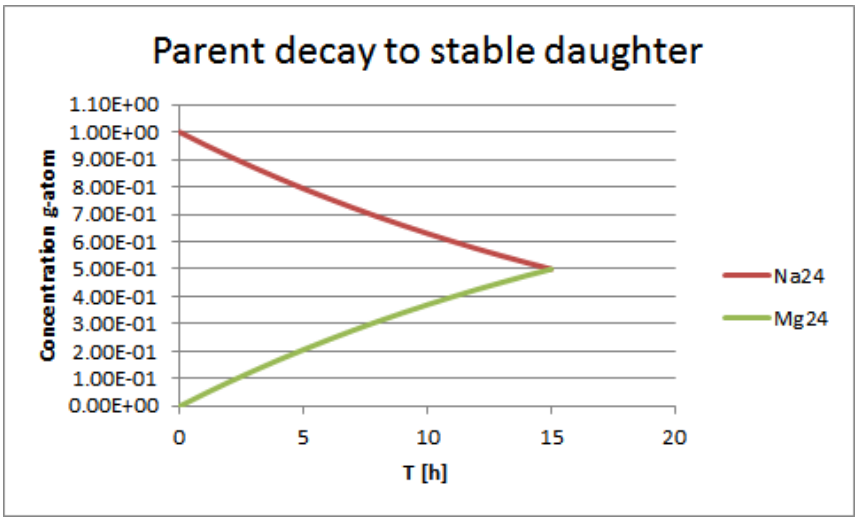

Figure 3.1: Na24 decay to Mg24

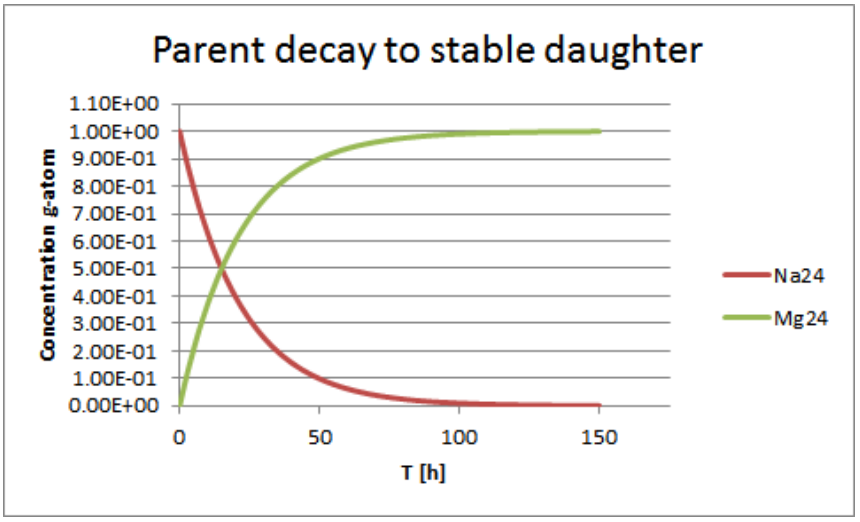

Figure 3.2: Na24 decay to Mg24

Analytical results
Na24 = $N_{Na24}(0)e^{-\lambda t} = 1.0 \times e^{\frac{-ln(2)}{15} \times 150}$ = 9.7656250003E-04 g.atom and
Mg-24 = 0.9902343750 g.atom (balance)
Solution from simulation:
Since $T_{\frac{1}{2}} = 15h \geq \frac{-ln(2)}{ln(0.001)} \times T_{step}$ where $T_{step} = 1h$, Na24 and Mg24 (stable) are considered to be long-lived in this case and hence the matrix exponential method was used by the code to solve this problem. The code output and graph is listed in Table 3.2 and Figure 3.2.

T (h)	Na24	Mg24
0	1.0000000000E+00	0.0000000000E+00
1	9.5484160391E-01	4.5158396090E-02
2	9.1172248856E-01	8.8277511442E-02
3	8.7055056330E-01	1.2944943670E-01
4	8.3123789614E-01	1.6876210386E-01
5	7.9370052598E-01	2.0629947402E-01
⋮	⋮	⋮
146	1.1748291368E-03	9.9882517086E-01
147	1.1217757373E-03	9.9887822426E-01
148	1.0711181442E-03	9.9892888186E-01
149	1.0227481668E-03	9.9897725183E-01
150	9.7656250000E-04	9.9902343750E-01

Table 3.2: Output for Na24 decay after 150h

The code yielded an answer that was accurate to ten decimals.

3.9.3 Problem 3: Long-lived parent decay to short-lived daughter

Solution method: Matrix exponential method for the parent and Bateman for the daughter.

Estimate the concentration of Mg28, Al28 and Si28 for ^{28}Mg $\xrightarrow{\beta-}$ ^{28}Al $\xrightarrow{\beta-}$ ^{28}Si after a time interval of 3h.

Initial Mg28 concentration = 1.0 g.atom

$T_{\frac{1}{2}}$ of Mg28 = 75276s

$T_{\frac{1}{2}}$ of Al28 = 134.43s

and Si28 is stable.

Answer:

Analytical results

Mg28 = $N_{Mg28}(0)e^{-\lambda_{Mg28}t}$ = 9.90533772491E-01 g.atom

Al28 = $\frac{\lambda_{Mg28}}{\lambda_{Al28}-\lambda_{Mg28}}N_{Mg28}(0)\left[e^{-\lambda_{Mg28}t}-e^{\lambda_{Al28}t}\right]$ = 1.16196700490E-03 g.atom

Si28 = 9.3042605043E-02 g.atom (balance)

Solution from simulation: Mg28 is considered to be long-lived and Al28 short-lived hence the matrix exponential method was used by the code to solve for Mg28 and Bateman for finding Al28 and Si28. The code output is listed in Table 3.3 and Figure 3.3.

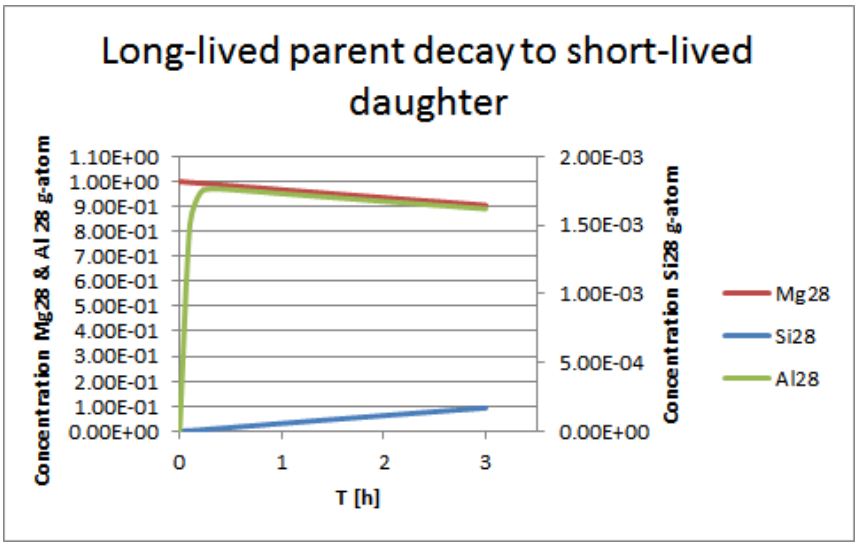

Figure 3.3: Mg28 decay to Al28 and Si28

T (h)	Na24	Mg24	Si24
0	1.0000000000E+00	0.0000000000E+00	0.0000000000E+00
1	9.6739433350E-01	1.7306907360E-03	3.0874975767E-02
2	9.3585179648E-01	1.6742604260E-03	6.2473943093E-02
3	9.0533772491E-01	1.6196700489E-03	9.3042605043E-02

Table 3.3: Output for Mg28 decay after 3h

The code results are in agreement with the analytical results to the tenth decimal.

3.9.4 Problem 4: Short-lived parent decay to long-lived daughter

Solution method: Bateman for the parent and matrix exponential method for the daughter.
Estimate the concentration of La144 and Ce144 for $^{144}\text{La} \xrightarrow{\beta-} {}^{144}\text{Ce}$ after time interval of 30min.
Initial La144 concentration = 1.0 g.atom
$T_{\frac{1}{2}}$ of La144 = 40.9s
$T_{\frac{1}{2}}$ of Ce144 = 24615360s
Answer:
Analytical results
La144 = $N_{La144}(0)e^{-\lambda_{La144}t}$ = 5.6459383567E-14 g.atom
Ce144 = $\frac{\lambda_{La144}}{\lambda_{Ce144}-\lambda_{La144}} N_{La144}(0)\left[e^{-\lambda_{La144}t} - e^{\lambda_{Ce144}t}\right]$ = 9.9995097633E-01 g.atom
Solution from simulation: La144 is considered to be short-lived and Ce144 long-lived hence the Bateman solution was used by the code to solve for La144 and the matrix exponential method for finding Ce144. The code output and graph is listed in Table 3.4 and Figure 3.4.

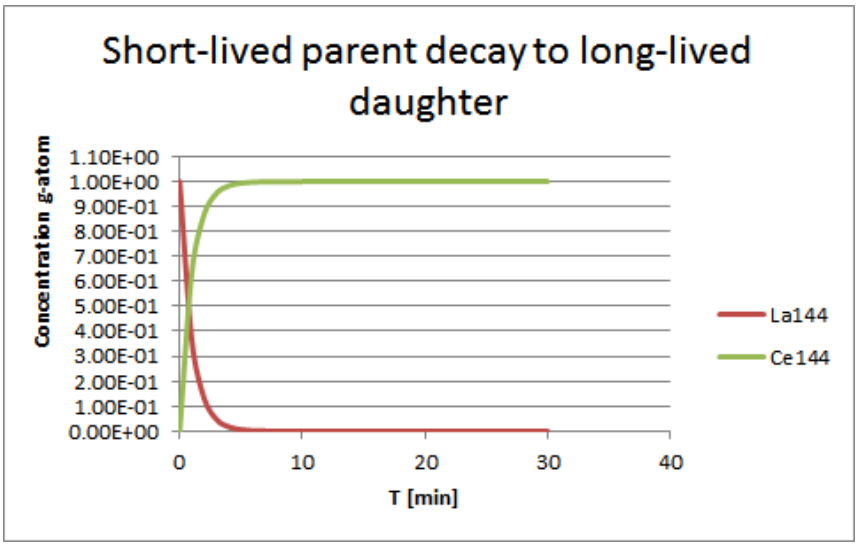

Figure 3.4: La144 decay to Ce144

T (min)	La144	Ce144
0	1.0000000000E+00	0.0000000000E+00
10	3.8362954264E-05	9.9994640318E-01
20	1.4717162599E-09	9.9996786965E-01
30	5.6459383568E-14	9.9995097633E-01

Table 3.4: Output for La144 decay after 30 min

The code results are in agreement with the analytical results to the tenth decimal.

Chapter 4

Results of simulation of isotope depletion

The output of the simulation code for this work was compared to that of an ORIGEN output for a reference equilibrium fuel cycle (Bell, 1973). The reference reactor initial fuel load was of 1 metric ton of 3.3% enriched uranium fuel. The output of the reactor was 30MW and was kept at that level by increasing the neutron flux as the U235 concentration decreased. It ran for 1100 days to give a burnup of 33000 MWd/t. The output lists the nuclide concentrations for 515 isotopes every 110 days. Stacey (2004) also give some actinide concentrations for 3.13% enriched fuel with a burnup of 32000 MWd/t that was also calculated with ORIGEN. A comparison of the two is shown below in Table 4.1 in gram-atoms.

Isotope	Bell	Stacey
U234	6.08E-01	5.19E-01
U235	2.98E+01	3.40E+01
U236	1.35E+01	1.93E+01
U237	0.00E+00	4.47E-02
U238	3.29E+03	3.96E+03
Np237	1.57E+00	1.99E+00
Np239	1.94E-01	3.33E-01
Pu238	5.21E-01	6.78E-01
Pu239	1.91E+01	2.17E+01
Pu240	6.29E+00	9.02E+00
Pu241	5.34E+00	4.29E+00
Pu242	1.63E+00	1.46E+00
Am241	2.25E-01	1.04E-01
Am243	3.29E-01	3.88E-01
Cm242	5.76E-02	4.17E-02

Table 4.1: Comparison of ORIGEN outputs

The two are in reasonable agreement with each other. The differences may be due to dissimilarities between the initial fuel compositions.

4.1 Simulation output

To compare the simulation output of this work with ORIGEN (Bell, 1973), the same isotopes were used. A step size of 3600 s was used and the simulation was ran for $1100 \times 24 = 26400$ iterations. This is equivalent to operating just over three years and results in fuel burnup of 33000 MWd/t. A comparison of the two is shown below in gram-atoms with the ratio of the simulation results to that of Bell.

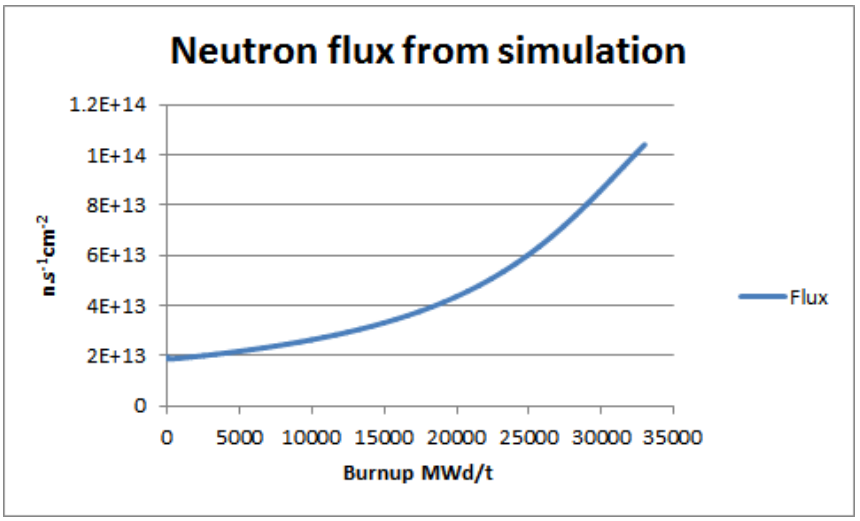

Figure 4.1: Neutron flux from simulation

Isotope	ORIGEN	Simulation
U234	5.19E-01	7.26E-01
U235	3.40E+01	7.97E+00
U236	1.93E+01	1.88E+01
U237	4.47E-02	8.00E-03
U238	3.96E+03	3.99E+03
Np237	1.99E+00	2.09E-01
Np239	3.33E-01	3.30E-01
Pu238	6.78E-01	3.95E-02
Pu239	2.17E+01	1.05E+01
Pu240	9.02E+00	5.99E+00
Pu241	4.29E+00	1.04E+00
Pu242	1.46E+00	6.25E-01
Am241	1.04E-01	1.80E-02
Am243	3.88E-01	1.28E-02
Cm242	4.17E-02	8.90E-03
Cm244	1.24E-01	8.42E-04

Table 4.2: Comparison of ORIGEN output vs simulation

The simulation results produced by the code did not give a good agreement with that of ORIGEN. For example, the simulation gives the U235 concentration as 7.79 g.atom compared to ORIGEN's value of 34. This is most likely due to the simulation output having less fissionable actinides like plutonium. Since in PWR's plutonium fissioning contribute a significant fraction of the power output with increasing burnup, the lower concentration of especially Pu239, means that in the simulation the flux have to be increased so that the diminishing amount of U235 can still maintain the power output. As can be seen in Figure 4.1, the flux increases exponentially, which would explain the lower U235 concentration.

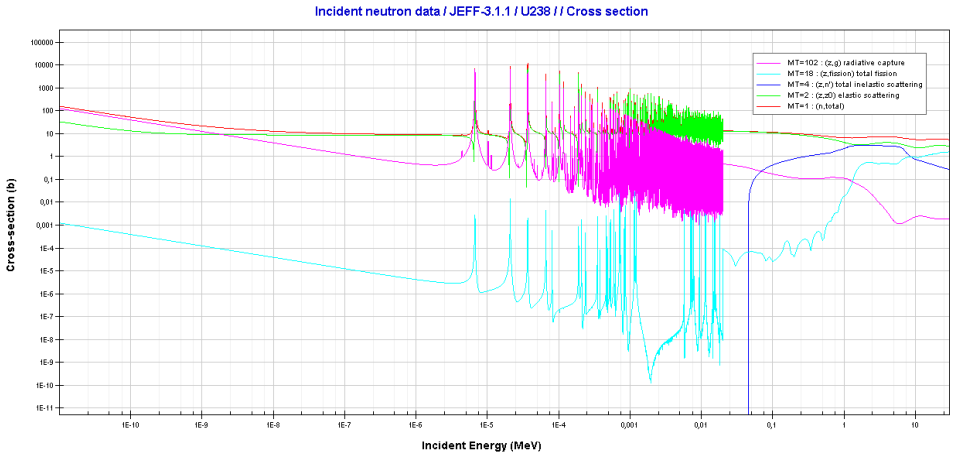

Figure 4.2: U238 neutron cross sections

Pu239 production begins with neutron capture by U238. The reactions are

$$U238 \xrightarrow[2.72b]{(n,\gamma)} U239 \xrightarrow[24mi]{\beta-} Np239 \xrightarrow[2.3day]{\beta-} Pu239 \xrightarrow[270b]{(n,\gamma)} Pu240 \tag{4.1}$$

The thermal neutron capture cross section for U238 used in the simulation is 2.72b, however this appears not to be sufficiently large enough to produce enough plutonium.

4.2 Effects of resonance escape factor on U238

One reason for lower depletion of U238 in the simulation compared to the ORIGEN results is that the simulation only takes thermal neutrons into account. U238 has large resonances that can capture epi-thermal neutrons as can be seen in Figure 4.2 using JEFF-3.1.1 evaluations (Nuclear Power for Everybody, 2017).

The probability of a neutron not being adsorbed by the U238 resonances while being moderated is called the resonance escape probability, p. To consider the effect of the resonance escape probability on the depletion of U238, we can derive an equation to calculate its depletion. If the probability that a neutron is not adsorbed by resonance is p, the resonance escape probability, then the amount of neutrons adsorbed is $\frac{1-p}{p}\phi$, where ϕ is the flux of neutrons that have been thermalised. Then we can write the rate of change in N_{238}, the concentration of U238, (ignoring radioactive decay and assuming constant flux), as

$$\frac{dN_{238}}{dt} = -\phi\sigma_a N_{238} - \frac{1-p}{p}\phi$$

$$\text{Set } k = \frac{1-p}{p}$$

$$\frac{dN_{238}}{dt} + \phi\sigma_a N_{238} = -k\phi$$

$$\text{Multiply both sides by integration factor } = e^{\int \phi\sigma_a dt} = e^{\phi\sigma_a t}$$

$$e^{\phi\sigma_a t}(\frac{dN_{238}}{dt} + \phi\sigma_a N_{238}) = -k\phi e^{\phi\sigma_a t}$$

but from the product and chain rule of differentiation

$$\frac{d(N_{238}e^{\phi\sigma_a t})}{dt} = e^{\phi\sigma_a t}(\frac{dN_{238}}{dt} + \phi\sigma_a N_{238})$$
$$\therefore \frac{dN_{238}e^{\phi\sigma_a t}}{dt} = -k\phi e^{\phi\sigma_a t}$$

Integrate both sides with respect to t

$$\int \frac{dN_{238}e^{\phi\sigma_a t}}{dt}dt = \int -k\phi e^{\phi\sigma_a t}dt$$
$$N_{238}e^{\phi\sigma_a t} = -k\phi \int e^{\phi\sigma_a t}dt + C$$
$$N_{238}e^{\phi\sigma_a t} = -\frac{k}{\sigma_a}e^{\phi\sigma_a t} + C$$
$$N_{238} = -\frac{k}{\sigma_a} + Ce^{-\phi\sigma_a t}$$

If $t = 0$, then $N_{238} = N_{238}(0)$

$$\therefore C = N_{238}(0) + \frac{k}{\sigma_a}$$
$$\text{but } k = \frac{1-p}{p}$$
$$\therefore N_{238}(t) = \left[N_{238}(0) + \frac{1-p}{\sigma_a p}\right]e^{-\phi\sigma_a t} - \frac{1-p}{\sigma_a p}$$

Take the average flux as $2.92 \times 10^{13} ns^{-1}cm^{-2}$ from the ORIGEN reference results (Bell, 1973), the resonance escape possibility, p, as typically 0.87 for a thermal reactor (Bodansky, 1996), the thermal neutron absorption cross section for U2389 as $2.72b$ from ENDF/VI.8 (OECD, 2017) and $N_{238}(0)$ as $4060 \times 6.022 \times 10^{23}\, atoms$ as is used in the simulation. Then if we calculate the U238 concentration after 1100 days, we find that

With resonance absorption

$$N_{238}(1100d) = 4029.46796\, g.atoms \tag{4.2}$$

and without

$$N_{238}(1100d) = 4029.46864\, g.atoms \tag{4.3}$$

a difference of 0.000017%. Therefore resonance absorption alone cannot account for the discrepancy. The fact that this simulation only takes thermal neutrons into account, in stead of the full energy spectrum, is also a factor.

4.3 Increasing U238 capture cross section

To test the theory that the simulation effective cross section of U238 is too small to account for Pu production, more simulations were run with the capture cross section for U238 set at 5 b and 10 b respectively.

Isotope	ORIGEN	$\sigma_c^{U238} = 5$b	$\sigma_c^{U238} = 10$b
U234	5.19E-01	8.11E-01	8.83E-01
U235	3.40E+01	1.64E+01	2.92E+01
U236	1.93E+01	1.77E+01	1.59E+01
U237	4.47E-02	4.20E-03	2.12E-03
U238	3.96E+03	4.01E+03	4.01E+03
Np237	1.99E+00	1.50E-01	9.94E-02
Np239	3.33E-01	3.32E-01	3.68E-01
Pu238	6.78E-01	2.62E-02	1.68E-02
Pu239	2.17E+01	1.88E+01	3.55E+01
Pu240	9.02E+00	8.42E+00	1.19E+01
Pu241	4.29E+00	1.31E+00	1.57E+00
Pu242	1.46E+00	5.34E-01	4.32E-01
Am241	1.04E-01	3.06E-02	4.68E-02
Am243	3.88E-01	7.83E-03	4.45E-03
Cm242	4.17E-02	8.84E-03	8.01E-03

Table 4.3: Comparison of actinide concentrations for different U238 capture cross sections at 33000 MWd/t burnup

The results show that increasing the capture cross section of U238, the results for plutonium concentration more closely match those of ORIGEN. The increased plutonium also moderated the flux compared to the original result. As can be seen in in Figure 4.3, increasing the capture cross section for U238 to 10b results in a more realistic flux.

4.4 Definition of spectral indices used in ORIGEN

A major difference between ORIGEN and this depletion code is that the code uses only thermal neutron cross sections, whereas ORIGEN uses an 'effective' cross section which is calculated by weighting the three energy group cross sections from one of the ORIGEN standard libraries with spectral indices for the respective system as shown in Appendix A.

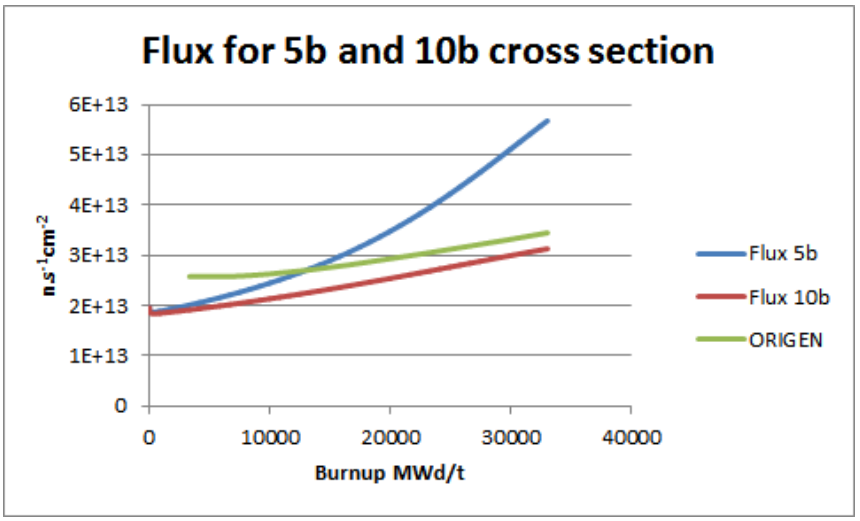

Figure 4.3: Neutron flux from simulation for $\sigma_c^{U238} = 5$b and $\sigma_c^{U238} = 10$b

4.5 Fission product buildup

The buildup of some important fission products as predicted by the simulation is compared with that of ORIGEN.

4.5.1 Strontium-90

The first is Strontium-90. It is formed as follows in the reactor:

$$^{235}U(n,f)^{90}Kr \xrightarrow{\beta-} {}^{90}Rb \xrightarrow{\beta-} {}^{90}Sr \tag{4.4}$$

where Kr90 has a yield of 0.0439695 atoms per fission. Sr90 is also formed directly as a fission product with a yield of 0.000737128 atoms per fission of U235.

$$^{235}U(n,f)^{90}Sr \tag{4.5}$$

Sr90 has a half life of 28 years so it will build up in the reactor.

There is a good correspondence between the simulation and ORIGEN for Sr90 buildup as can be seen in Figure 4.4.

4.5.2 Caesium-137

Cs137 is formed as follows in the reactor:

$$^{235}U(n,f)^{137}I \xrightarrow{\beta-} {}^{137}Xe \xrightarrow{\beta-} {}^{137}Cs \tag{4.6}$$

$$^{235}U(n,f)^{137}Cs \tag{4.7}$$

$$^{136}Cs(n,\gamma)^{137}Cs \tag{4.8}$$

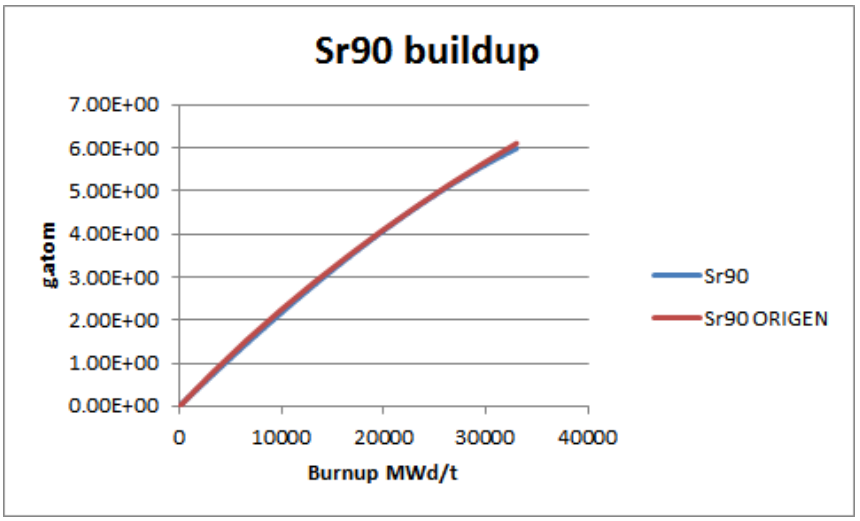

Figure 4.4: Sr90 buildup

where I137 has a yield of 0.0262236 atoms per U235 fission. Cs137 is also formed as a direct fission product from U235 with a yield of 0.000599988 atoms per fission, as well as from the other fissionable isotopes. It is also formed from Cs136 by neutron absorption with a thermal neutron cross section of 1.3b.

Cs137 has a half-life of 30 years so it will build up in the reactor.

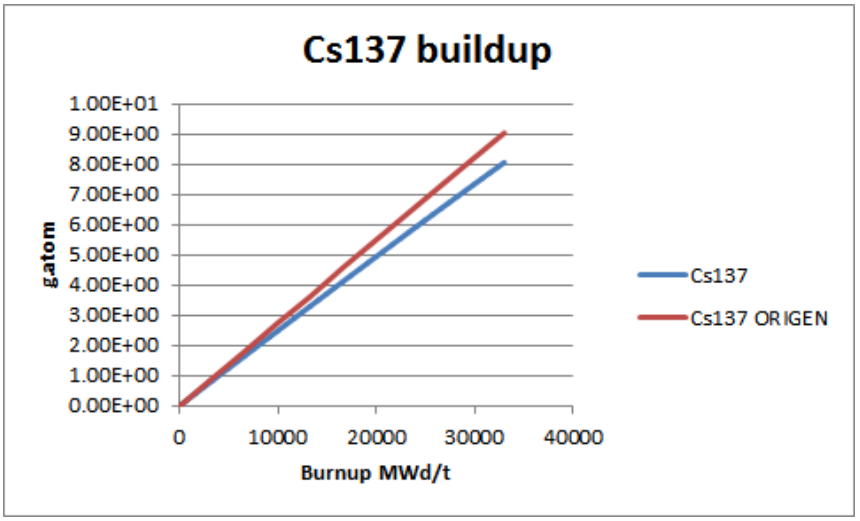

Figure 4.5: Cs137 buildup

There is a reasonable correspondence between the simulation and that of ORIGEN's results as can be seen in 4.5. The difference in results are most likely due to the difference in flux and fissile product yield between the simulation and ORIGEN and that the simulation only takes thermal neutrons into consideration.

4.5.3 Xenon-135

Xe135 is a reactor poison with a large neutron absorption cross section. It builds up in a reactor to an equilibrium value. When the reactor is shut down, the Xe135 concentration increases due to I135 decay to a maximum value and then decreases due to its own radioactive decay. From Stacey (2004) the formula for the equilibrium concentration is

$$\begin{aligned} X_{eq} &= \frac{(\gamma^{Te} + \gamma^{I} + \gamma^{Xe}) \sum_f \Phi}{\lambda^{Xe} + \sigma_a^{Xe} \Phi} \\ &= \frac{(0.03216 + 0.02927 + 0.000785)(140)(584.9E-24)(1.87E+13)}{2.107E-05 + (2670933E-24)(1.87E13))} \\ &= 1.34E-03 \text{ g.atom} \end{aligned}$$

where γ is the fraction of the isotope produced from fission and the flux values and U235 concentration are from the simulation.

The time constant is $1/(\lambda^X + \sigma_a^X \Phi) \approx 30h$. The simulation was run for an initial startup to 50 h. Then a shutdown and another 100 h with no flux was simulated. The resultant Xe135 concentration is plotted in Figure 4.6. As can be seen, the Xe135 approaches the equilibrium value of 1.34E-03 g.atom after 50 h and then increases to a maximum 11 h after the reactor shutdown. The simulation output corresponds well with that of Stacey (2004).

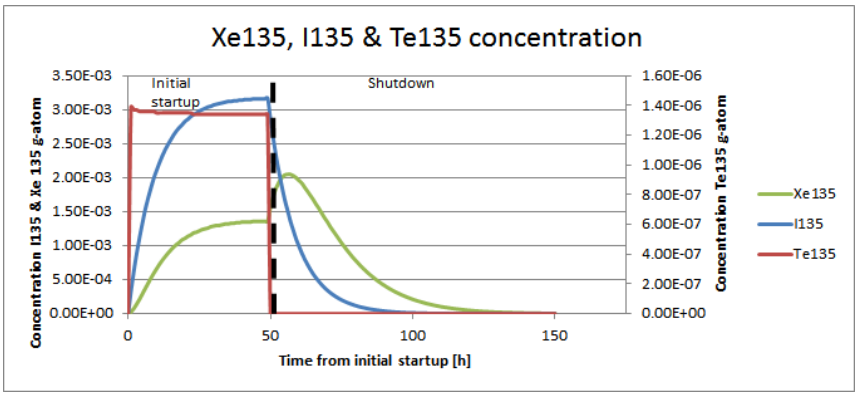

Figure 4.6: Xe135 buildup and decay

4.5.4 Samarium-149

Another reactor poison is Samarium-149. It also has a large cross section for thermal neutrons like Xe135, but it is not radioactive, so its concentration reaches an equilibrium after a few hundred hours during typical reactor operation. Sm149 is formed from beta decay of Pm149, which is formed by beat decay of the fission product Nd149.

$$^{235}U(n,f)^{149}Nd \xrightarrow{\beta-} {}^{149}Pm \xrightarrow{\beta-} {}^{149}Sm(n,\gamma)^{150}Sm \tag{4.9}$$

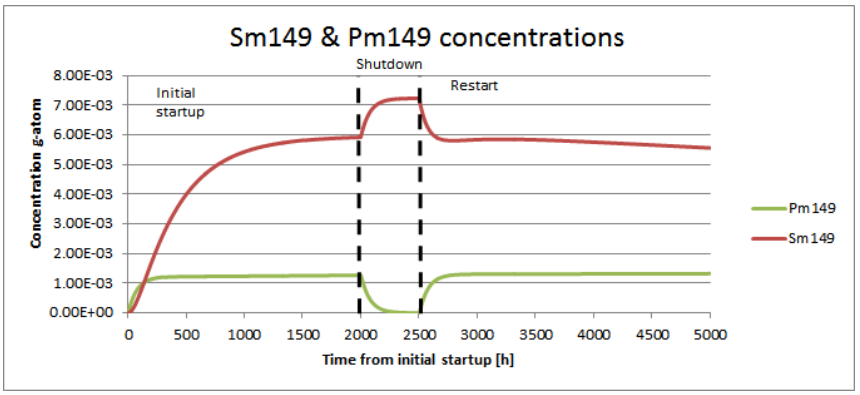

Figure 4.7: Sm149 buildup and decay

When the reactor is shutdown, the Samarium-149 concentration builds up as a result of the decay of the accumulated Promethium-149. If the reactor is started again, the Samarium-149 is burned up and returns to its equilibrium level. This scenario was simulated and the results are shown in Figure 4.7. The half life for Nd149 is 1.73h, so it's concentration is relatively low for the graph time domain and is omitted. The results are qualitatively the same as Stacey (2004), but not quantitatively, as the simulation flux is lower than that used by Stacey's example. The gradual decrease of the Sm149 concentration after the restart is due to the increase in flux that is required to keep the reactor power output at the required level with depleting fuel.

4.6 Decay heat

Decay heat can be simulated by setting the flux to zero and executing the code. The isotope inventory of the reference reactor after burnup of 33GWd/t was used to calculate the decay heat for up to 1000 days. A simulation was also done for the first hour after shutdown. The decay heat in the simulation was calculated as follows:

$$Q_{rd}(t) = \sum_i [N_i(t_0)(1 - e^{-\lambda_i \Delta t})E_i] \tag{4.10}$$

where $Q_{rd}(\Delta t)$ is the decay heat released in the time step of $\Delta t = t - t_0$, E_i is the total radioactive decay energy released for a nuclide i decay (sum of α, β and γ components), λ_i is the radioactive decay constant and $N_i(t_0)$ is the isotopic concentration for nuclide i at the start of the time step. The decay power is then calculated by dividing the decay heat by the time step

$$P_{rd}(t) = \frac{Q_{rd}(t)}{\Delta t} \tag{4.11}$$

Empirical formulas used for decay heat estimation (Garland, 1998) takes only beta and gamma decay into account. If P is the decay heat and P_0 is the nominal reactor power, then

$$\frac{P}{P_0} = 0.066 \left[{\tau_{elapsed}}^{-0.2} - (\tau_s + \tau_{elapsed})^{-0.2} \right] \tag{4.12}$$

where τ is the time since reactor start-up and τ_s is the time of reactor shutdown measured from the time of startup. To use the expression with a shutdown as the time reference point we define $\tau_{elapsed} = \tau - \tau_s$.

Another method of estimating decay heat is using the Rhode Island Safety Analysis Report - Part B, Thermal Hydraulic Analysis [RI-SAR]. It refers to the American Nuclear Society reference curve of 1968 and presents a tabulated list of values. It is known as the ANS5.1 / N18.6 standard. The following approximation can be used for the curve.

$$\frac{P}{P_0} = 0.005a \left[{\tau_{elapsed}}^{-b} - (\tau_s + \tau_{elapsed})^{-b} \right] \tag{4.13}$$

where a and b are constants which depend on τ_s as follows:

T after shutdown (s)	a	b
0.1-10	12.05	0.0639
10 to 150	15.31	0.1807
150 to 8E+08	27.48	0.2926

The formula is said to fit the ANS curve to within ±6%. The ANS standard is applicable as a general estimate of the decay heat. Errors up to 50% can result for short (<1000 seconds) and long (>10^7 seconds) times (Todreas, 1990). In the mid-range, errors are of the order of +10% and -20%.

The decay heat was calculated by the simulation for a fuel with 33 GWd/t burnup and 30 MW nominal power and plotted with equations 4.12 and 4.13 in Figure 4.8.

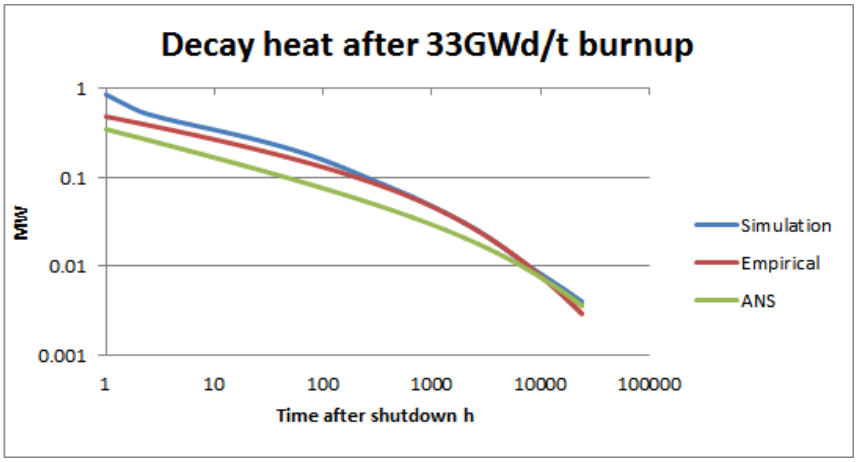

Figure 4.8: Decay heat calculation

The simulation gives a higher decay heat value than Equation 4.12 until 500 h after shutdown, from where it conforms nicely. It also gives a higher value than the ANS prediction until 10000 h after shutdown. This is most likely due to an increase of radioactive actinides that also have a neutron absorption cross section after shutdown. For example, the decay of actinides U239 and Am243 to Np239.

Chapter 5

Conclusions

The simulation was set up to solve isotopic concentrations for fission and decay reactions. Once all the data was set up, the simulation was easy and flexible to use. To model a specific reaction like those used for the code validation, only the initial nuclide concentrations and required duration need to be set. The simulation will read all the relevant data from its libraries which contain data for over 1700 isotopes and isomers and write the output. The methods employed to solve the isotope concentrations, namely matrix exponentiation and the Bateman formulas, were also quite fast. Even having to deal with a 515 by 515 matrix, each complete step of the simulation only took about one second to complete, meaning three years of operation can be simulated in hourly increments with about eight hours of computing time on a basic desktop computer. The method of solving the simultaneous differential equations for the isotope concentrations at each time step also has the benefit of producing data that is easily graphed. If new isotope data becomes available, then the simulation can load it in ENDF format.

The simulation modelled decay reactions very well, but was less successful with neutron interaction induced reactions when compared to ORIGEN results. This was due to only thermal neutrons being considered, which underestimates the resonance absorption of epithermal neutron by especially U238. Further development of the simulation could focus on the use of spectral indices as in Appendix A.

The simulation as is can be used for decay chain and decay heat analysis. It can handle multiple decay chains as well as isotopes with widely differing decay constants. The simulation can also do order of magnitude estimations of fission products and isotope concentrations of a thermal neutron fission reactor.

Appendices

Appendix A

Definition of spectral indices used in ORIGEN

In the ORIGEN computer code effective one-group cross sections are calculated by weighting the three energy group cross sections from one of the ORIGEN standard libraries with spectral indices for the respective system (Salahuddin and Arshad, 1987).

The effective one-group cross section in ORIGEN is defined as:

$$\sigma^{eff} = \sigma_{th} + RI \times RES + \sigma_F \times FAST \tag{A.1}$$

where

σ^{eff} = Total 2200m/sec neutron cross section

RI = Resonance integral for all epithermal neutrons

$= \int_{0.5Mev}^{1MeV} \frac{\sigma(E)}{E} de$

σ_F = Fission spectrum (FS) averaged cross-section for all reactions with a threshold above 1 MeV

$= \frac{\int_{1Mev}^{\infty} \sigma(E)\Phi_{FS}(E)dE}{\int_{1Mev}^{\infty} \Phi_{FS}(E)dE}$

The three spectral indices THERM, RES and FAST for thermal, resonance and fast energy systems are defined as follows:

THERM: The ratio of the neutron reaction rate for a $\frac{1}{V}$ absorber with a population of neutrons that has a Maxwell-Boltzmann distribution of energies at absolute temperature T, to the reaction rate with 2200 m/sec neutrons

$= \sqrt{\frac{\pi}{4}\frac{T_o}{T}}$

where

$T_o = 293.16°\text{K}$

$T =$ Effective moderator temperature in °K

In the definition of THERM it has been assumed that for normal neutron energies, the cross section of an isotope varies with the reciprocal of the neutron speed.

RES: The ratio of the resonance flux per unit lethargy to the thermal neutron flux

$= \frac{\frac{\Phi_{res}}{\Delta U}}{\Phi_{th}}$

where

$\Phi_{res} = \int_{0.5eV}^{1Mev} \Phi(E)dE$

$\Delta U = \int_{0.5eV}^{1Mev} \frac{dE}{E}$

$\Phi_{th} = \int_{0}^{0.5eV} \Phi(E)dE$

In the definition of RES it has been assumed that for the resonance region, the neutron flux varies varies as the reciprocal of the neutron energy.

FAST: 1.45 times the ration of flux above 1 MeV to the thermal neutron flux

$= 1.45 \frac{\Phi_{fast}}{\Phi_{th}}$

where

$\Phi_{fast} = \int_{1MeV}^{\infty} \Phi(E)dE$

$1.45 = \frac{\int_{0}^{\infty} \Phi_{FS}(E)dE}{\int_{1MeV}^{\infty} \Phi_{FS}(E)dE}$

In the definition of FAST it has been assumed that the neutron spectrum has the same energy dependence as the fission spectrum (FS).

Appendix B

Python code Read_ENDF_data

```
# Read ENDF VI data and write to a new file
# To be used on files that went through PREPRO routine
# including SIGMA for broadend cross-sections

import os, fileinput, math
import numpy as np

# Set up variables
##MF Description
##1 General information
##2 Resonance parameter data
##3 Reaction cross sections
##4 Angular distributions for emitted particles
##5 Energy distributions for emitted particles
##6 Energy-angle distributions for emitted particles
##7 Thermal neutron scattering law data
##8 Radioactivity and fission-product yield data
##9 Multiplicities for radioactive nuclide production
##10 Cross sections for radioactive nuclide production
##12 Multiplicities for photon production
##13 Cross sections for photon production
##14 Angular distributions for photon production
##15 Energy distributions for photon production
##23 Photo- or electro-atomic interaction cross sections
##26 Electro-atomic angle and energy distribution
##27 Atomic form factors or scattering functions for photo-atomic
    interactions
##28 Atomic relaxation data
##30 Data covariances obtained from parameter covariances and sensitivities
##31 Data covariances for nu(bar)
##32 Data covariances for resonance parameters
##33 Data covariances for reaction cross sections
##34 Data covariances for angular distributions
##35 Data covariances for energy distributions
##39 Data covariances for radionuclide production yields
##40 Data covariances for radionuclide production cross sections
```

```
MF="3"
##NLIB Library Definition
##0 ENDF/B - United States Evaluated Nuclear Data File
##1 ENDF/A - United States Evaluated Nuclear Data File
##2 JEFF - NEA Joint Evaluated Fission and Fusion File (formerly
##JEF)
##3 EFF - European Fusion File (now part of JEFF)
##4 ENDF/B High Energy File
##5 CENDL - China Evaluated Nuclear Data Library
##6 JENDL - Japan Evaluated Nuclear Data Library
##21 SG-23 - Fission product library of theWorking Party on Evaluation
##Cooperation Subgroup-23 (WPEG-SG23)
##31 INDL/V - IAEA Evaluated Neutron Data Library
##32 INDL/A - IAEA Nuclear Data Activation Library
##33 FENDL - IAEA Fusion Evaluated Nuclear Data Library
##34 IRDF - IAEA International Reactor Dosimetry File
##35 BROND - Russian Evaluated Nuclear Data File (IAEA version)
##36 INGDB-90 - Geophysics Data
##37 FENDL/A - FENDL activation evaluations
##41 BROND - Russian Evaluated Nuclear Data File (original version)
NLIB="0"

##NSUB IPART ITYPE Sub-library Names
##0 0 0 Photo-Nuclear Data
##1 0 1 Photo-Induced Fission Product Yields
##3 0 3 Photo-Atomic Interaction Data
##4 0 4 Radioactive Decay Data
##5 0 5 Spontaneous Fission Product Yields
##6 0 6 Atomic Relaxation Data
##10 1 0 Incident-Neutron Data
##11 1 1 Neutron-Induced Fission Product Yields
##12 1 2 Thermal Neutron Scattering Data
##113 11 3 Electro-Atomic Interaction Data
##10010 1001 0 Incident-Proton Data
##10011 1001 1 Proton-Induced Fission Product Yields
##10020 1002 0 Incident-Deuteron Data
##10030 1003 0 Incident-Triton Data
##20030 2003 0 Incident-Helion (3He) Data
##20040 2004 0 Incident-Alpha data
NSUB="10"
IPART="1"
ITYPE="0"

# Material number MAT
# U235
MAT="9228"

# ZA = 1000.0 x Z + A,
ZA="92238"

# Reaction type number MT
##MT Meaning Description
```

```
# 16 (n,2n)
# 17 (n,3n)
##18 (z,xf) total prompt fission
##19 (z,f) first chance fission
##20 (z,nf) second chance fission
##21 (z,2nf) third chance fission
##38 (z,3nf) fourth chance fission
##452 vT total number of neutrons per fission
##455 vd number of delayed neutrons per fission
##456 vp number of prompt neutrons per fission
##458 components of energy release in fission
# 101 (n,adsorption)
# 103 (n,p)
# 107 (n,alpha)
MT="107"

##x(n) is the nth value of x,
##y(n) is the nth value of y,
##NP is the number of pairs (x and y) given,
##INT(m) is the interpolation scheme identification number used in the mth
    range,
##NBT(m) is the value of n separating the mth and the (m + 1)th
    interpolation ranges.
##The list of allowed interpolation schemes is given in Table 16.
##Table 16: Definition of Interpolation Types
##INT Interpolation Scheme
##1 y is constant in x (constant, histogram)
##2 y is linear in x (linear-linear)
##3 y is linear in ln(x) (linear-log)
##4 ln(y) is linear in x (log-linear)
##5 ln(y) is linear in ln(x) (log-log)
##6 special one-dimensional interpolation law, used for charged-particle
    cross sections
##only
##11-15 method of corresponding points (follow interpolation laws of 1-5)
##21-25 unit base interpolation (follow interpolation laws of 1-5)
##Interpolation code, INT=1 (constant), implies that the function is
    constant and equal
##to the value given at the lower limit of the interval.

# Broadenend temperrture
t_br=600

# Log file name
# Path to pre-processed ENDF files
filepath = os.path.join\
           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/ENDF/IBMPc/
    PREPRO15/',\
            'SIGMA1-'+str(t_br)+'K.OUT')
# Path to output file
outputfilepath = os.path.join\
           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/Datafiles/',\
```

```
135             MF+MT.rjust(3,'_')+'_'+str(t_br)+'K.dat')
136
137 # Extract a number from a string
138 def extract_nbr(input_str):
139     num_char = (".", "+", "-","E")
140
141     l = ""
142     num=""
143     tokens = input_str.split()
144     for token in tokens:
145
146         for char in token:
147             if char.isdigit() or (char in num_char):
148                 num = num + char
149
150         try:
151             l+=str((float(num)))
152         except ValueError:
153             pass
154     if l=="":
155         return(0)
156     else:
157         return(float(l))
158
159 # print("Extracted no :",extract_nbr("1.00000E-5"))
160 #Start main loop
161 print("Start reading file")
162 # Find the file segment we're looking for
163
164 fseqn=MF.rjust(2)
165 fseqn=fseqn+(MT.rjust(3))+"    1"
166 ZA="1001"
167 # Energy in eV
168 E0=0.0253
169
170 print("Finding : ",fseqn," for ZA = all and T=",t_br,"K")
171
172 with open(outputfilepath, 'wb') as outputfile:
173     lines=iter(fileinput.input(files=(filepath)))
174     for line in lines:
175     # Read MF MT NS
176         mf_l=line[70:80]
177         if mf_l == fseqn:
178             za_l=line[0:11]
179             line_no=lines.lineno()
180     #         if str(int(extract_nbr(za_l)))==ZA:
181             ZA=str(int(extract_nbr(za_l)))
182             print("line number : ",line_no," ZA = ",za_l)
183             line=next(lines)
184             QI=str(extract_nbr(line[11:22]))
185             line=next(lines)
186     # We assume the data has already been linearly interpolated
```

```
187     # get number of data pairs
188             n_prs=int(extract_nbr(line[0:11]))
189             n_lines=math.trunc(n_prs/3)
190             n_last=n_prs-(n_lines*3)
191     # Read the interpolated data into np array
192             crssec = np.zeros((n_prs,2),dtype=np.float64)
193             for i in range(0,n_lines*3-1,3):
194                 line=next(lines)
195                 k=0
196                 for j in range(0,6,2):
197                     crssec[i+k,0]=extract_nbr(line[11*(j):11*(j+1)])
198                     crssec[i+k,1]=extract_nbr(line[11*(j+1):11*(j+2)])
199                     k+=1
200     # Read last line, can be 0,1 or 2 pairs
201             if n_last==0:
202                 pass
203             else:
204                 if n_last==1:
205                     line=next(lines)
206                     k=3
207                     j=0
208                     crssec[i+k,0]=extract_nbr(line[11*(j):11*(j+1)])
209                     crssec[i+k,1]=extract_nbr(line[11*(j+1):11*(j+2)])
210                 else:
211                     line=next(lines)
212                     k=3
213                     for j in range(0,4,2):
214                         crssec[i+k,0]=extract_nbr(line[11*(j):11*(j+1)])
215                         crssec[i+k,1]=extract_nbr(line[11*(j+1):11*(j+2)])
216                         k+=1
217 
218     # Find the corresponding value for E0
219             crssE0pos=np.searchsorted(crssec[:,0],E0)
220             if crssE0pos==0 or crssE0pos==len(crssec):
221                 print(E0," not in array ")
222                 crssE0=0
223             if crssec[crssE0pos,0]==E0:
224                 crssE0=crssec[crssE0pos,1]
225             else:
226     # Interpolate
227                 crssE0=(E0-crssec[crssE0pos-1,0])/\
228                         (crssec[crssE0pos,0]-crssec[crssE0pos-1,0])\
229                         *(crssec[crssE0pos,1]-crssec[crssE0pos-1,1])+crssec[
    crssE0pos-1,1]
230     # Check if negative, means E0<smallest listed value
231             if crssE0<0:
232                 crssE0=0
233 # Ignore ZA where A=000 natural isotopes
234             if ZA[-3:] != '000':
235                 outputfile.write(bytes(ZA,encoding='utf-8'))
236                 outputfile.write(bytes('\t',encoding='utf-8'))
237                 outputfile.write(bytes(str(crssE0)+'\t',encoding='utf-8'))
```

```
238                     outputfile.write(bytes(QI+'\n',encoding='utf-8'))
239
240 print("Completed")
```

Appendix C

Python code
Read_ENDF_decay_data

```
# Read ENDF VI decay data and write to a new file

import os, fileinput, math
import numpy as np

##8 Radioactivity and fission-product yield data
MF="8"

##NLIB Library Definition
##0 ENDF/B - United States Evaluated Nuclear Data File
NLIB="0"

##NSUB IPART ITYPE Sub-library Names
##4 0 4 Radioactive Decay Data
NSUB="4"
IPART="0"
ITYPE="4"

# Material number MAT
# U235
MAT="9228"

# ZA = 1000.0     Z + A,
ZA="92238"

# Reaction type number MT
##MT Meaning Description
##457 Radioactive decay data

MT="457"

# Log file name
# Path to pre-processed ENDF files
```

```
filepath = os.path.join\
           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/ENDF/',\
            'endf-b-vi-8_decay.dat')
# Path to output file
outputfilepath = os.path.join\
           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/Datafiles/',\
            MF+MT.rjust(3,'_')+'.dat')

# Extract a number from a string
def extract_nbr(input_str):
    num_char = (".", "+", "-","E")

    l = ""
    num=""
    tokens = input_str.split()
    for token in tokens:

        for char in token:
            if char.isdigit() or (char in num_char):
                num = num + char

        try:
            l+=str((float(num)))
        except ValueError:
            pass
    if l=="":
        return(0)
    else:
        return(float(l))

#Start main loop
print("Start reading file")
# Find the file segment we're looking for

fseqn=MF.rjust(2)
fseqn=fseqn+(MT.rjust(3))+"    1"

# Energy in eV
E0=0.0253

print("Finding : ",fseqn," for ZA = all")

with open(outputfilepath, 'wb') as outputfile:
# Iterate over input file
    lines=iter(fileinput.input(files=(filepath)))
    for line in lines:
    # Read MF MT NS
        mf_l=line[70:80]
        if mf_l == fseqn:
            za_l=line[0:11]
            liso=int(extract_nbr(line[33:44])) # Isomeric state number
            line_no=lines.lineno()
```

```
86      #         if str(int(extract_nbr(za_l)))==ZA:
87              ZA=str(int(extract_nbr(za_l))+100000*liso)
88              print("line number : ",line_no," ZA = ",za_l)
89              line=next(lines)
90      # get half-life in sec
91              t1_2=extract_nbr(line[0:11])
92      # assume NC is always 3, so just skip next line
93              line=next(lines)
94              line=next(lines)
95      # get number of decay modes
96              n_dkm=int(extract_nbr(line[55:66]))
97              for i in range(0,n_dkm):
98                  line=next(lines)
99      # get reation type
100                 rtyp=float(extract_nbr(line[0:11]))
101     # get isomeric state flag for daughter isotope
102     # RFS=0.0 ground state, =1.0 first isomeric state etc
103                 rfs=int(extract_nbr(line[11:22]))
104     # q total decay energy available
105                 q=float(extract_nbr(line[22:33]))
106     # branching fraction
107                 br=float(extract_nbr(line[44:55]))
108 # Ignore ZA with A=000 natural isotopes
109                 if ZA[-3:] != '000':
110                     outputfile.write(bytes(ZA+'\t'+str(t1_2)+'\t',encoding='
    utf-8'))
111                     outputfile.write(bytes(str(rtyp)+'\t'+str(q)+'\t',
    encoding='utf-8'))
112                     outputfile.write(bytes(str(br)+'\t',encoding='utf-8'))
113                     outputfile.write(bytes(str(rfs)+'\t',encoding='utf-8'))
114         # Determine daughter products
115                     za_p=extract_nbr(za_l)
116                     z_d=math.trunc(za_p/1000)
117                     a_d=int(za_p)-z_d*1000
118         # Get new ZA from decay type
119         # Beta minus
120                     if rtyp==1:
121                         z_d+=1
122                         outputfile.write(bytes(str(rfs*100000+z_d*1000+a_d)+
    '\n',encoding='utf-8'))
123         # Beta minus alpha
124                     elif rtyp==1.4:
125                         z_d-=1
126                         a_d-=4
127                         outputfile.write(bytes(str(rfs*100000+z_d*1000+a_d)+
    '\n',encoding='utf-8'))
128         # Beta minus delayed neutron
129                     elif rtyp==1.5:
130                         z_d+=1
131                         a_d-=1
132                         outputfile.write(bytes(str(rfs*100000+z_d*1000+a_d)+
    '\n',encoding='utf-8'))
```

```
        # EC or beta +
                    elif rtyp==2:
                        z_d-=1
                        outputfile.write(bytes(str(rfs*100000+z_d*1000+a_d)+
    '\n',encoding='utf-8'))
        # Isomeric transition
                    elif rtyp==3:
                        outputfile.write(bytes(str(rfs*100000+z_d*1000+a_d)+
    '\n',encoding='utf-8'))
        # Alpha
                    elif rtyp==4:
                        z_d-=2
                        a_d-=4
                        outputfile.write(bytes(str(rfs*100000+z_d*1000+a_d)+
    '\n',encoding='utf-8'))
        # Spontaneous fission
                    elif rtyp==6:
                        outputfile.write(bytes('SF'+'\n',encoding='utf-8'))
                    else:
                        outputfile.write(bytes('\n',encoding='utf-8'))

print("Completed")
```

Appendix D

Python code
Read_ENDF_neutron_fission_product_da

```
# Read ENDF VI neutron induced fission product data and write to a new file
# Only use data for thermal neutrons

import os, fileinput, math
import numpy as np

##8 Radioactivity and fission-product yield data
MF="8"

##NLIB Library Definition
##0 ENDF/B - United States Evaluated Nuclear Data File
NLIB="0"

##NSUB IPART ITYPE Sub-library Names
##4 0 4 Radioactive Decay Data
NSUB="4"
IPART="0"
ITYPE="4"

# Material number MAT
# U235
MAT="9228"

# ZA = 1000.0 x Z + A,
ZA="92238"

# Reaction type number MT
##MT Meaning Description
##454 Independent fission product yield

MT="454"

# Log file name
```

```
# Path to pre-processed ENDF files
filepath = os.path.join\
           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/ENDF/',\
            'endf-b-vi-8_nfpy.dat')
# Path to output file
outputfilepath = os.path.join\
           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/Datafiles/',\
            MF+MT.rjust(3,'_')+'.dat')

# Extract a number from a string
def extract_nbr(input_str):
    num_char = (".", "+", "-","E")

    l = ""
    num=""
    tokens = input_str.split()
    for token in tokens:

        for char in token:
            if char.isdigit() or (char in num_char):
                num = num + char

        try:
            l+=str((float(num)))
        except ValueError:
            pass
    if l=="":
        return(0)
    else:
        return(float(l))

#Start main loop
print("Start reading file")
# Find the file segment we're looking for

fseqn=MF.rjust(2)
fseqn=fseqn+(MT.rjust(3))+"    1"

# Energy in eV
E0=0.0253

print("Finding : ",fseqn," for ZA = all")

with open(outputfilepath, 'wb') as outputfile:
    outputfile.write(bytes('Radioactive decay data'+\
                           '\t'+'File '\
                           +MF+' Reaction type '+MT+'\n'\
                           ,encoding='utf-8'))
    outputfile.write(bytes('ZA'+'\t'+'ZAP'+'\t'+'FPS state'+'\t'\
                           +'Yield'+'\t'+'Yield 1sigma'+'\n',encoding='utf-8
    '))
# Iterate over input file
```

```
    lines=iter(fileinput.input(files=(filepath)))
    for line in lines:
    # Read MF MT NS
        mf_l=line[70:80]
        if mf_l == fseqn:
            za_l=line[0:11]
            line_no=lines.lineno()
    #         if str(int(extract_nbr(za_l)))==ZA:
            ZA=str(int(extract_nbr(za_l)))
            print("line number : ",line_no," ZA = ",za_l)
            line=next(lines)
    # get neutron energy
            en=extract_nbr(line[0:11])
    # jsut get thermal neutron data
            if en==E0:
                print("En :",en)
    # get number of items in fission product state array
                nn=int(extract_nbr(line[44:55]))
    # create a temp array
                tmp_a=np.zeros((nn,1),dtype=np.float64)
                tmp_pos=0
                n_lines=math.trunc(nn/6)
                n_last=nn-(n_lines*6)
                for i in range(0,n_lines):
                    line=next(lines)
                    for j in range(0,6):
                        tmp_a[tmp_pos]=extract_nbr(line[j*11:(j+1)*11])
                        tmp_pos+=1
    # read last line
                if n_last!=0:
                    line=next(lines)
                    for i in range(0,n_last):
                        tmp_a[tmp_pos]=extract_nbr(line[i*11:(i+1)*11])
                        tmp_pos+=1
    # Now we loop through the array writing the data
                for i in range(0,nn,4):
                    ZAPF=int(tmp_a[i])
                    FPS=float(tmp_a[i+1])
                    YI=float(tmp_a[i+2])
                    DYI=float(tmp_a[i+3])
                    outputfile.write(bytes(ZA+'\t'+str(ZAPF)+'\t',encoding='
    utf-8'))
                    outputfile.write(bytes(str(FPS)+'\t'+str(YI)+'\t',
    encoding='utf-8'))
                    outputfile.write(bytes(str(DYI)+'\n',encoding='utf-8'))

print("Completed")
```

Appendix E

Python code Read_ENDF_mass

```
# Read ENDF VI data and write to a new file
# To be used on files that went through PREPRO routine
# including SIGMA for broadend cross-sections

import os, fileinput, math
import numpy as np

# Set up variables
##MF Description
##1 General information
##2 Resonance parameter data
##3 Reaction cross sections
##4 Angular distributions for emitted particles
##5 Energy distributions for emitted particles
##6 Energy-angle distributions for emitted particles
##7 Thermal neutron scattering law data
##8 Radioactivity and fission-product yield data
##9 Multiplicities for radioactive nuclide production
##10 Cross sections for radioactive nuclide production
##12 Multiplicities for photon production
##13 Cross sections for photon production
##14 Angular distributions for photon production
##15 Energy distributions for photon production
##23 Photo- or electro-atomic interaction cross sections
##26 Electro-atomic angle and energy distribution
##27 Atomic form factors or scattering functions for photo-atomic
    interactions
##28 Atomic relaxation data
##30 Data covariances obtained from parameter covariances and sensitivities
##31 Data covariances for nu(bar)
##32 Data covariances for resonance parameters
##33 Data covariances for reaction cross sections
##34 Data covariances for angular distributions
##35 Data covariances for energy distributions
##39 Data covariances for radionuclide production yields
##40 Data covariances for radionuclide production cross sections
```

```
MF="1"
##NLIB Library Definition
##0 ENDF/B - United States Evaluated Nuclear Data File
##1 ENDF/A - United States Evaluated Nuclear Data File
##2 JEFF - NEA Joint Evaluated Fission and Fusion File (formerly
##JEF)
##3 EFF - European Fusion File (now part of JEFF)
##4 ENDF/B High Energy File
##5 CENDL - China Evaluated Nuclear Data Library
##6 JENDL - Japan Evaluated Nuclear Data Library
##21 SG-23 - Fission product library of theWorking Party on Evaluation
##Cooperation Subgroup-23 (WPEG-SG23)
##31 INDL/V - IAEA Evaluated Neutron Data Library
##32 INDL/A - IAEA Nuclear Data Activation Library
##33 FENDL - IAEA Fusion Evaluated Nuclear Data Library
##34 IRDF - IAEA International Reactor Dosimetry File
##35 BROND - Russian Evaluated Nuclear Data File (IAEA version)
##36 INGDB-90 - Geophysics Data
##37 FENDL/A - FENDL activation evaluations
##41 BROND - Russian Evaluated Nuclear Data File (original version)
NLIB="0"

##NSUB IPART ITYPE Sub-library Names
##0 0 0 Photo-Nuclear Data
##1 0 1 Photo-Induced Fission Product Yields
##3 0 3 Photo-Atomic Interaction Data
##4 0 4 Radioactive Decay Data
##5 0 5 Spontaneous Fission Product Yields
##6 0 6 Atomic Relaxation Data
##10 1 0 Incident-Neutron Data
##11 1 1 Neutron-Induced Fission Product Yields
##12 1 2 Thermal Neutron Scattering Data
##113 11 3 Electro-Atomic Interaction Data
##10010 1001 0 Incident-Proton Data
##10011 1001 1 Proton-Induced Fission Product Yields
##10020 1002 0 Incident-Deuteron Data
##10030 1003 0 Incident-Triton Data
##20030 2003 0 Incident-Helion (3He) Data
##20040 2004 0 Incident-Alpha data
NSUB="10"
IPART="1"
ITYPE="0"

# Material number MAT
# U235
MAT="9228"

# ZA = 1000.0     Z + A,
ZA="92238"

# Reaction type number MT
##MT Meaning Description
```

```
# 16 (n,2n)
# 17 (n,3n)
##18 (z,xf) total prompt fission
##19 (z,f) first chance fission
##20 (z,nf) second chance fission
##21 (z,2nf) third chance fission
##38 (z,3nf) fourth chance fission
##452  T  total number of neutrons per fission
##455  d  number of delayed neutrons per fission
##456  p  number of prompt neutrons per fission
##458 components of energy release in fission
# 101 (n,adsorption)
# 103 (n,p)
# 107 (n,alpha)
MT="451"

##x(n) is the nth value of x,
##y(n) is the nth value of y,
##NP is the number of pairs (x and y) given,
##INT(m) is the interpolation scheme identification number used in the mth
    range,
##NBT(m) is the value of n separating the mth and the (m + 1)th
    interpolation ranges.
##The list of allowed interpolation schemes is given in Table 16.
##Table 16: Definition of Interpolation Types
##INT Interpolation Scheme
##1 y is constant in x (constant, histogram)
##2 y is linear in x (linear-linear)
##3 y is linear in ln(x) (linear-log)
##4 ln(y) is linear in x (log-linear)
##5 ln(y) is linear in ln(x) (log-log)
##6 special one-dimensional interpolation law, used for charged-particle
    cross sections
##only
##11-15 method of corresponding points (follow interpolation laws of 1-5)
##21-25 unit base interpolation (follow interpolation laws of 1-5)
##Interpolation code, INT=1 (constant), implies that the function is
    constant and equal
##to the value given at the lower limit of the interval.

# Broadenend temperrture
t_br=600

# Log file name
# Path to pre-processed ENDF files
filepath = os.path.join\
           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/ENDF/IBMPc/
    PREPRO15/',\
            'ENDFB-8.DAT')
# Path to output file
outputfilepath = os.path.join\
           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/Datafiles/',\
```

```
135             MF+MT.rjust(3,'_')+'.dat')
136
137 # Extract a number from a string
138 def extract_nbr(input_str):
139     num_char = (".", "+", "-","E")
140
141     l = ""
142     num=""
143     tokens = input_str.split()
144     for token in tokens:
145
146         for char in token:
147             if char.isdigit() or (char in num_char):
148                 num = num + char
149         try:
150             l+=str((float(num)))
151         except ValueError:
152             pass
153     if l=="":
154         return(0)
155     else:
156         return(float(l))
157
158 #Start main loop
159 print("Start reading file")
160 # Find the file segment we're looking for
161
162 fseqn=MF.rjust(2)
163 fseqn=fseqn+(MT.rjust(3))+"    1"
164 ZA="1001"
165 # Energy in eV
166 E0=0.0253
167
168 print("Finding : ",fseqn," for ZA = all and T=",t_br,"K")
169
170 with open(outputfilepath, 'wb') as outputfile:
171     outputfile.write(bytes('Ratio of mass of atom to a neutron'+\
172                            '\t'+'File '\
173                            +MF+' Reaction type '+MT+'\n'\
174                            'ZA'+'\t'+'AWR'+'\n',encoding='utf-8'))
175     lines=iter(fileinput.input(files=(filepath)))
176     for line in lines:
177     # Read ZA AWR
178         mf_l=line[70:80]
179         if mf_l == fseqn:
180             za_l=str(int(extract_nbr(line[0:11])))
181             awr=str(extract_nbr(line[11:22]))
182             line_no=lines.lineno()
183             ZA=str(int(extract_nbr(za_l)))
184             print("line number : ",line_no," ZA = ",za_l)
185 # Ignore ZA where A=000 natural isotopes
186             if ZA[-3:] != '000':
```

```
187                     outputfile.write(bytes(za_l+'\t'+awr+'\n',encoding='utf-8'))
188
189 print("Completed")
```

Appendix F

Python code Isotope depletion

```
# Isotope depletion calculator
#
# Change log
# Use numpy matrix exponent function
# Add time dependence for neutron flux (2017-02-27)
# Change composition to 3.3% enrichment to compare against ORIGEN values
# V2.0
# Use full sized matrix for all isotopes
# Change to ZA=Z*1000+A nomenclature

# V2.1
# Calculate macroscopic cross section to include capture

# V2.2
# Include moderator and boron shim

# V2.3
# Do a second concentration calculation using average of initial and
# flux calculated after interval
# Removed moderator and boron shim

# V3.0
# Only calculate concentrations of nuclides whose diagonal elements are less
    than
# 2ln(1000) using matrix exponential method.
# Calculate flux from constant power


# V3.1
# Calculate decay chains given isotope and maximum half lives
# Use Bateman equation for shortlived isotopes
# Fixed bug in trans_m where matrix reference and not data was copied

# V3.2
# Remove fission products to see effect
# Fixed bug in matrix assignmnets where matrix reference and not data was
```

```
    copied
36
37 # V3.3
38 # Tweaked initial charge masses
39
40 # V3.4
41 # Only use isotopes that actually are produced
42 # File Isotopes_non_zero.dat
43 # Changed output file write statement bug that didn't write last step
    results
44 # Change flux calc to take into account mass of moderator
45 # Add ability to start calculations again from last line of data in output
    file
46
47 # V3.5
48 # Try a reduced isotope and see what happens
49 # Just 92 - 96
50 # Use temp cn_new matrix to calulate intermediate flux
51
52 # V3.6
53 # Macroscopic cross section fuel only
54 # Calc flux from power required didn't work
55 # Add capture macroscopic cross section
56 # Added burnup calculation
57
58 # V3.7
59 # Use only isotopes that are in ORIGEN calcs
60 # Output actual heat production
61 # Changed flux calculation
62 # Fixed bug that evaluated sub decay chains as well
63
64 # V3.71
65 # Calculate flux from fission only
66 # Expanded list of fisionable isotopes
67
68 # V3.73
69 # Use Bateman for isotopes with so that tm norm less than 2 ln 1000
70
71 # V3.8
72 # Evaluate shortlived isotope decay chains by parent and two daughters at a
    time
73 # New routine branch_decay takes branching fractions into account
74
75 # V3.81
76 # Use ORIGEN criteria for short lived
77 # Create queues from long-lived daughters
78 # V3.82
79 # Take account of fission product formation as well
80 # in short-lived decay chains
81
82 # V3.9
83 # Try ORIGEN method for flux prediction
```

```
84
85 # V3.91
86 # Added an extra term in the Taylor expansion for flux
87 # V3.92
88 # Only use second derivative for first step in flux calc
89
90 # V3.95
91 # Increase U238 and Pu240 adsorption cross sections to
92 # compensate for resonance adsorption
93 # Use flux_s for flux
94 # V3.96
95 # Increase U238 cross section for adsorption to 10b
96 # to see if it matched ORIGEN output
97
98 import scipy.linalg as la
99 import uuid
100 import numpy as np
101 from sys import exit
102 import os, fileinput
103 from time import time, localtime, strftime
104 from collections import defaultdict
105 from functools import reduce
106 import copy
107 import pdb
108
109 n_avo=6.022140857e23 # Avogadro's number
110 power_th=30*1e6 # J/s
111 energy_fission=3.24e-11  # Energy per fission in J
112 mass_neutron=1.00866491588 # Mass in atomic mass units
113 eV_J=1.602176565e-19 # 1 eV in J
114 m_fuel=999675.7 # Mass of fuel in grams
115 # Fuel density in g/cm3
116 dens_fuel=10.7
117 power_sp=power_th/1e6/(m_fuel/dens_fuel) # Specific power in MW/cm^3
118 restart=True # If False append to existing logfile
119 t=0
120 flux0=0
121
122 print("Fuel mass: ",m_fuel," g")
123
124 # Fraction of isotope in fuel
125 fuel_perc={92234:0.000264556, 92235:0.032917451, 92236:0, 92237:0,
    92238:0.966817993, 93237:0, 93238:0, 93239:0,\
126       94238:0, 94239:0, 94240:0, 94241:0, 94242:0, 94243:0, 95241:0,
    95242:0, 95243:0}
127
128 # Check if it adds up to 1
129 if sum(fuel_perc.values()) != 1:
130     print("Error is fuel percentages. Adds up to ",sum(fuel_perc.values()))
131
132 # Extract a number from a string
133 # will return zero if no number is found
```

```
def extract_nbr(input_str):
    num_char = (".", "+", "-","E","e")
    l = ""
    num=""
    tokens = input_str.split()
    for token in tokens:
        for char in token:
            if char.isdigit() or (char in num_char):
                num = num + char
        try:
            l+=str((float(num)))
        except ValueError:
            pass
    if l=="":
        return(0)
    else:
        return(float(l))

# List of isotopes in ZA = Z*10000 + A + ISO*100000
# Read from file
filepath = os.path.join\
           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/Datafiles/',\
            'Isotopes_Origen.dat')
istp=[]
with open(filepath) as f:
    istp = [int(line.rstrip('\n')) for line in f]
print('Read in ',len(istp),'isotopes')

# List of istope masses in ZA
# Read from file
filepath = os.path.join\
           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/Datafiles/',\
            'Isotopes_mass.dat')
istp_mass=[]

with open(filepath, 'r') as f:
    istp_mass= [(tuple(line.rstrip('\n').split('\t'))) for line in f]
print('Read in ',len(istp_mass),'isotope masses')

# Convert to g (values in file are in neutron mass)
istp_mass=dict(istp_mass)
istp_mass=({int(k):float(v)*mass_neutron for k, v in istp_mass.items()})

# List of spontaneous radioactive decay data ZA
# Read from file
# Radioactive decay data,File 8 Reaction type 457
# ZA,T1_2,Reaction type,Energy,Branching fraction,RFS,Daughter
# Create dictionaries
filepath = os.path.join\
           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/Datafiles/',\
            '8457_Origen.dat')
istp_decay=[]
```

```
186
187 with open(filepath, 'r') as f:
188     istp_decay= [(tuple(line.rstrip('\n').split('\t'))) for line in f]
189 print('Read in ',len(istp_decay),'radioactive isotopes')
190
191 # Half-life in s
192 istp_hl={}
193
194 # Define a dictionary with the changes in Z and A for different decay types
195
196 decay_parents={}
197 # Reaction type = 1 Beta minus Z+1 A
198 decay_parents[1.0]=1*1000+0
199 # Reaction type = 1.4 Beta minus alpha Z-1 A-4
200 decay_parents[1.4]=-1*1000-4
201 # Reaction type = 1.5 Beta minus neutron Z+1 A-1
202 decay_parents[1.5]=1*1000-1
203 # Reaction type = 2 EC Beta plus Z-1 A
204 decay_parents[2.0]=-1*1000+0
205 # Reaction type = 3 Isomeric trnsition Z A
206 decay_parents[3.0]=0*1000+0
207 # Reaction type = 4 Alpha decay Z-2 A-4
208 decay_parents[4.0]=-2*1000-4
209
210 # Reaction type, energy, branch frac, daughter
211 istp_prt=defaultdict(list)
212
213 # Reaction type, energy, branch frac, parent
214 istp_dgt=defaultdict(list)
215
216 for i in range(len(istp_decay)):
217     if istp_decay[i][6] != 'SF':
218         istp_prt[int(istp_decay[i][0])].append(float(istp_decay[i][2])) # RT
219         istp_prt[int(istp_decay[i][0])].append(float(istp_decay[i][3])) #
    Energy
220         istp_prt[int(istp_decay[i][0])].append(float(istp_decay[i][4])) #
    Branch fraction
221         istp_prt[int(istp_decay[i][0])].append(int(istp_decay[i][6]))   #
    Daughter
222
223 # Dictionary of daughter products
224         istp_dgt[int(istp_decay[i][6])].append(float(istp_decay[i][2])) # RT
225         istp_dgt[int(istp_decay[i][6])].append(float(istp_decay[i][3])) #
    Energy
226         istp_dgt[int(istp_decay[i][6])].append(float(istp_decay[i][4])) #
    Branch fraction
227         istp_dgt[int(istp_decay[i][6])].append(int(istp_decay[i][0]))   #
    Parent
228
229 # Add halflife as sum of different decays branching fractions
230         try:
231             istp_hl[int(istp_decay[i][0])]=istp_hl[int(istp_decay[i][0])] +
```

```
    float(istp_decay[i][1]) * float(istp_decay[i][4]) # Halflife * BF
232         except KeyError:
233             istp_hl[int(istp_decay[i][0])]=float(istp_decay[i][1]) * float(
    istp_decay[i][4]) # Halflife * BF
234 # Read fission cross-sections and energy release from file
235 filepath = os.path.join\
236           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/Datafiles/',\
237            '3_18_600K_Origen.dat')
238 cs_fi=defaultdict(list)
239 with open(filepath, 'r') as f:
240      for line in f:
241          cs_fi[int(tuple(line.rstrip('\n').split('\t'))[0])].append(tuple(
    line.rstrip('\n').split('\t'))[1:])
242
243 # Convert values to float and delete sigma = 0 values
244 for k, v in list(cs_fi.items()):
245     cs_fi[k]=(float(v[0][0]),float(v[0][1]))
246     if float(v[0][0]) == 0.0:
247         del cs_fi[k]
248 print('Read in ',len(cs_fi),'isotopes for fission')
249
250 # Read capture cross-sections and energy release from file
251 filepath = os.path.join\
252           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/Datafiles/',\
253            '3102_600K_Origen.dat')
254 cs_cp=defaultdict(list)
255 with open(filepath, 'r') as f:
256      for line in f:
257          cs_cp[int(tuple(line.rstrip('\n').split('\t'))[0])].append(tuple(
    line.rstrip('\n').split('\t'))[1:])
258
259 # Convert values to float and delete sigma = 0 values
260 for k, v in list(cs_cp.items()):
261     cs_cp[k]=(float(v[0][0]),float(v[0][1]))
262     if float(v[0][0]) == 0.0:
263         del cs_cp[k]
264 print('Read in ',len(cs_cp),'isotopes for capture')
265
266 # Read (n,alpha) cross-sections and energy release from file
267 filepath = os.path.join\
268           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/Datafiles/',\
269            '3107_600K_Origen.dat')
270 cs_na=defaultdict(list)
271 with open(filepath, 'r') as f:
272      for line in f:
273          cs_na[int(tuple(line.rstrip('\n').split('\t'))[0])].append(tuple(
    line.rstrip('\n').split('\t'))[1:])
274
275 # Convert values to float
276 for k, v in list(cs_na.items()):
277     cs_na[k]=(float(v[0][0]),float(v[0][1]))
278     if float(v[0][0]) == 0.0:
```

```
        del cs_na[k]
print('Read in ',len(cs_na),'isotopes for (n,alpha)')

# Read fission product yields from file
filepath = os.path.join\
           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/Datafiles/',\
            '8454_Origen.dat')

# Create an array for fission products
# 1288 unique fission products
# Dictionary key fissionable isotope
# Dict values tuples (ZAP, state, yield, yield uncertainty 1 sigma))
fs_pr=defaultdict(list)
with open(filepath, 'r') as f:
    for line in f:
        fs_pr[int(tuple(line.rstrip('\n').split('\t'))[0])].\
            append(tuple(map(float,line.rstrip('\n').split('\t')[1:])))
print('Read in ',len(fs_pr),'isotopes for fission products')

### List of fissionable isotopes in ZA convention
### with known energy release for thermal neutron
istp_fi=(91232, 92232, 92233, 92234, 92235, 92236, \
         92237, 92238,  93237, 93238,  \
         94238, 94239, 94240, 94241, 94242, 94243, \
         95241, 95242, 95242, 95243, 96242, 96243, \
         96244, 96245, 96246, 96247, 96248, 97249, \
         98249, 98251, 98252, 98253)
# removed 93236, 94236,

# Initialise four production and loss vectors for each isotope
# Parents for decay
# Parents for neutron adsorption
# Radioactive decay
# Neutron adsorption

# Radioactive decay vector
rad = np.zeros((len(istp),1),dtype=np.float64)

# Get half-lifes to calculate lambda
print('Get half-lifes to calculate lambda')
for i in istp_hl:
    try:
        rad[istp.index(i)] = -(np.log(2)/istp_hl[i])
    except:
        pass
        #print(i,' not in list')

# Neutron adsorption vector
print('Neutron adsorption vector')
nad = np.zeros((len(istp),1),dtype=np.float64)

# Add the capture cross-sections
```

```
331 print('Add the capture cross-sections')
332 for i in cs_cp:
333     try:
334         nad[istp.index(i)] += -(cs_cp[i][0])*1e-24
335     except:
336         print(i)
337 
338 # Add the fission cross-sections
339 print('Add the fission cross-sections')
340 for i in cs_fi:
341     try:
342         nad[istp.index(i)] += -(cs_fi[i][0])*1e-24
343     except:
344         pass
345 
346 # Add the (n,alpha) cross-sections
347 print('Add the (n,alpha) cross-sections')
348 for i in cs_na:
349     try:
350         nad[istp.index(i)] += -(cs_na[i][0])*1e-24
351     except:
352         print(i)
353 
354 # Parents for neutron adsorption array
355 # (n,gamma) and fission products
356 
357 pna = np.zeros((len(istp),len(istp)),dtype=np.float64)
358 
359 # Add the (n,gamma) cross-sections
360 # ZA(n,g)ZA+1
361 print('Add the (n,gamma) cross-sections')
362 # Adjust 92238 cross section
363 ##cs_cp[92238]=(10, 4806200.)
364 
365 for i in cs_cp:
366     try:
367         pna[istp.index(i+1),istp.index(i)] += +(cs_cp[i][0])*1e-24
368     except:
369         print(i+1,' is not in capture list')
370 
371 # Add the (n,alpha) cross-sections
372 # ZA(n,a)Z-2 A+1-4
373 print('Add the (n,alpha) cross-sections')
374 for i in cs_na:
375     try:
376         pna[istp.index(i-2*1000-3),istp.index(i)] += +(cs_na[i][0])*1e-24
377     except:
378         print(i-2*1000-3,' is not on n,alpha list')
379 
380 # Add the fission products
381 print('Add the fission products')
382 for i in istp_fi:
```

```
383     for j in fs_pr[i]:
384         pna[istp.index(int(j[0])),istp.index(i)]+=j[2]*cs_fi[i][0]*1e-24
385 
386 # Parents for radioactive decay array
387 # alpha and beta decay
388 print('Add parents for radioactive decay array')
389 prd = np.zeros((len(istp),len(istp)),dtype=np.float64)
390 
391 for k,v in istp_prt.items():
392     for i in range(0,len(v),4):
393         if v[i+3] != 'SF':
394             try:
395                 prd[istp.index(v[i+3]),istp.index(k)]=v[i+2]*(-rad[istp.
    index(k)])
396                 # Daughter               parent         branch frac*decay
    constant
397             except:
398                 pass #print(v[i+3])
399 
400 # Initialise transition matrix
401 tm = np.zeros((len(istp),len(istp)),dtype=np.float64)
402 
403 def trans_m(fl):
404 # This creates the transition matrix by adding the parent decay,
405 # parents for nuclear adsorption, radioactive decay and adsorption arrays
406 # multiplied by fl flux
407     global tm,prd,pna,istp,cn
408     tm=prd.copy()
409     tm+=pna.copy()*fl
410 # Maybe do this once, create a diagonal and just add to tm every time
411     for k,v in istp_hl.items():
412         try:
413             tm[istp.index(k),istp.index(k)]+=rad[istp.index(k)][0]
414         except:
415             pass
416 
417     for i in istp:
418         tm[istp.index(i),istp.index(i)]+=nad[istp.index(i)][0]*fl
419     # Adjust for resonance capture
420     p=0.87 # Resonance escape probability
421     tm[istp.index(92238),istp.index(92238)]+=-((1-p)/p)*fl/cn[istp.index
    (92238)]
422     tm[istp.index(92239),istp.index(92238)]+=+((1-p)/p)*fl/cn[istp.index
    (92238)]
423     return
424 
425 # Calculate initial flux from thermal power output
426 # Calculate energy released by all fissionable isotopes
427 en_fi=0.0
428 
429 cn = np.zeros((len(istp),1),dtype=np.float64)
430 if restart==True:
```

```
431     # Create initial coefficient matrix
432     # Initialise concentration vector in no of atoms
433     # Multiply percentage with fuel mass divide by atomic mass mult by
        Avogadro
434     print('Initialise x[0]')
435     for i in fuel_perc:
436         cn[istp.index(i)]=fuel_perc[i]*m_fuel/istp_mass[i]*n_avo
437     for i in istp_fi:
438         en_fi += cs_fi[i][0]*1e-24 * cs_fi[i][1] * eV_J * cn[istp.index(i)
        ][0]
439     flux0=float(power_th/en_fi)
440     print('{:s}; {:3e};'.format('Flux :',flux0))
441     t=0
442
443 else:
444     # Read in last values from data file
445     filepath = os.path.join\
446        ('C:/Users/keysertl/Documents/Nuclear/Dissertation/Datafiles/',\
447         'n_conc.csv')
448     with open(filepath, 'rb') as f:
449         frst_ln=f.readline() # Read the first line.
450         f.seek(-2, 2)              # Jump to the second last byte.
451         while f.read(1) != b"\n": # Until EOL is found...
452             f.seek(-2, 1)          # ...jump back the read byte plus one more
        .
453         last_ln = f.readline()       # Read last line.
454         istp_names=frst_ln.decode("utf-8").rstrip('\n').split(',') # Split
        in names
455         istp_values=last_ln.decode("utf-8").rstrip('\n').split(',')
456         for (i,j) in zip(istp_names,istp_values):
457             try:
458                 cn[istp.index(int(i))]=extract_nbr(j)*n_avo  #*(m_fuel
        /1000000)
459             except:
460                 if i=='Flux':
461                     flux0=extract_nbr(j)
462                     print('{:s}; {:3e};'.format('Flux :',flux0))
463                 if i[0]=='t':
464                     t=extract_nbr(j)
465                     print('{:s}; {:3.0f};'.format('Time :',t))
466
467 # Check if all isotopes have a known mass
468 m_all=0.0
469 for i,v in enumerate(cn):
470     if v!=0:
471         try:
472             m_all+=v[0]/n_avo*istp_mass[istp[i]]
473         except:
474 ##            print('Missing mass',istp[i])
475             pass
476
477 trans_m(flux0)
```

```
478
479 def flux_s():
480 # Calculate flux from energy release
481 # to ensure required power
482
483     global power_th, cs_fi, cn, cs_cp, cs_na, istp_dgt, istp, cn_prev,step
484 # Calculate energy from fission
485     en_fiss=0
486     for i in istp_fi:
487         en_fiss += cs_fi[i][0]*1e-24 * cs_fi[i][1] * eV_J * cn[istp.index(i)
    ][0]
488 ##     print('Energy fission',en_fiss)
489
490 # Calculate heat from neutron capture reactions
491     en_cap=0
492     for k, v in cs_cp.items():
493         en_cap += float(v[0]*1e-24 * v[1] * eV_J * cn[istp.index(k)][0])
494 ##     print('Energy capture',en_cap)
495
496 # Calculate heat from n,alpha  reactions
497     en_na=0
498 ##     for k, v in cs_na.items():
499 ##         en_na += float(v[0]*1e-24 * v[1] * eV_J * cn[istp.index(k)][0])
500 ####     print('Energy n alpha',en_na)
501
502 # Calculate heat from radioactive decay
503     en_rd=0
504     for k,v in istp_dgt.items():
505         for i in range(0,len(v),4):
506             try:
507                 if v[i+3] != 'SF' and cn[istp.index(k)]>0.0:
508                     en_rd += float(cn_prev[istp.index(k)][0]*(1-np.exp(rad[
    istp.index(k)][0]*step)) * v[i+1] * v[i+2]*eV_J)
509             except:
510                 pass
511 ####     print('Energy decay',en_rd)
512
513     flux_tot=float((power_th - en_rd/step)/(en_fiss + en_cap + en_na))
514     return flux_tot
515
516 def flux(fl0,t_p,cn_m):
517 # Calculate the avelrage flux in a period t_p with starting flux fl0 and
518 # concentration matrix cn_m
519
520 # Calculate macroscopic cross-section for flux calculation
521 # Also first and second derivatives
522     global power_sp, boron_shim, power_th, m_fuel, dens_fuel, tm
523     mac_csf0 = 0
524     mac_csf1 = 0
525
526 # Macroscopic crosssection sum of fission cross-section*molecules/cm^3
527 # Sigma'(0) first derivative evaluated from X'(0)=AX(0) and
```

```
# X''(0)=AX'(0)=A^2X(0)
    c1=np.dot(tm,cn_m)

    for i in istp_fi:
        mac_csf0+=cs_fi[i][0]*1e-24*cn_m[istp.index(i)][0]/(m_fuel/dens_fuel) #m_all*dens_all
        mac_csf1+=cs_fi[i][0]*1e-24*c1[istp.index(i)][0]/(m_fuel/dens_fuel) #m_all*dens_all

    for i in cs_cp:
        mac_csf0+=cs_cp[i][0]*1e-24*cn_m[istp.index(i)][0]/(m_fuel/dens_fuel) #m_all*dens_all
        mac_csf1+=cs_cp[i][0]*1e-24*c1[istp.index(i)][0]/(m_fuel/dens_fuel) #m_all*dens_all

    fl=fl0*(1 - t_p/2*mac_csf1/mac_csf0)

    if abs((fl0-fl)/fl0)>0.2:
        print('Flux difference is more than 20%')
        print('Flux 0',fl0,'Flux 1',fl)
    return fl


q_tot=0

# Function to print non-zero values in a row of matrix for an isotope
# Prints parent, transformation coeff and number of atoms transformed
def pm(mat,ist):
    global cn,tm,prd,istp
    for ii,vv in enumerate(mat[istp.index(ist)]):
        if vv!=0:
            print('{:}; {:2e}; {:2e}; {:2e};'.format(istp[ii],tm[istp.index(ist),ii],cn[ii][0],cn[ii][0]*tm[istp.index(ist),ii]))

# Function to flatten a list
def flatten_all(iterable):
    for elem in iterable:
        if not isinstance(elem, list):
            yield elem
        else:
            yield from flatten_all(elem)

# find decay chains, return list of chains
def decay_chain(chain_list,halflife):
    for paths in chain_list:
        parent=paths[-1]
        try:
            parent_hl=istp_hl[parent]
        except: # If no halflife, then it is stable
            parent_hl=halflife-1 # Ensure it gets added
        if parent_hl < halflife:
            daughters=istp_prt[parent]
```

```
575             for i in range(0,len(daughters),4):
576                 if daughters[i+3] != 'SF': # No spontaneous decay
577                     if i>0: # More than one
578                         chain_list.append(copy.deepcopy(paths)) # copy
    static list
579                         del paths[paths.index(parent)+1:] # start new list
    with current parent
580                         paths.append(daughters[i+3])
581                     else:
582                         paths.append(daughters[i+3])
583                     chain_list=decay_chain(chain_list,halflife)
584         else:
585             return chain_list
586         return chain_list
587
588 def branch_decay(con,lmbd,bf,t):
589 # General decay formula with branching fractions
590 # con: initial concentration list
591 # lmbd : decay constants
592 # bf: branching fractions
593     m = len(lmbd)
594     if len(con) != m:
595         raise ValueError("Please pass equal number of decay"
596                          " constants as initial concentrations"
597                          " (you may want to pad lmbd with zeroes)")
598     m+=1 # range functions ends at m-1
599     br_fr=np.zeros((m+1,m+1),dtype=np.float64) # Branch frac[a,b] portion of
     a that transforms into b
600     for j in range(len(bf)):
601         br_fr[j+1,j+2]=bf[j]
602     N = np.zeros((m-1,1),dtype=np.float64)
603     # Calculate all the K values
604     K = np.zeros((m,m),dtype=np.float64)
605     K[1,1]=con[0]
606     for c in range(2,m):
607         for n in range(1,c):
608             sum_k=0
609             for i in range(n,c):
610                 sum_k+=br_fr[i,c]*lmbd[i-1]*K[i,n]
611             K[c,n]=sum_k/(lmbd[c-1]-lmbd[n-1])
612         K[c,c]=con[c-1]
613         for n in range(1,c):
614             K[c,c]-=K[c,n]
615
616     for c in range(1,m):
617         sum_k=0
618         for i in range(1,c+1):
619             sum_k+=K[c,i]*np.exp(-lmbd[i-1]*t)
620         N[c-1]=sum_k
621
622     return N
623
```

```
624 def branch_decay_list(ist,con,t):
625 # General decay formula with branching fractions
626 # con: initial concentration list
627 # ist: list of isotopes in decay chain
628 # lmbd : decay constants
629 # bf: branching fractions
630     m = len(ist)
631     if len(con) != m:
632         raise ValueError("Please pass equal number of decay"
633                          " constants as initial concentrations"
634                          " (you may want to pad lmbd with zeroes)")
635     bf=[]
636     lmbd=[]
637     for i in range(m):
638         lmbd.append(-rad[istp.index(ist[i])])
639         if i < m-1:
640             bf.append(prd[istp.index(ist[i+1]),istp.index(ist[i])]/lmbd[i])
641     m+=1 # range functions ends at m-1
642     br_fr=np.zeros((m+1,m+1),dtype=np.float64) # Branch frac[a,b] portion of
      a that transforms into b
643     for j in range(len(bf)):
644         br_fr[j+1,j+2]=bf[j]
645     N = np.zeros((m-1,1),dtype=np.float64)
646     # Calculate all the K values
647     K = np.zeros((m,m),dtype=np.float64)
648     K[1,1]=con[0]
649     for c in range(2,m):
650         for n in range(1,c):
651             sum_k=0
652             for i in range(n,c):
653                 sum_k+=br_fr[i,c]*lmbd[i-1]*K[i,n]
654             K[c,n]=sum_k/(lmbd[c-1]-lmbd[n-1])
655         K[c,c]=con[c-1]
656         for n in range(1,c):
657             K[c,c]-=K[c,n]
658
659     for c in range(1,m):
660         sum_k=0
661         for i in range(1,c+1):
662             sum_k+=K[c,i]*np.exp(-lmbd[i-1]*t)
663         N[c-1]=sum_k
664
665     return N
666
667 def bateman_parent(lmbd, t):
668 ##    Calculate daughter concentrations (number densities) from single
    parent
669 ##    Assumes an initial parent concentration of one (and zero for all
    daughters)
670 ##    Parameters
671 ##    ----------
672 ##    lmbd: array_like
```

```
673 ##         decay constants (one per species)
674 ##     t: float time
675
676     n = len(lmbd)
677     N = np.zeros((n,1),dtype=np.float64)
678     lmbd_prod = 1
679     for i in range(n):
680         if i > 0:
681             lmbd_prod *= lmbd[i-1]
682         sum_k = 0
683         for k in range(i+1):
684             prod_l = 1
685             for l in range(i+1):
686                 if l == k:
687                     continue
688                 prod_l *= lmbd[l] - lmbd[k]
689             sum_k += np.exp(-lmbd[k]*t)/prod_l
690         N[i] = lmbd_prod*sum_k
691     return N
692
693 def bateman_full(y0s, lmbd, t):
694 ##     Calculates a linear combination of single-parent chains
695 ##     Generalized helper function for when y0 != [1, 0, 0, ... ]
696 ##     Parameters
697 ##     ----------
698 ##     y0s: array_like
699 ##         Initial concentrations
700 ##     t: float
701 ##         time
702     n = len(lmbd)
703     if len(y0s) != n:
704         raise ValueError("Please pass equal number of decay"
705                          " constants as initial concentrations"
706                          " (you may want to pad lmbd with zeroes)")
707     N = np.zeros((n,1),dtype=np.float64)
708
709     for i, y0 in enumerate(y0s):
710         if y0 == 0:
711             continue
712         Ni = bateman_parent(lmbd[i:], t)
713         for j, yj in enumerate(Ni, i):
714             N[j] += y0*yj
715     return N
716
717 def bateman_mod(ist,c,t):
718     # Modified Bateman as developed by Vondy
719     # ist : list of isotopes
720     # c : Array of initial concentrations
721     # t : step
722
723     n=len(ist)
724     N = np.zeros((n,1),dtype=np.float64)
```

```
725     d = np.zeros((n,1),dtype=np.float64)
726     sum_j=0
727     pdb.set_trace()
728     for ic, x in enumerate(ist): # Populate d first
729         d[ic]=-tm[istp.index(x),istp.index(x)]
730     for ic, x in enumerate(ist):
731 #         print(ic,x)
732 
733         i=ic+1
734         sum_k=0
735         for k in range(1,i):
736             prod_n=1
737             for n in range(k,i):
738                 prod_n*=tm[istp.index(ist[n]),istp.index(ist[n-1])]/d[n-1]
739             if prod_n < 1e-6:
740                 prod_n = 0 # Ignore contribution of isotope k if prod is
    less than 1e-06
741             sum_k+=c[k-1]*prod_n
742             sum_j=0
743             for j in range(k,i):
744                 prod_n=1
745                 for n in range(k,i):
746                     if n!=j:
747                         prod_n*=d[n-1]/(d[n-1]-d[j-1])
748                 if d[i-1]!=d[j-1]:
749                     sum_j+=d[j-1]*(np.exp(-d[j-1]*t)-np.exp(-d[i-1]*t))/(d[i
    -1]-d[j-1])*prod_n
750                 else:
751                     sum_j+=d[j-1]*t*np.exp(-d[j-1]*t)*prod_n
752         N[ic]=c[ic]*np.exp(-d[ic]*t) + sum_k*sum_j
753     return N
754 
755 def gauss_seid(ist,c):
756     # Solving the concentrations caused by long lived precurosrs
757     # to shortlived isotopes
758     # ist : short-lived isotopes
759     # c : concentrations
760     max_iter=1000
761     sum_j=0
762     dgt=istp_dgt[ist]
763     Nk=cn[istp.index(ist)]
764 ##    pdb.set_trace()
765     for n in range(max_iter):
766         for j in range(0,len(dgt),4): #For all parents
767             sum_j+=tm[istp.index(ist),istp.index(dgt[j+3])]*cn[istp.index(
    dgt[j+3])]
768         Nk1=-1/tm[istp.index(ist),istp.index(ist)]*sum_j
769         if Nk1/Nk<1e-8:
770             break
771         else:
772             Nk=Nk1
773     return Nk1
```

```
774
775 # Function to check if a sublist exists in a list
776 def sublist_exists(list, sublist):
777     for i in range(len(list)-len(sublist)+1):
778         if sublist == list[i:i+len(sublist)]:
779             return True #return position (i) if you wish
780     return False #or -1
781
782 ##pdb.set_trace()
783
784 # Start main loop
785 # Use matrix exponential method for longlived
786 # and Bateman for short-lived
787
788 # Step in secs
789 step=int(3600)
790 # Halflife threshold for shortlived isotopes
791 shortlived_hl=-np.log(2)/(np.log(0.001)/step)
792 iterations=700
793
794 # Open logfile and write headers
795 # Log file name
796 filepath = os.path.join\
797           ('C:/Users/keysertl/Documents/Nuclear/Dissertation/Datafiles/',\
798            'n_conc.csv')
799 if restart==True:
800     file_mode='wb'
801 else:
802     file_mode='ab'
803
804 with open(filepath, file_mode) as logfile:
805     q_tot=q_fs=q_rd=q_cp=q_na=0
806     burnup=0
807     n_out=np.zeros((len(istp),1),dtype=np.float64)
808     if restart==True:
809         for i in istp:
810             logfile.write(bytes(str(i)+',',encoding='utf-8'))
811         logfile.write(bytes('Burnup'+',',encoding='utf-8'))
812         logfile.write(bytes('Flux,Q_fs,Q_rd,Q_cp,Q_na,Q_tot,P',encoding='utf
    -8'))
813         logfile.write(bytes((',t\n'),encoding='utf-8'))
814     # Write initial values to file
815         n_out=cn/n_avo #/(m_fuel/1000000)
816         np.savetxt(logfile, n_out.reshape(1, n_out.shape[0]), delimiter=',',
     newline='', header='', footer='', )
817         logfile.write(bytes(','+str(burnup)+',',encoding='utf-8'))
818         logfile.write(bytes(str(flux0)+','+str(q_fs)+','+str(q_rd)+','+str(
    q_cp)+','+str(q_na)+','+str(q_tot),encoding='utf-8'))
819         logfile.write(bytes((','+str(power_sp*(m_fuel/dens_fuel))+','+str(t)
    +'\n'),encoding='utf-8'))
820     start_time=time()
821     print('{:s}; {:s};'.format('Start at ', strftime("%a, %d %b %Y %H:%M:%S"
```

```
    , localtime())))
822     st=1
823 
824     while t<step*iterations:
825 
826 # Calculate average interval flux from fission
827         cn_prev=cn.copy()
828         flux1=flux_s()
829         trans_m(flux1)
830         print('{:s} {:2e}'.format('Calculated average interval flux ',float(
    flux1)))
831 
832 # First calculate shortlived decay chains
833         parents_done=[] # List of completed parents
834         short_lived=[]
835         s_lived=[]
836         s_lived_un=[]
837         parent_list={} # Dictionary of parent concentrations
838         ll_daughter_list=[] # List of long-lived daughters
839         shortl_list={} # Dictionary of initial concentrations of shortlived
    isotopes
840         long_lived_dghtr=[] # List of end of chain products
841         cn_list={} # Dictionary of concentrations of shortlived chains
842         chain_list=[] # List of decay chains used to check if chain has been
     evaluated already
843         for j in range(len(cn)):
844             try:
845                 halfl=istp_hl[istp[j]]
846 # There is some of the isotope with a short halflife and does not have a
    shortlived parent
847                 if halfl<shortlived_hl and cn[j]>0:
848                     decay_list=decay_chain([[istp[j]]],shortlived_hl) # Find
     the decay chains
849                     for m in range(len(decay_list)):
850                         chain_list.append(decay_list[m]) # Create list of
    chains
851                         s_lived.append(decay_list[m][:-1]) # List without
    long-lived daughter
852                         ll_daughter_list.append(decay_list[m][-1])
853 
854             except: #No half life
855                 pass
856 
857 # Copy unique chains to list
858         max_ch_len=0
859         for ch in chain_list:
860             if len(ch)>max_ch_len:
861                 max_ch_len=len(ch)
862             for ch2 in chain_list:
863                 if ch!=ch2 and sublist_exists(ch,ch2):
864                     chain_list.remove(ch2)
865 # Do a second pass
```

```
866         for ch in chain_list:
867             for ch2 in chain_list:
868                 if ch!=ch2 and sublist_exists(ch,ch2):
869                     chain_list.remove(ch2)
870 
871         done_conc={} # dictionary of completed parent-daughter
    concentrations
872         cn_list={} # dictionary of completed concentrations
873         decay_const=[]
874         long_lived_dghtr=set(ll_daughter_list)
875         n=0 # loop counter
876         done_list=[] # List of already evaluated isotopes
877         for lld in long_lived_dghtr: # Loop through all the long-lived
    daughters
878             dec_ch=[]
879             for ch in chain_list: # Find all chains that end in lld
880                 if ch[-1]==lld:
881                     parent=ch[0]
882                     parents_done.append(parent)
883                     dec_ch.append(ch)
884                 # assign the concentrations, if already done use zero
885             for m in range(len(dec_ch)):
886                 con_list=np.zeros((len(dec_ch[m]),1),dtype=np.float64)
887                 done_parent=[] # Keeps track of parents of short-lived to
    not add them twice
888 ##                    long_lived_dghtr.append(dec_ch[m][-1]) # Add last item
     in list
889                 for k,l in enumerate(dec_ch[m]):
890 ##                    decay_const[k]=float(-rad[istp.index(l)])
891                     if k < (len(dec_ch[m])-1):
892                         # assign values
893                         if l==dec_ch[m][0]: # Parent
894                             con_list[k]=+float(cn[istp.index(l)])
895                         elif l not in done_list: # If not done use original
    concentration a long with parent
896                             con_list[k]=float(cn[istp.index(l)])
897                             con_list[k-1]=float(cn[istp.index(dec_ch[m][k
    -1])])
898                             done_parent.append(dec_ch[m][k-1])
899                     else:
900                         con_list[k]=0 # Don't copy last item in chain
901                 # Check for branching fractions and if parent, daughter
    was done already
902                 dc=branch_decay_list(dec_ch[m],con_list,step) # Evaluate the
     decay chain
903 ##                dc=bateman_mod(dec_ch[m],con_list,step) # Evaluate the
    decay chain
904 ##                        print('Bateman result',dc)
905                 # Adjust concentration matrix
906                 cn_list[dec_ch[m][0]]=float(dc[0]) # parent never gets added
907                 for k in range(1,len(dec_ch[m])):
908                     try:
```

```
909                         cn_list[dec_ch[m][k]]+=float(dc[k])
910                     except KeyError:
911                         if dec_ch[m][k] not in done_list:
912                             cn_list[dec_ch[m][k]]= float(dc[k])
913                     if dec_ch[m][k] not in done_list and dec_ch[m][k]!=
    parent:
914                         done_list.append(dec_ch[m][k])
915             n+=1 # Loop
916 
917         short_lived=flatten_all(s_lived)
918         short_lived_un=set(short_lived)
919 
920 # Create new concentration and transmutation matrix
921 # by removing short-lived chain concentrations and decay constants
922         cn_long=cn.copy()
923         cn_new=cn.copy()
924         tm_long=tm.copy()
925 
926         for i in short_lived_un:
927             cn_long[istp.index(i)] = 0 # Subtract isotopes that take part in
     short lived decay
928         print('Long lived istopes',len(cn_long))
929 
930 # Calculate new concentration matrix using initial flux
931         if len(short_lived_un)>0:
932             cn_long=np.dot(la.expm(tm_long*step),cn_long,out=None)
933     # Return values to original matrices
934             for i in short_lived_un:
935                 cn_long[istp.index(i)]+=cn_list[i]
936             for i in long_lived_dghtr:
937                 cn_long[istp.index(i)]+=cn_list[i]
938         else:
939             cn_long=np.dot(la.expm(tm*step),cn_long,out=None)
940         cn=cn_long.copy()
941 
942 # Calculate heat from fission
943         q_fs=0
944         for i in istp_fi:
945             q_fs += float(step * flux1 * cs_fi[i][0]*1e-24 * cs_fi[i][1] *
    eV_J * cn_prev[istp.index(i)][0])
946 
947 # Calculate heat from radioactive decay
948         q_rd=0
949         dheat=0
950         for k,v in istp_dgt.items():
951             for i in range(0,len(v),4):
952                 try:
953                     if v[i+3] != 'SF' and cn_prev[istp.index(k)]>0.0:
954                         dheat = float(cn_prev[istp.index(k)][0]*(1-np.exp(
    rad[istp.index(k)][0]*step)) * v[i+1] * v[i+2]*eV_J)
955                         q_rd += dheat
956                 except:
```

```
                    pass

# Calculate heat from neutron capture  reactions
        q_cp=0
        for k, v in cs_cp.items():
            q_cp += float(step * flux1 * v[0]*1e-24 * v[1] * eV_J * cn_prev[
    istp.index(k)][0])

# Calculate heat from n,alpha  reactions
        q_na=0
        for k, v in cs_na.items():
            q_na += float(step * flux1 * v[0]*1e-24 * v[1] * eV_J * cn_prev[
    istp.index(k)][0])

 # Calculate total heat
        q_tot=(q_fs+q_cp+q_na) + q_rd
##        print('{:s}; {:2e};'.format('Calculated heat release MJ',q_tot/1e6
    ))
##        print('{:s}; {:2e};'.format('Calculated average power MW',q_tot/1
    e6/step))
        burnup+=q_tot/1e6/step * step/3600/24 #/ (m_fuel/1e6)
        flux0=flux1
        st+=1
        t+=step
        print('Step ',st,' at time',strftime("%a, %d %b %Y %H:%M:%S",
    localtime()))

# Write output to file
        n_out=cn/n_avo #/(m_fuel/1000000)
        np.savetxt(logfile, n_out.reshape(1, n_out.shape[0]), delimiter=',',
     newline='', header='', footer='', )
        logfile.write(bytes(','+str(burnup)+',',encoding='utf-8'))
        logfile.write(bytes(str(flux0)+','+str(q_fs)+','+str(q_rd)+','+str(
    q_cp)+','+str(q_na)+','+str(q_tot),encoding='utf-8'))
        logfile.write(bytes((','+str(q_tot/1e6/step)+','+str(t)+'\n'),
    encoding='utf-8'))

logfile.close()

print("Burnup in MWd/t")
print(burnup)
print("End at :",time())
print((time()-start_time)/iterations," per iteration")

m_burnt=0
print("Fuel element percentage in atom gram per ton fuel")
for x in fuel_perc:
    fuel_perc[x]=cn[istp.index(x)]/n_avo #/m_fuel*1e6
    print('{:d}; {:3e}; {:3e}'.format(x,float(fuel_perc[x]),float(fuel_perc[
    x]*n_avo*istp_mass[x])))
    m_burnt+=fuel_perc[x]/n_avo*istp_mass[x]
print('{:s}; {:3f}; {:s}'.format("Fuel mass: ",float(m_burnt/1e6)," tons"))
```

References

[1] Bateman, H., *Solution of a system of differential equations occurring in the theory of radioactive transformations*. Proc. Cambridge Philos. Soc. 15, 423–427. 1910.

[2] Bell, M.J., *ORNL- 4628, ORIGEN - THE ORNL Isotope Generation And Depletion Code*. Oak Ridge National Laboratory, Oak Ridge, Tennessee 37830, May 1973

[3] Bodansky, D., *Nuclear energy : principles, practices, and prospects 2nd ed*, Springer-Verlag New York 2004, 1996, ISBN 0-387-20778-3.

[4] Bowman, S.M., *SCALE 6: Comprehensive Nuclear Safety Analysis Code System*. Nuclear Technology Vol 174:128-148, May 2011.

[5] Cullen, D.E., *PREPRO 2015 ENDF/B Pre-processing Codes (ENDF/B-VII Tested), IAEA-NDS-39*. International Atomic Energy Agency Nuclear Data Services, Vienna International Centre, P.O. Box 100, A1400 Vienna, Austria, 2015.

[6] Diop, C.M., *Integral form of nuclide generation and depletion equations for Monte Carlo simulation. Application to perturbation calculations* .DEN/DANS/DM2S/SERMA, CEA/Saclay, 91191 Gif sur Yvette, France, June, 2007.

[7] Garland, M.J., *Decay Heat Estimates for MNR*, Technical Report 1998-03, McMaster University. Downloaded from http://http://www.nuceng.ca/papers/decayhe1b.pdf. Accessed on 2017-10-22. 1998

[8] Gauld, I.C., Radulescu, G., Ilas, G., Murphy, B.D., Williams, M.L. and Wiarda, D., *Isotopic Depletion and Decay Methods and Analysis Capabilities in SCALE*. Nuclear Technology Vol 174:169-195, May 2011.

[9] Harr, L.J., *Precise Calculation Of Complex Radioactive Decay Chains*. Air Force Institute Of Technology. Wright-Patterson Air Force Base, Ohio. March 2007.

[10] Herman, H. and Trkov, A., *ENDF-6 Formats Manual: Data Formats and Procedures for the Evaluated Nuclear Data File ENDF/B-VI and ENDF/B-VII*. National Nuclear Data Center, Brookhaven National Laboratory, Upton, NY 11973-5000, www.nndc.bnl.gov, June 2009

[11] Ilas, G.,Gauld, I.C. andLiljenfeldt, H., *Validation of ORIGEN for LWR used fuel decay heat analysis with SCALE*. Nuclear Engineering and Design 273:58–67, 2014

[12] International Atomic Energy Agency Nuclear Data Services, 2017. *PREPRO 2017 Home Page* [online]. Vienna International Centre, P.O. Box 100, A1400 Vienna, Austria . Available from: https://www-nds.iaea.org/public/endf/prepro/

[13] MacFarlane, R.E., *An Introduction to the ENDF formats.* Los Alamos National Laboratory, Los Alamos, USA, 2000.

[14] MacFarlane, R.E. and Kahler, A.C., *Methods for Processing ENDF/B-VII with NJOY.* Nuclear and Particle Physics, Astrophysics and Cosmology Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM 87545, October 2010

[15] Mara, H.,Katakura, J. and Nakagawa, T., *A Computer Code for Calculation of Radioactive Nuclide Generation and Depletion, Decay Heat and γ Ray Spectrum FPGS90.*Japan Atomic Energy Research Institute, JAERl-Data/Code-95-014, JP9605120, Oct 1995.

[16] Nuclear Energy Agency, Organisation for Economic Co-operation and Development, 2017. Paris, France Available from https://www.oecd-nea.org/dbforms/data/eva/evatapes/endfb_6_rel8/

[17] Nuclear Power for Everybody, 2017. *Resonance Escape Probability.* Available from http://www.nuclear-power.net/nuclear-power/reactor-physics/nuclear-engineering-fundamentals/neutron-nuclear-reactions/neutron-absorption/neutron-absorption-cross-section/.[Accessed 31 October 2017]

[18] Pusa, M. and Leppänen, J., *Computing the Matrix Exponential in Burnup Calculations.* Nuclear Science and Engineering: 164, 140–150. 2010

[19] Salahuddin, A. and Arshad, M., *Isotopic Composition of Actinides and Fission Products in the Irradiated Fuel of the 10MW Low Enriched PARR Reference Core.* Reactor Physics Group, Nuclear Engineering Division, Pakistan Institute of Nuclear Science & Technology, P.O. Nilcre, Islamabad, Oct 1987

[20] Stacey, W., *Nuclear Reactor Physics.* Wiley, 2nd ed, ISBN-13: 978-3527406791, 2004.

[21] Strenge, D.L., *A General Algorithm for Radioactive Decay with Branching and Loss From a Medium.* Presented at the Health Physics Society and American Association of Physics In Medicine Conference, July 23-27, 1995, Boston, Massachusetts

[22] Subbaiah, K.V., *ISOGEN: Interactive isotope generation and depletion code.* Radiation Protection and Environment, Vol 39, Issue 4:212-221, October - December 2016.

[23] Todreas, N.E. and Kazimi, M.S., *Nuclear Systems I, Thermal Hydraulic Fundamentals*, Hemisphere Publishing Corporation, 1990, ISBN 0-89116-935-0 (v. 1), 1990