UNIVERSITY OF BIRMINGHAM

Department of Metallurgy & Materials

Activity Manual

Contents

List of Figures

Chapter 1

Background

1.1 Ion Irradiation

A computer program, Activity, was developed to predict the activity and gamma lines of materials irradiated with an ion beam. It uses the TENDL[1] proton reaction cross section database, the Stopping and Range of Ions in Matter (SRIM)[2] code, a Nuclear Data Services (NDS) radioactive decay database [3] and an ENDF gamma decay database [4]. An extended version of Bateman's equation is used to calculate the activity at time t, and this equation is solved analytically, with the option to also solve by numeric inverse Laplace transform as a failsafe. The program outputs the expected activity and gamma lines of the activated material.

1.1.1 Ion Irradiation at the University of Birmingham

The Scanditronix MC-40 Cyclotron is used at the University of Birmingham to create a beam of protons or other light ions. The energies of these ions are typically between 10 MeV and 60 MeV with beam currents ranging up to 50 microamps (3.1×10^{14} protons per second). Target materials are irradiated by this cyclotron for a number of reasons, including purposely creating radioactive isotopes for the nearby Queen Elizabeth Hospital, investigating ion irradiation damage and emulating neutron irradiation.

The Cyclotron is usually used to create radioactive isotopes for medical use, but an additional beam line has been devoted to material science investigations into radiation damage. While the creation of radioactive isotopes is desired in some cases, material being tested for radiation damage should preferably have low levels of radioactivity.

It is expensive to arrange the irradiation of target materials by high energy neutrons sources, whereas it is relatively inexpensive to irradiate using an ion beam on the MC-40 Cyclotron. The energies can be controlled, and a set dose at a single energy, or a range of energies, can be precisely deposited in the target material.

The Activity code discussed here was developed to calculate the activity of a target material irradiated by a proton beam. It has been developed in Fortran and uses data from the TENDL-2013 proton cross section database, SRIM ion transport code and NDS radioactive decay database.

1.1.2 Simulating Ion Irradiation with SRIM

A package of ion transport codes, SRIM, is freely available to download and use to investigate the transport of ions through matter. SRIM uses the binary collision approximation (BCA) to simulate the passage of ions in a material. It is an approximate method, and one key restriction is that the does not take into account the structure of the material, and this approximation is therefore also imposed on the Activity code.

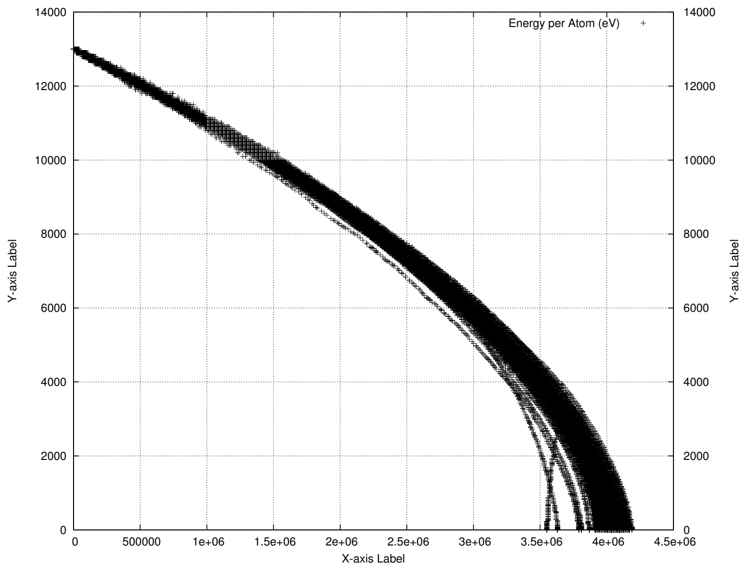

Figure 1.1: Proton energy loss in Fe simulated with SRIM [2]

One file that SRIM creates is of importance to the Activity code, and that is the trajectory file that contains the energy and x,y,z co-ordinate data points for simulated ions moving through matter. Figure 1.1 shows the trajectory of protons passing through an Iron target, and it is this set of data points (together with the cross section database) that the Activity code uses to calculte the reaction rates for the transmutation of nuclei in the target. At higher energies, the ions slow as they lose energy due to electronic stopping, but as the ion energy drops the mechanism of loss through nuclear collisions becomes important. The spreading of ion depths at lower energies is a result of the higher momentum transfer during nuclear collisions, as can be seen in Figure 1.1.

1.1.3 Transmutation of Nuclei by Ion Irradiation

Considering a simplified nuclear potential well, energetic protons approaching a nucleus may overcome the coulomb potential barrier. They are captured by the nucleus and held within the potential well by the strong nuclear force. This process may leave the nucleus in an excited and unstable state, depending on the input energy of the proton and configuration of nucleons. The process is probabilistic, and the average chance of a reaction (the microscopic cross section) may be measured as a function of the projectile, projectile energy and target, either experimentally or by optical model potential calculations. The reaction rate is calculated from the microscopic cross section using the following equation:

$$R = \frac{J}{e} n_t \sigma \cdot 10^{28} \delta t \tag{1.1}$$

- R Reaction Rate (reactions per second)
- J Beam current (A)
- n_t Number density of target (atoms per cubic metre)
- σ Microscopic reaction cross section (barns)
- e Elementary charge (1.602177E-19C)
- δT Target thickness (m)

1.2 Decay and Activity Equations

1.2.1 Radioactive Decay

Radioactive decay is the random change in nucleons or energy state of an unstable nucleus. It is impossible to predict when a nucleus will decay, but the decay of a collection of nuclei is statistical in nature. The radioactivity and number of unstable nuclei at time t can be predicted using the decay constant λfor the radioactive isotope. This constant is defined as follows:

$$\lambda = -\frac{N'(t)}{N(t)} \tag{1.2}$$

The number of radioactive nuclei N(t) at time t is given by the following equation, where N(0) is the starting number of nuclei:

$$N(t) = N(0)\exp(-t\lambda) \tag{1.3}$$

The activity A(t) of the radioactive nuclei is predicted at time t by using the following equations, where N'(t) is the change in amount of nuclei with respect to time:

$$A(t) = -N'(t) = \lambda N(t) \tag{1.4}$$

$$A(t) = \lambda N(0)\exp(-t\lambda) \tag{1.5}$$

1.2.2 Bateman Equation for Radioactive Decay

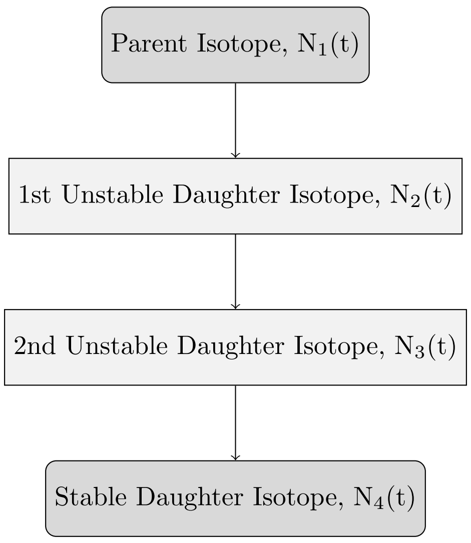
Parent Isotope, N_1(t)
1st Unstable Daughter Isotope, N_2(t)
2nd Unstable Daughter Isotope, N_3(t)
Stable Daughter Isotope, N_4(t)

Figure 1.2: An example decay chain from an unstable parent isotope, through unstable daughter isotopes ending with a stable daughter isotope.

The English mathematician Harry Bateman derived an equation to calculate the amount of each isotope in a decay chain, illustrated in Figure 1.2, at time t.

$$N_n(t) = \sum_{i=1}^{i=n}\left(\left(\prod_{j=i}^{j=n-1}\lambda_{(ij+1)}\right)\sum_{j=i}^{j=n}\left(\frac{N_{i0}\exp(-\lambda_j t)}{\prod_{p=i,p\neq j}^{p=n}(\lambda_p-\lambda_j)}\right)\right) \tag{1.6}$$

When a radioactive isotope decays, there may be more than one mode of decay, and this leads to branching factors. Pb-214 only decays via beta decay to Bi-214, giving a branching factor of 1.0, whereas Bi-214 has a 99.979% chance of decaying to Po-214 by beta decay and a 0.021% of emitting an alpha particle and decaying to Tl-210 (branching factors of 0.99979 and 0.00021 respectively) [5].

When a target material is irradiated, there is a source term for transmuted nuclei due to the irradiation. The daughter isotopes of these transmuted isotopes will also be affected by the irradiation and will transmute further, giving a source term for each daughter isotope as a result of the irradiation. Sources for each isotope in the decay chain, and branching factors between a parent isotope and its daughter isotope/s must be accounted for.

The Bateman equation was derived using Laplace transforms, and this same method has been used to develop a modified equation that incorporates branching factors and production rates for each isotope in the decay chain, as illustrated by Figure 1.3.

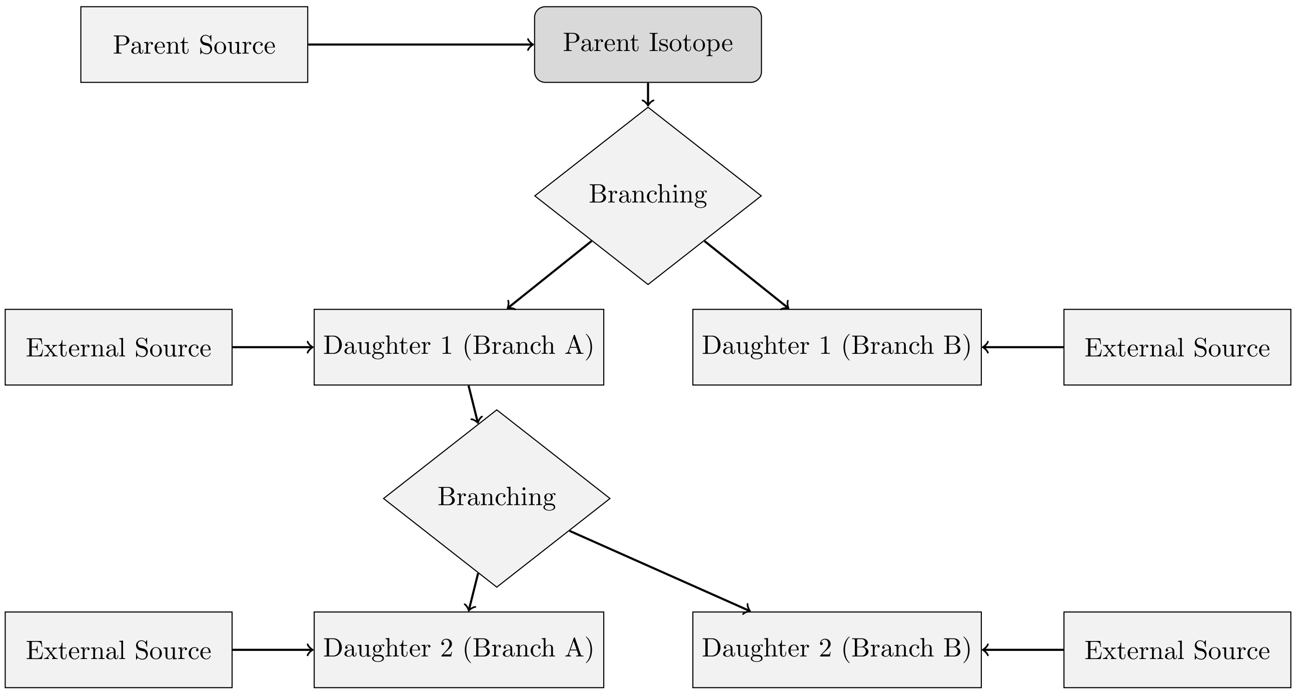

Figure 1.3: An example of several decay chains including branching factors and possible external source terms for each isotope on each chain.

1.2.3 Laplace Transform

Laplace Transforms are a useful mathematical tool, and allow ordinary differential equations to be solved by simple algebraic manipulation in the s domain. Bateman took advantage of Laplace Transforms in deriving his equation, and this is the method that has been taken here as well.

$$F(s) = \int_{0}^{\infty} f(t) \exp(-st) dt \tag{1.7}$$

1.2.4 Constructing the Differential Equations

The first step is to set up differential equations for the parent isotope, unstable daughter isotopes and stable daughter isotope. The parent isotope has a source term, due to production, and a loss term, due to decay. The unstable daughter isotopes have two source terms, from the production by irradiation induced transmutation and the decay of preceding isotopes in the decay chain, and a loss term, due to decay. Finally, the stable daughter that finalizes the decay chain has two source terms (the same as the unstable daughters) but no loss term.

The variables (and vectors) used in these equations are defined as follows:

- $\vec{\lambda}$ vector containing isotope decay constants λ_i
- $\vec{b}$ vector containing isotope to isotope branching factors b_i
- $\vec{w}$ vector containing isotope production rates w_i
- t time at which activity/amount of isotope is measured
- $N_i(0)$ starting amount of the ith isotope
- $N_i(t)$ amount of the ith isotope at time t
- $N_i'(t)$ change in amount of the ith isotope, with respect to time, at time t

The differential equations for the parent isotope (first isotope), unstable daughter isotopes (ith isotopes) and stable, final, daughter isotope (zth isotope) in the time domain are as follows:

$$N_1'(t) = \omega_1 - \lambda_1 N_1(t) \tag{1.8}$$

$$N_i'(t) = \omega_i + b_{i-1}\lambda_{i-1}N_{i-1}(t) - \lambda_i N_i(t) \tag{1.9}$$

$$N_z'(t) = \omega_z + b_{z-1}\lambda_{z-1}N_{z-1}(t) \tag{1.10}$$

Applying the Laplace Transform to these three differential equations allows them to be manipulated and solved algebraically in the s-domain.

$$N_1(s) = \frac{1}{s+\lambda_1}N_1(0) + \frac{1}{s(s+\lambda_1)}\omega_1 \tag{1.11}$$

$$N_i(s) = \frac{1}{s(s+\lambda_i)}(\omega_i) + \frac{1}{s+\lambda_i}(b_{i-1}\lambda_{i-1}N_{i-1}(s)) + \frac{1}{s+\lambda_i}N_i(0) \tag{1.12}$$

$$N_z(s) = \frac{1}{s^2}\omega_z + \frac{1}{s}b_{z-1}\lambda_{z-1}N_{z-1}(s) + \frac{1}{s}N_z(0) \tag{1.13}$$

1.2.5 Numerical Inversion of the Laplace Transform

The Gaver-Stehfest[6] algorithm was developed in the 1960s and 1970s and is a method of calculating the inverse of a Laplace Transform in the real number domain. It is an easy to implement and reasonably accurate method, although it is an approximation to the real value. A comparison between an analytic and numeric inversion for the unstable isotope Po-218 is discussed at the end of this section.

$$f(t) \approx f_n(t) = \frac{\ln(2)}{t}\sum_{k=1}^{2n} a_k(n)F(s) \text{ where } n \geq 1, t > 0 \tag{1.14}$$

$$s = \frac{k\ln(2)}{t} \tag{1.15}$$

$$a_k(n) = \frac{(-1)^{(n+k)}}{n!} \sum_{j=Floor(\frac{k+1}{2})} j^{n+1} \binom{n}{j} \binom{2j}{j} \binom{j}{k-j} \tag{1.16}$$

The equation for the ith isotope may be calculated by recursively calculating the equations by numeric inversion, starting from the first (parent isotope) and inserting the result into each subsequent recursion until the ith isotope is reached (changing the equations appropriately for the parent, unstable daugher and stable daughter isotopes).

1.2.6 Analytic Solution by Partial Fraction Expansion

The equation for the ith isotope in the s domain can be written in full by substituting the preceding equation until the parent isotope is reached, and this full eqaution may be rearranged with the production amount of each isotope and starting amount of each isotope in individual terms. Each of these terms is multiplied by a fraction that can be expanded, using partial fractions, and inverted analytically.

This is illustrated with an example unstable isotope, fourth in the decay chain (including the parent isotope):

$$\begin{aligned}
N_4(s) = \frac{1}{(s+\lambda_1)(s+\lambda_2)(s+\lambda_3)(s+\lambda_4)} b_2 b_3 b_4 \lambda_1 \lambda_2 \lambda_3 N_1(0) \\
+\frac{1}{(s+\lambda_2)(s+\lambda_3)(s+\lambda_4)} b_3 b_4 \lambda_2 \lambda_3 N_2(0) \\
+\frac{1}{(s+\lambda_3)(s+\lambda_4)} b_4 \lambda_3 N_3(0) \\
+\frac{1}{(s+\lambda_4)} N_4(0) \\
+\frac{1}{s(s+\lambda_1)(s+\lambda_2)(s+\lambda_3)(s+\lambda_4)} b_2 b_3 b_4 \lambda_1 \lambda_2 \lambda_3 \omega_1 \\
+\frac{1}{s(s+\lambda_2)(s+\lambda_3)(s+\lambda_4)} b_3 b_4 \lambda_2 \lambda_3 \omega_2 \\
+\frac{1}{s(s+\lambda_3)(s+\lambda_4)} b_4 \lambda_3 \omega_3 \\
+\frac{1}{s(s+\lambda_4)} \omega_4
\end{aligned} \tag{1.17}$$

An example stable isotope, fourth (last) in the decay chain (including the parent isotope):

$$\begin{aligned}N_4(s) = {} & \frac{1}{s(s+\lambda_1)(s+\lambda_2)(s+\lambda_3)} b_2 b_3 b_4 \lambda_1 \lambda_2 \lambda_3 N_1(0) \\ & + \frac{1}{s(s+\lambda_2)(s+\lambda_3)} b_3 b_4 \lambda_2 \lambda_3 N_2(0) \\ & + \frac{1}{s(s+\lambda_3)} b_4 \lambda_3 N_3(0) \\ & + N_4(0) \\ & + \frac{1}{s^2(s+\lambda_1)(s+\lambda_2)(s+\lambda_3)} b_2 b_3 b_4 \lambda_1 \lambda_2 \lambda_3 \omega_1 \\ & + \frac{1}{s^2(s+\lambda_2)(s+\lambda_3)} b_3 b_4 \lambda_2 \lambda_3 \omega_2 \\ & + \frac{1}{s^2(s+\lambda_3)} b_4 \lambda_3 \omega_3 \\ & + \frac{1}{s^2} \omega_4\end{aligned} \tag{1.18}$$

By using partial fraction expansion and standard Laplace Transforms, the set of equations below is used to calculate the amount of the mth isotope in the decay chain, providing the mth isotope is unstable.

$$N_m(t; \vec{\lambda}, \vec{b}, \vec{w}) = \sum_{k=1,m} r(k; \vec{\lambda}, \vec{b}) \left[f(t; k, m, \vec{\lambda}) N_k(0) + g(t; k, m, \vec{\lambda}) w_k \right] \tag{1.19}$$

$$r(k, m, \vec{\lambda}) = \begin{cases} \prod_{i=k,m-1} (b_{i+1}\lambda_i), & \text{if } k < m \\ 1, & \text{if } k = m \end{cases} \tag{1.20}$$

$$f(t; k, m, \vec{\lambda}) = (-1)^{m-k} \sum_{i=k,m} \left[\exp(-\lambda_i t) \prod_{j=k,m; j\neq i} \left(\frac{1}{\lambda_i - \lambda_j} \right) \right] \tag{1.21}$$

$$g(t; k, m, \vec{\lambda}) = \frac{1}{\prod_{i=k,m} \lambda_i} + (-1)^{m-k+1} \sum_{i=k,m} \left[\frac{1}{\lambda_i} \exp(-\lambda_i t) \prod_{j=k,m; j\neq i} \left(\frac{1}{\lambda_i - \lambda_j} \right) \right] \tag{1.22}$$

The set of equations below is used to calculate the amount of the mth isotope in the decay chain, where the mth isotope is stable.

$$N_m(t; \vec{\lambda}, \vec{b}, \vec{w}) = N_m + w_m t + \sum_{k=1,m-1} r(k; \vec{\lambda}, \vec{b}) \left[f(t; k, m-1, \vec{\lambda}) N_k(0) + g(t; k, m, \vec{\lambda}) w_k \right] \tag{1.23}$$

$$r(k, m, \vec{\lambda}) = \begin{cases} \prod_{i=k,m-1} (b_{i+1}\lambda_i), & \text{if } k < m \\ 1, & \text{if } k = m \end{cases} \tag{1.24}$$

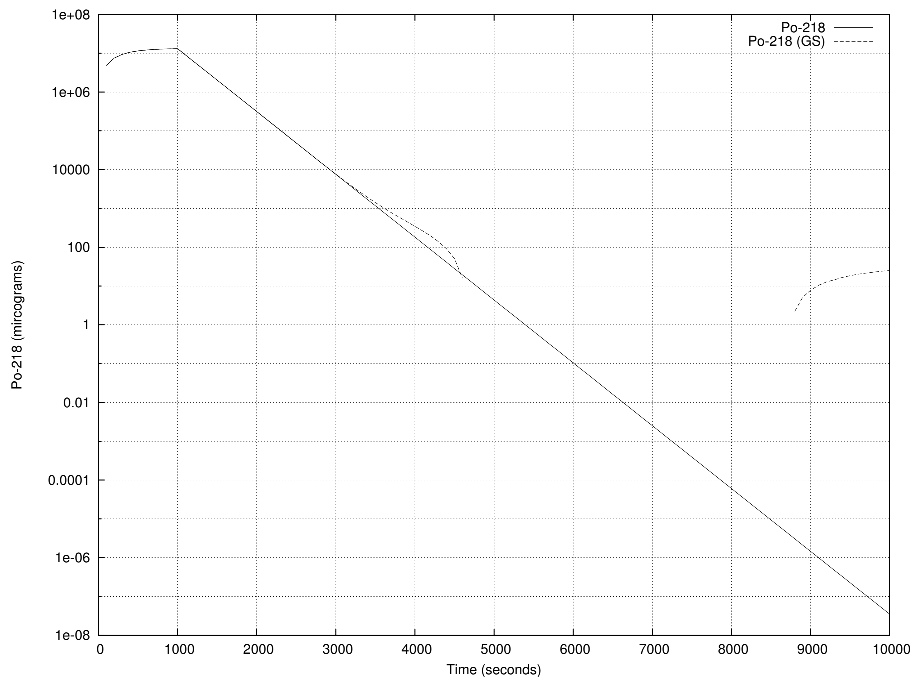

Figure 1.4: Decay of Po-218: Analytice and Gaver-Stehfest Calculations [5]

$$f(t;k,m,\vec{\lambda}) = \frac{1}{\prod_{i=k,m}\lambda_i} + (-1)^{m-k+1}\sum_{i=k,m}\left[\frac{1}{\lambda_i}\exp(-\lambda_i t)\prod_{j=k,m;j\neq i}\left(\frac{1}{\lambda_i-\lambda_j}\right)\right] \tag{1.25}$$

$$g(t;k,m,\vec{\lambda}) = \frac{1}{\prod_{i=k,m}\lambda_i}t + \frac{\sum_{i=k,m}\left[\prod_{j=k,m;j\neq i}\lambda_j\right]}{\prod_{i=k,m}\lambda_i^2} + (-1)^{m-k+1}\sum_{i=k,m}\left[\frac{1}{\lambda_i^2}\exp(-\lambda_i t)\prod_{j=k,m;j\neq i}\left(\frac{1}{\lambda_i-\lambda_j}\right)\right] \tag{1.26}$$

1.2.7 Preference: Analytic over Numeric

The numeric solution only requires the equation to be solved in the s-domain; the Gaver-Stehfest algorithm performs the inversion. It is worth the extra effort to derive and implement an analytic solution, as the numeric is only an approximation. Examples of the pitfalls of the numeric solution are that it can give negative amounts of an isotope and the difference between the numeric and analytic calculated amounts can become quite large when the isotope decays away to a very small value. Figure 1.4 shows the predicted decay of a sample of Po-218 irradiated for 1,000s, and sampled until 10,000s. In the region between 4,000s and 9,000s the amount from the numeric calculation drops below zero, whereas the analytic calculation remains above zero, as would be expected.

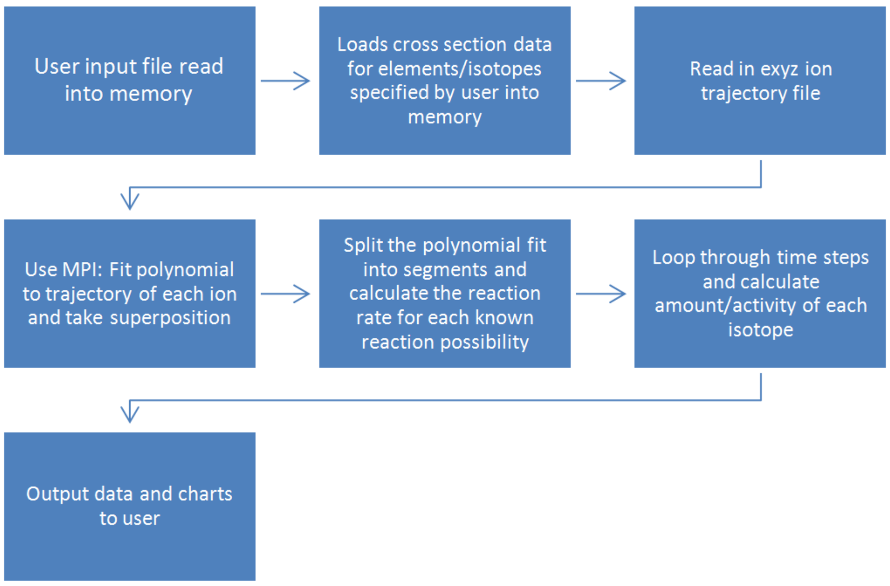
User input file read into memory
Loads cross section data for elements/isotopes specified by user into memory
Read in exyz ion trajectory file
Use MPI: Fit polynomial to trajectory of each ion and take superposition
Split the polynomial fit into segments and calculate the reaction rate for each known reaction possibility
Loop through time steps and calculate amount/activity of each isotope
Output data and charts to user

Figure 1.5: Flow chart of major processes in the Activity code

1.3 Computational Methods

1.3.1 Activity Code

The Activity program has been developed in Fortran and takes advantage of MPI (Message Parsing Interface) to speed up calculation times by allowing the use of multiple processes in parallel. It has a self contained maths library, although this could be improved in the future by using optimised maths libraries for certain functions (e.g. matrix operations).

The code was developed on a Debian version of Linux, but it should be supported on other variants of Linux and Unix, and does not require any specialist hardware.

The user is required to prepare an input file that contains the instructions required to perform a calculation. In addition to the input file, the user must provide an EXYZ ion trajectory file output by SRIM. Activity will read in the user input file, and the SRIM and data files listed within, before performing the calculation. Figure 1.5 shows a flowchart of the major steps the code performs.

There are various settings in the user input file, but the main ones relating to the simulated experiment are:

- element composition of target (percentage by mass)
- beam flux (current), energy, duration and area on target
- activity measurement time (end of the "experiment")
- material density
- target thickness

Several data files are generated by Activity and, if the user has matplotlib [7], charts will be created too. The most relevant to the user are:

- gammaLines.dat - tally into discrete bins of predicted gamma counts

- ionTraj.dat - the averaged ion trajectory used in the calculation
- isotopeActivityFileG.dat - a large data file detailing the activity of every predicted radioactive isotope in the target at user specified times following irradioation

The charts include:

- activityTop5.png - activity of the top 5 active isotopes as a function of time after irradiation starts
- gammaLines.png - predicted gamma spectrum expected at the "experiment end time"

The Activity code uses the equations derived above to calculate the amount and activity of each isotope in the calculation. One problem with the original Bateman equation that also exists in the set of modified Bateman equations is that two different isotopes with the same decay constant will cause a singularity and a halt in the calculation. The activity code loops through all the decay constants in use before it attempts to run the calculation. If any isotope decay constants match they are varied by a small amount relative to the decay constant. It repeats this process until all decay constants are unique before proceding.

1.3.2 Approximations

The accuracy of the Activity code is dependant on the input files provided by the user and the method used to calculate the reaction rates and resulting activity. The TENDL proton database consists of experimental measured cross sections as well as values calculated using the optical model potential. Using the latest database is recommended.

SRIM uses the binary collision approximation to simulate ion transport. It is a well tested code that has been used for many years. One limitation is that the structure of the material is not taken into account. This would have an impact on a user of the Activity program if they were trying to calculate, for example, whether a FCC (face centered cubic) steel would be irradiated differently when compared to a BCC (body centered cubic) steel. The Activity code would determine the activity of the steel as a function of the ion current, ion type and the density, thickness and composition of the steel, not its structure.

This version of the Activity code averages the path of all the SRIM simulated ions, rather than treating each ion differently. This may or may not have an impact on the results. If a new version of the code is developed there would be an option to calculate reaction rates for each induvidual simulated ion, and a comparison could then be made to the calculations using the averaged path of a set of ions.

The final approximation would be to use the numeric solution to the activity equations, although the analytic solution is forced within the code unless it returns a failed result.

1.3.3 Results

A target of high purity Iron was irradiated with 36 MeV protons by the University of Birmingham Scanditronix MC-40 Cyclotron. The target was 0.5mm thick and was irradiated at a current of 0.5 micro Amps for 300 seconds, irradiating approximately 0.25g of Iron. A high purity Germanium detector was used to measure the gamma peaks three days after irradiation.

The peak that dominated the readings was the 931 keV Cobalt 55 line. After calibrating the detector and adjusting the readings, this peak was measured at 44,300Bq+/-2,000Bq. The activity of this peak as predicted by the Activity code was 44,565Bq.

A target of high purity Molybdenum was irradiated with 13 MeV protons by the University of Birmingham Scanditronix MC-40 Cyclotron. The target was 0.5mm thick and was irradiated at a current of 5 micro Amps

Table 1.1: Gamma peaks predicted and measured for a 13 MeV ion irradiated sample of Mo

Gamma Energy (keV)	Predicted Activity (Bq)	Experimental Activity (Bq)
766	4.45E6	5.11E5+/-2.5E4
778	6.14E6	1.36E6+/-6.8E4
812	5.04E6	1.15E6+/-5.8E4
850	6.00E6	1.39E6+/-7.0E4
126	9.33E5	2.10E5+/-1.1E4

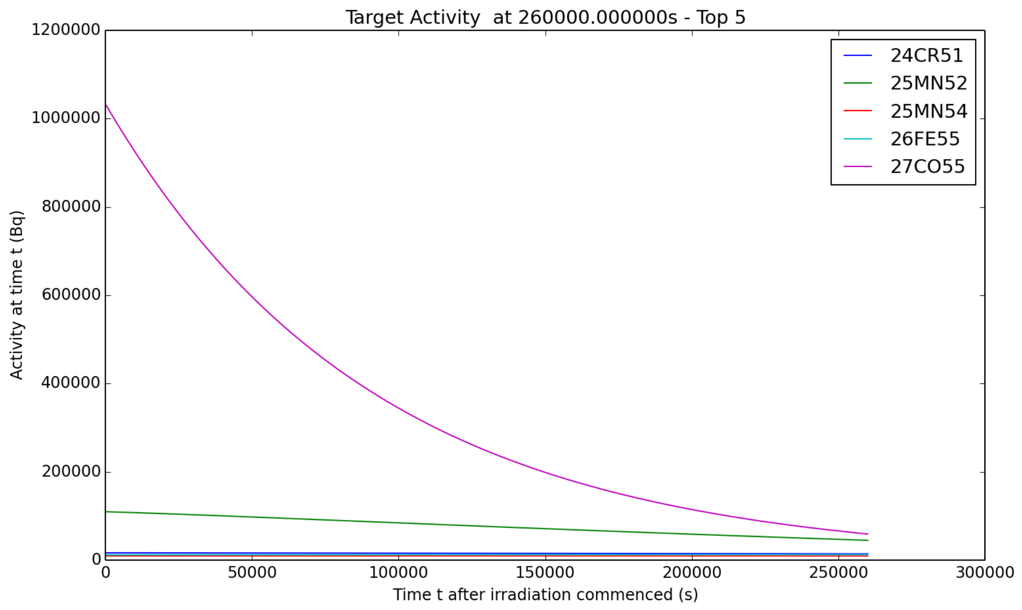

Figure 1.6: Sample Activity code output chart for the top five most active isotopes for Iron irradiated by 36MeV protons.

for 1500 seconds, irradiating approximately 0.3g of Molybdenum. A high purity Germanium detector was used to measure the gamma peaks three days after irradiation.

Five peaks were of interest, and these are listed in Table 1.1.

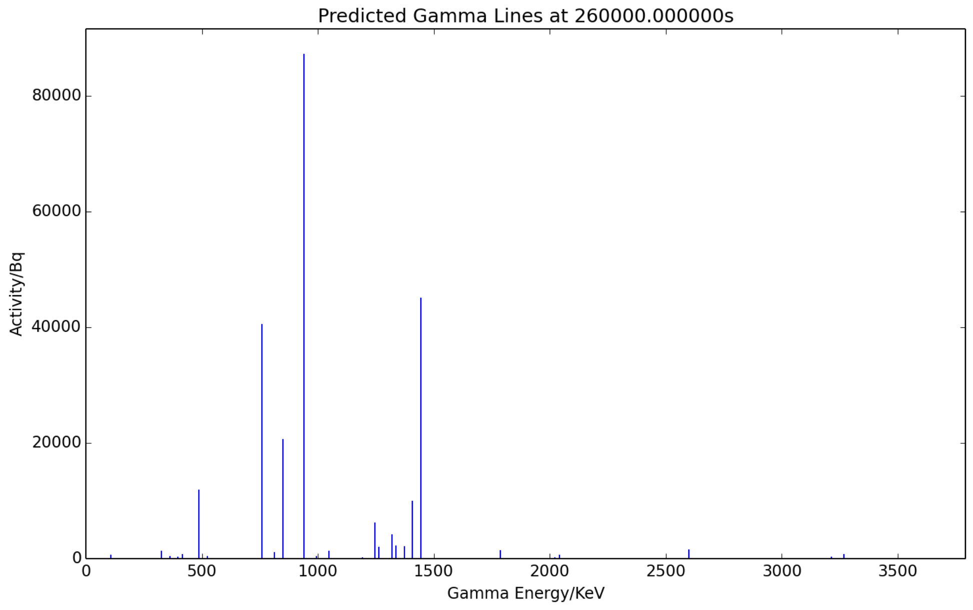

Figure 1.7: Sample Activity code output chart for the expected gamma lines to be measured for Iron irradiated by 36MeV protons.

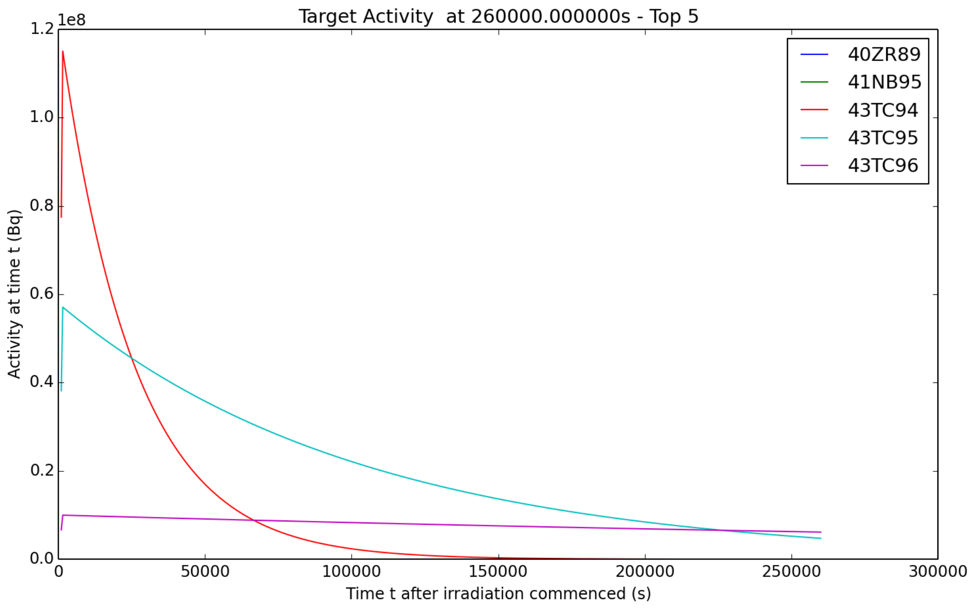

Figure 1.8: Sample Activity code output chart for the top five most active isotopes for Molybdenum irradiated by 13MeV protons.

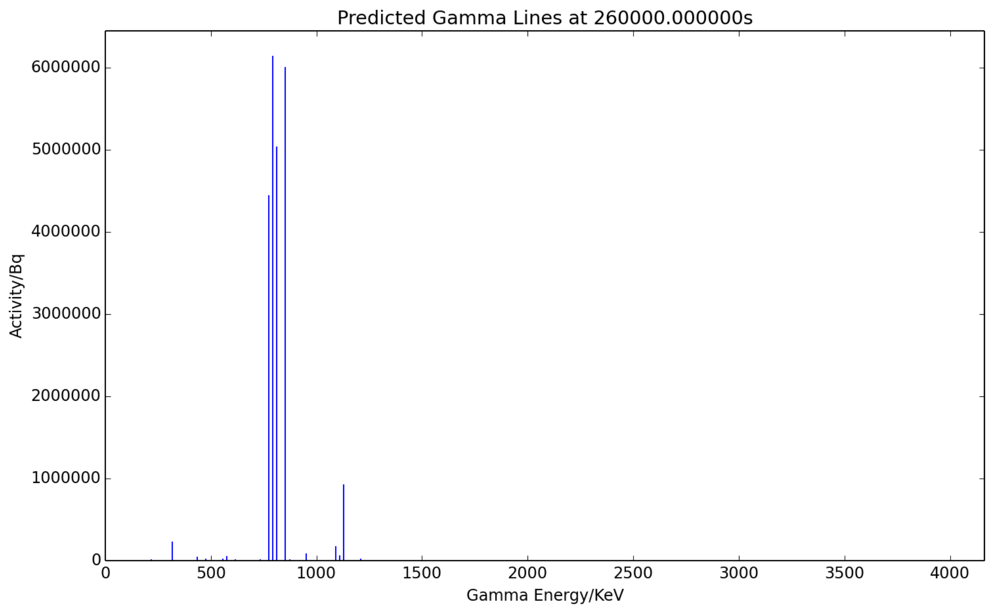

Figure 1.9: Sample Activity code output chart for the expected gamma lines to be measured for Molybdenum irradiated by 13MeV protons.

Chapter 2

Installation and Using the Activity Code

2.1 Getting Started

2.1.1 Prerequisites

This code was designed to run on Linux and has been developed and tested on the Debian based distribution Mint. It has not been tested on any other versions of Linux or Unix, nor has it been tested on cygwin.

The minimal requirements of the code are:

- 250MB per process + 200MB overhead
- 1GB Hard Drive space
- Ideally a modern multicore processor
- Fortran90
- OpenMPI

Optional:

- GNUPlot
- Python & MatPlotLib

2.2 Installing Activity

2.2.1 Download Source Code

The most recent source code is available for download from github. It may be downloaded and extracted using the terminal and the commands in listing 1.

Listing 2.1: bash

```bash
cd ~
mkdir -p ~/activity/tar
cd ~/activity/tar
wget https://github.com/BenPalmer1983/activity/raw/master/activity.tar.gz
tar -xzvf activity.tar.gz
```

2.2.2 Compile Source Code

Once the source files have been downloaded and extracted, run the install script then remove the downloaded files (if you need to free up space). The terminal commands are given in listing 2. Providing the prerequisites are available, the process will only take a few minutes (with a reasonable Internet connection).

Listing 2.2: bash

```bash
cd ~/activity/tar
./install.sh #choose installation directory when prompted; default is ~/activity
rm -R ~/activity/tar
```

The activity code is now installed and the activity.x binary may be executed on the terminal. If the terminal fails to find the binary, the .bashrc or .bash_profile file must be edited, or a symlink created from the bin file to a valid and accessible bin directory.

2.3 Input File

The input file sets the target starting composition, the beam and simulated experiment settings as well as the paths to the required data files.

#elements

The elements keyword instructs the code to read in the composition of the target material. The elements and their percentage by weight are listed under the keyword.

Listing 2.3: Input File

```
#elements
Fe 72
Cr 18
Ni 8
Mg 2
```

#isotopes #decaymodes #gammaenergies #xsfiles

Four sets of data files are distributed with the code, and the paths to these files are defined underneath each keyword. N.B. the path for the xsfiles is to the directory that holds all the cross section data files, rather than each individual file.

Listing 2.4: Input File

```
#isotopes
"/home/ben/activity/data/isotopes.txt"
#decaymodes
"/home/ben/activity/data/decaymodes.txt"
```

```
#gammaenergies
"/home/ben/activity/data/gammaenergies.txt"
#xsfiles
"/home/ben/activity/data/xs"
```

#trajfile

The SRIM exyz file used for the calculation is pointed to under this keyword.

Listing 2.5: Input File

```
#trajfile
"/home/ben/activity/examples/Fe36MeV.exyz"
```

#polyfitorder #integrationgranularity

The code calculates the reaction rate of each projectile and target nucleus by first fitting a polynomial to the energy/depth data file. This polynomial gives an average path for the projectiles travelling through the target, allowing E(x) to be calculated quickly. The order of the polynomial is selected here, and a 5th order shold give a reasonable fit. The integration granularity keyword determines how many sections the polynomial is split up into. The energy, cross section for each section and the beam settings are used to calculate the reaction rate for each section, and these are all summed up to give the overall reaction rate.

Listing 2.6: Input File

```
#polyfitorder
5
#integrationgranularity
10
```

#beamflux #beamenergy #beamduration #beamarea

These keywords are self explanatory and control the beam settings used in the calculation.

Listing 2.7: Input File

```
#beamflux
0.5 uA
#beamenergy
36 MeV
#beamduration
300 s
#beamarea
100 mm2
```

#amtime #timestep

The time at which the activity of the sample is measured is set by the #amtime keyword. The Activity code calculates the activity of each radioactive isotope in the calculation from the time the beam starts until the äctivity measured time; and the intervals at which these calculations are made is determined by the #timestep keyword.

Listing 2.8: Input File

```
#amtime
260000 s
```

```
#timestep
1000 s
```

#projectile

The current version only supports protons as the projectile, therefore the only possible atomic and mass number combination for a projectile is 1 1.

Listing 2.9: Input File

```
#projectile
1 1
```

#targetthickness #materialdensity

Both the target thickness and density of the material it is made from are input with these keywords.

Listing 2.10: Input File

```
#targetthickness
0.5 mm
#materialdensity
8000 kgm3
```

#vpi

This keyword is only being used for a feature currently under testing, so the keyword can be ommited from the input file.

Listing 2.11: Input File

```
#vpi
60.2
```

#individualisotopeactivity #verboseterminal

If the #individualisotopeactivity keyword is followed by ÿes", additional data files are output containing the activity of each individual isotope in the calulcation. #verboseterminal followed by ÿes"will output more verbosely to the terminal.

Listing 2.12: Input File

```
#individualisotopeactivity
yes
#verboseterminal
yes
```

#targetdpa

This keyword is only being used for a feature currently under testing, so the keyword can be ommited from the input file.

Listing 2.13: Input File

```
#targetdpa
0.0
```

#gammachartresolution

The gamma chart is created by tallying the gamma values into bins, and this figure specifies the resolution(number of bins) used to create the output chart.

Listing 2.14: Input File

```
#gammachartresolution
200
```

2.4 Acknowledgements

We would like to thank and acknowledge the input and advice of the following:

- Dr Chris Cooper and John Hewett for irradiation activation data points.
- The University of Birmingham for providing the funding for this project.

Appendices

Appendix A

Example Input File

A.1 Iron 36MeV Proton Beam

Listing A.1: Fe36MeV.in

```
#elements
Fe 100
#isotopes
"/home/ben/activity/data/isotopes.txt"
#decaymodes
"/home/ben/activity/data/decaymodes.txt"
#gammaenergies
"/home/ben/activity/data/gammaenergies.txt"
#xsfiles
"/home/ben/activity/data/xs"
#trajfile
"/home/ben/activity/examples/Fe36MeV.exyz"
#polyfitorder
5
#integrationgranularity
10
#beamflux
0.5 uA
#beamenergy
36 MeV
#beamduration
300 s
#beamarea
100 mm2
#amtime
260000 s
#timestep
1000 s
#projectile
1 1
#targetthickness
0.5 mm
#materialdensity
8000 kgm3
```

```
#vpi
60.2
#individualisotopeactivity
yes
#verboseterminal
yes
#targetdpa
0.0
#gammachartresolution
200
```

Appendix B

Fortran 90 Code

B.1 Fortran 90 Implementation of Analytic Method

Listing B.1: Fortran 90

```fortran
  Type :: decayChainObj
    Real(kind=DoubleReal) :: time = 0.0D0
    !Real(kind=DoubleReal) :: productionRate = 0.0D0
    Integer(kind=StandardInteger) :: isotopes
    Character(Len=16), Dimension(1:100) :: label
    Real(kind=DoubleReal), Dimension(1:100) :: productionRate = 0.0D0
    Real(kind=DoubleReal), Dimension(1:100) :: branchFactor = 1.0D0 ! from isotope parent
    Real(kind=DoubleReal), Dimension(1:100) :: decayConstant = -1.0D0 ! negative for stable
    Real(kind=DoubleReal), Dimension(1:100) :: halfLife = -1.0D0 ! negative for stable
    Real(kind=DoubleReal), Dimension(1:100) :: amountStart = 0.0D0
    Real(kind=DoubleReal), Dimension(1:100) :: amountEnd = 0.0D0
  End Type

  Subroutine CalcIsotopeChain(decayChain)
! Uses inverse laplace transform to calculate isotope amounts at time t (after time = 0)
! t time in seconds after t=0
! w production rate of parent isotope
! isotope chain data
    Implicit None ! Force declaration of all variables
! Vars In/Out
    Type(decayChainObj) :: decayChain
! Vars Private
    Integer(kind=StandardInteger) :: i
    Real(kind=DoubleReal) :: t, nEnd
    Real(kind=DoubleReal), Dimension(1:100) :: W ! Production Rate
    Real(kind=DoubleReal), Dimension(1:100) :: L ! Lambda
    Real(kind=DoubleReal), Dimension(1:100) :: N ! Starting number of atoms
    Real(kind=DoubleReal), Dimension(1:100) :: B ! Exp
! Complete decay chain data
    Call decayChainComplete(decayChain)
! store input in shortned name arrays to make equations clearer
    t = decayChain%time
    Do i=1,decayChain%isotopes
      W(i) = decayChain%productionRate(i)
      L(i) = decayChain%decayConstant(i)
      B(i) = decayChain%branchFactor(i)
      N(i) = decayChain%amountStart(i)
    End Do
! Break infinities
    Call DecayBreakInfinities(L,decayChain%isotopes)
! Run analytic calculations
! Loop through isotopes
! Using L-1(1/(q+ps) = (1/p)*exp(-1*(q/p)*t) and partial fractions
    Do i=1,decayChain%isotopes
      nEnd = CalcIsotopeChainCalc(t,i,W,L,N,B)
      decayChain%amountEnd(i) = nEnd
    End Do
  End Subroutine CalcIsotopeChain

  Subroutine decayChainComplete(decayChain)
! Completes the decay chain object
    Implicit None ! Force declaration of all variables
! Vars In/Out
    Type(decayChainObj) :: decayChain
! Vars Private
    Integer(kind=StandardInteger) :: i, n
    n = 0
    Do i=1,100
      n = n + 1
      If(decayChain%decayConstant(i).eq.-1.0D0.and.decayChain%halfLife(i).gt.0.0D0)Then
! complete decay constant from half life
        decayChain%decayConstant(i) = lnTwo/decayChain%halfLife(i)
      End If
      If(decayChain%halfLife(i).le.0.0D0.and.decayChain%decayConstant(i).gt.0.0D0)Then
! complete decay constant from half life
        decayChain%halfLife(i) = lnTwo/decayChain%decayConstant(i)
      End If
! Adjust for stable isotope
      If(decayChain%decayConstant(i).lt.0.0D0)Then
        decayChain%halfLife(i) = -1.0D0
        decayChain%decayConstant(i) = 0.0D0
      End If
      If(decayChain%halfLife(i).lt.0.0D0)Then
        decayChain%halfLife(i) = -1.0D0
        decayChain%decayConstant(i) = 0.0D0
      End If
! Break out if stable
      If(decayChain%decayConstant(i).eq.0.0D0)Then
        Exit
      End If
    End Do
    decayChain%isotopes = n
  End Subroutine decayChainComplete

  Subroutine DecayBreakInfinities(L,n)
!
    Implicit None ! Force declaration of all variables
```

```fortran
! Vars In/Out
    Real(kind=DoubleReal), Dimension(:) :: L ! Lambda
    Integer(kind=StandardInteger) :: n
! Vars Private
    Integer(kind=StandardInteger) :: i,j
    Logical :: breaking
! Loop and alter matching decay constants slightly
    breaking = .true.
    Do While(breaking)
      breaking = .false.
      Do i=1,n-1
        Do j=i+1,n
          If(L(i).eq.L(j))Then
            breaking = .true.
            L(i) = L(i)*1.0000001D0 ! Vary by 0.00001%
          End If
        End Do
      End Do
    End Do
  End Subroutine DecayBreakInfinities

  Function CalcIsotopeChainCalc(t,m,W,L,N,B) Result (nEnd)
    Implicit None ! Force declaration of all variables
! Vars In
    Real(kind=DoubleReal) :: t
    Integer(kind=StandardInteger) :: m
    Real(kind=DoubleReal), Dimension(1:100) :: W ! Production Rate
    Real(kind=DoubleReal), Dimension(1:100) :: L ! Lambda
    Real(kind=DoubleReal), Dimension(1:100) :: N ! Starting number of atoms
    Real(kind=DoubleReal), Dimension(1:100) :: B ! Branch factor
! Vars Out
    Real(kind=DoubleReal) :: nEnd
! Vars Private
    Integer(kind=StandardInteger) :: i, j, k, z
    Real(kind=DoubleReal) :: mult, multR
    Real(kind=DoubleReal) :: nChange
! Init
    nEnd = 0.0D0
! ---------------------------------------------
! UNSTABLE Isotopes
! ---------------------------------------------
    If(L(m).gt.0.0D0)Then
! Loop through terms
      Do k=1,m
        multR = CalcIsotopeChainMultR(k,m,L,B)
        mult = multR * N(k)
! decay of starting matter
        nChange = CalcIsotopeChainF_Unstable(k,m,t,mult,L)
        nEnd = nEnd + nChange
! production term
        mult = multR * W(k)
        nChange = CalcIsotopeChainG_Unstable(k,m,t,mult,L)
        nEnd = nEnd + nChange
        print *,k,nEnd
      End Do
    Else
! ---------------------------------------------
! STABLE Isotopes
! ---------------------------------------------
! Loop through terms
      nEnd = N(m)+t*W(m)
      Do k=1,m-1
        multR = CalcIsotopeChainMultR(k,m,L,B)
        mult = multR * N(k)
! decay of starting matter
        nChange = CalcIsotopeChainF_Stable(k,m,t,mult,L)
        nEnd = nEnd + nChange
! production term
        mult = multR * W(k)
        nChange = CalcIsotopeChainG_Stable(k,m,t,mult,L)
        nEnd = nEnd + nChange
        print *,k,nEnd
      End Do
    End If
  End Function CalcIsotopeChainCalc

  Function CalcIsotopeChainMultR(k,m,L,B) Result (multR)
! Vars In
    Integer(kind=StandardInteger) :: k, m
    Real(kind=DoubleReal), Dimension(1:100) :: L ! Lambda
    Real(kind=DoubleReal), Dimension(1:100) :: B ! Branching Factor
! Vars Out
    Real(kind=DoubleReal) :: multR
! Private
    Integer(kind=StandardInteger) :: i
! Result
    multR = 1.0D0
    If(k.lt.m)Then
      Do i=k,m-1
        multR = multR * B(i+1) * L(i)
      End Do
    End If
  End Function CalcIsotopeChainMultR

! --------------------------
! Unstable Isotope Functions
! --------------------------

  Function CalcIsotopeChainF_Unstable(k,m,t,mult,L) Result (nChange)
! Vars In
    Integer(kind=StandardInteger) :: k, m
    Real(kind=DoubleReal) :: t, mult
    Real(kind=DoubleReal), Dimension(1:100) :: L ! Lambda
! Vars Out
    Real(kind=DoubleReal) :: nChange
! Private
```

```
    Real(kind=DoubleReal) :: multP
    Real(kind=DoubleReal) :: p, q, r, s
    Integer(kind=StandardInteger) :: i, j
! Calculate isotope amount change
    nChange = 0.0D0
    multP = (-1.0D0)**(m-k)
! Loop through pfrac
    Do i=k,m
      r = 1.0D0
      Do j=k,m
        If(j.ne.i)Then
          r = r * (L(i)-L(j))
        End If
      End Do
      nChange = nChange + (1.0D0/r)*exp(-1.0D0*L(i)*t)*multP*mult
    End Do
  End Function CalcIsotopeChainF_Unstable

  Function CalcIsotopeChainG_Unstable(k,m,t,mult,L) Result (nChange)
! Vars In
    Integer(kind=StandardInteger) :: k, m
    Real(kind=DoubleReal) :: t, mult
    Real(kind=DoubleReal), Dimension(1:100) :: L ! Lambda
! Vars Out
    Real(kind=DoubleReal) :: nChange
! Private
    Real(kind=DoubleReal) :: multP
    Real(kind=DoubleReal) :: p, q, r, s
    Integer(kind=StandardInteger) :: i, j
! Calculate isotope amount change
    nChange = 0.0D0
    multP = (-1.0D0)**(m-k+1)
  ! term A
    r = 1.0D0
    Do i=k,m
      r = r * L(i)
    End Do
    nChange = nChange + (1.0D0/r)*mult
! Loop through pfrac
    Do i=k,m
      r = 1.0D0*L(i)
      Do j=k,m
        If(j.ne.i)Then
          r = r * (L(i)-L(j))
        End If
      End Do
      nChange = nChange + (1.0D0/r)*exp(-1.0D0*L(i)*t)*multP*mult
    End Do
  End Function CalcIsotopeChainG_Unstable

! --------------------------
! Stable Isotope Functions
! --------------------------
```

```fortran

  Function CalcIsotopeChainF_Stable(k,mIn,t,mult,L) Result (nChange)
! Vars In
    Integer(kind=StandardInteger) :: k, mIn
    Real(kind=DoubleReal) :: t, mult
    Real(kind=DoubleReal), Dimension(1:100) :: L ! Lambda
! Vars Out
    Real(kind=DoubleReal) :: nChange
! Private
    Integer(kind=StandardInteger) :: m
    Real(kind=DoubleReal) :: multP
    Real(kind=DoubleReal) :: p, q, r, s
    Integer(kind=StandardInteger) :: i, j
! In
    m = mIn-1
! Calculate isotope amount change
    nChange = 0.0D0
    multP = (-1.0D0)**(m-k+1)
! term A
    r = 1.0D0
    Do i=k,m
      r = r * L(i)
    End Do
    nChange = nChange + (1.0D0/r)*mult
! Loop through pfrac
    Do i=k,m
      r = 1.0D0*L(i)
      Do j=k,m
        If(j.ne.i)Then
          r = r * (L(i)-L(j))
        End If
      End Do
      nChange = nChange + (1.0D0/r)*exp(-1.0D0*L(i)*t)*multP*mult
    End Do
  End Function CalcIsotopeChainF_Stable


  Function CalcIsotopeChainG_Stable(k,mIn,t,mult,L) Result (nChange)
  ! Vars In
      Integer(kind=StandardInteger) :: k, mIn
      Real(kind=DoubleReal) :: t, mult
      Real(kind=DoubleReal), Dimension(1:100) :: L ! Lambda
  ! Vars Out
      Real(kind=DoubleReal) :: nChange
  ! Private
      Integer(kind=StandardInteger) :: m
      Real(kind=DoubleReal) :: multP
      Real(kind=DoubleReal) :: p, q, r, s
      Integer(kind=StandardInteger) :: i, j
  ! In
      m = mIn-1
  ! Calculate isotope amount change
      nChange = 0.0D0
      multP = (-1.0D0)**(m-k)
  ! term A
      r = 1.0D0
      Do i=k,m
        r = r * L(i)
      End Do
      nChange = nChange + (1.0D0/r)*t*mult
  ! term B
      p = CalcIsotopeChainC(L,k,m)
      q = 1.0D0
      Do i=k,m
        q = q*L(i)*L(i)
      End Do
      r = (-1.0D0)*(p/q)
      nChange = nChange + r*mult
  ! Loop through pfrac
      Do i=k,m
        r = 1.0D0*L(i)*L(i)
        Do j=k,m
          If(j.ne.i)Then
            r = r * (L(i)-L(j))
          End If
        End Do
        nChange = nChange + (1.0D0/r)*exp(-1.0D0*L(i)*t)*multP*mult
      End Do
  End Function CalcIsotopeChainG_Stable

  Function CalcIsotopeChainC(L,k,m) Result (numerator)
! Calculates numerator in isotope activity function
    Implicit None ! Force declaration of all variables
! Vars In
    Real(kind=DoubleReal), Dimension(:) :: L
    Integer(kind=StandardInteger) :: k, m
! Vars Out
    Real(kind=DoubleReal) :: numerator
! Vars Private
    Integer(kind=StandardInteger) :: i, j
    Real(kind=DoubleReal) :: tempMult
    numerator = 0.0D0
    Do i=k,m
      tempMult = 1.0D0
      Do j=k,m
        If(j.ne.i)Then
          tempMult = tempMult * L(j)
        End If
      End Do
      numerator = numerator + tempMult
    End Do
  End Function CalcIsotopeChainC
```

Bibliography

[1] A. J. Koning D. Rochman. “Modern Nuclear Data Evaluation With The TALYS Code System”. In: *Nuclear Data Sheets* 113 (2012).

[2] J. P. Biersack James F. Ziegler M. D. Ziegler. “SRIM - The stopping and range of ions in matter”. In: *Nuclear Instruments and Methods in Physics Research Section B* 268 (2010), pp. 1818–1823.

[3] A. A. Sonzogni. *Nuclear Data Services. www-nds.iaea.org.* 2006. URL: `http://www-nds.iaea.org/ndspub/download-endf/ENDF-B-VII.0/decay-index.htm` (visited on 08/14/2015).

[4] M. Herman M. B. Chadwick. “ENDF/B-VII.0: Next generation evaluated nuclear data library for nuclear science and technology.” In: *Nucl. Data Sheets* 107 (2006), p. 2931.

[5] P. Blaise M. Coste A. Courcelle T.D. Huynh C. Jouanne P. Leconte O. Litaize S. Mengelle G. Nogure J-M. Ruggiri O. Srot J. Tommasi C. Vaglio J-F. Vidal A. Santamarina D. Bernard. *The JEFF-3.1.1 Nuclear Data Library.* 2009. ISBN: 978-92-64-99074-6.

[6] H. Stehfest. “Algorithm 368: Numerical Inversion of Laplace Transform”. In: *Communications of the ACM* 13 (1970), pp. 47–49.

[7] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing In Science & Engineering* 9 (2007), pp. 90–95.