UNIVERSITY OF BIRMINGHAM

Department of Metallurgy & Materials

Activity V2 Manual

Contents

Chapter 1

Overview

1.1 Reason for the Program

The Activity program calculates how radioactive a target becomes after being irradiated by high energy ions. It uses the TENDL-2019 database that contains cross-section data for protons and deuterons, and the JEFF-3.3 Radioactive Decay Data File.

1.2 Program Requirements

The user must provide a exyz file from the SRIM ion transport program and a breakdown of the composition of the target, as well as other irradiation parameters.

Chapter 2

Installation

The program needs Python 3 installed in order to run. At the time of writing, it has only been developed to run on a Linux operating system, but it shouldn't require much adjusting to run on a Windows computer too.

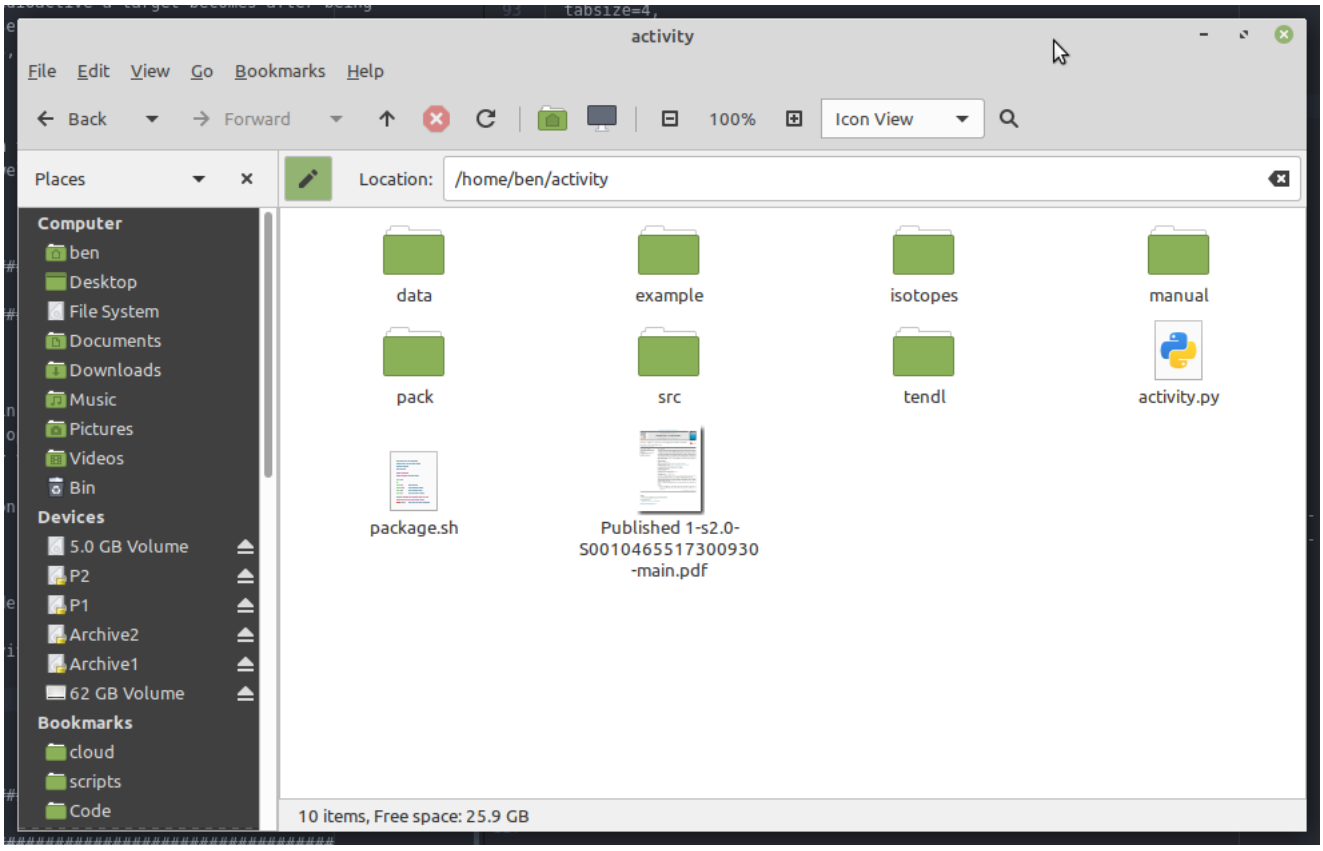
Download and install the latest version of Python 3:

https://www.python.org/downloads

The latest version of the activity code with data files must also be downloaded:

https://github.com/BenPalmer1983/activity_v2

For example, I have installed to /home/ben/activity

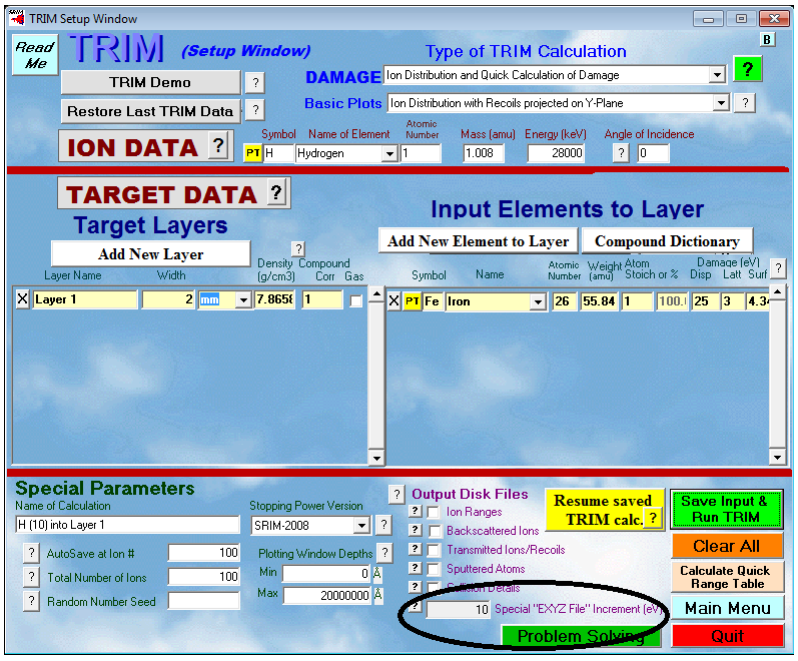


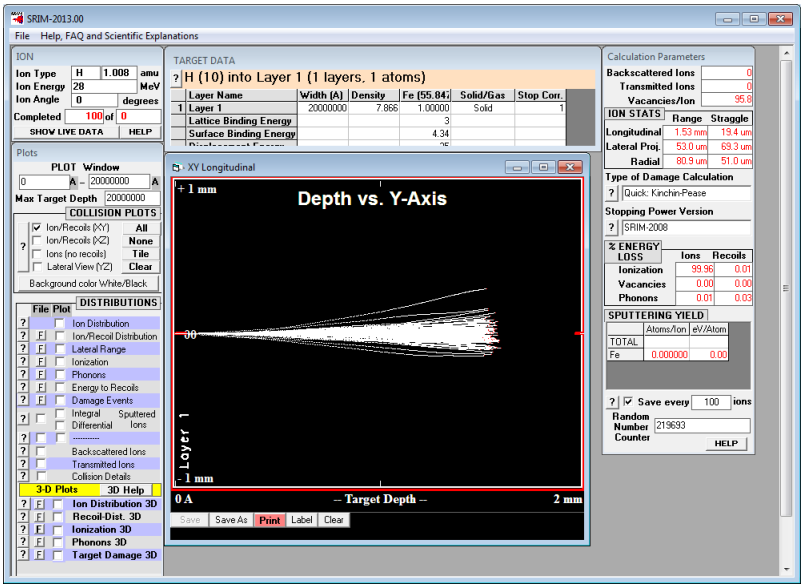
Chapter 3

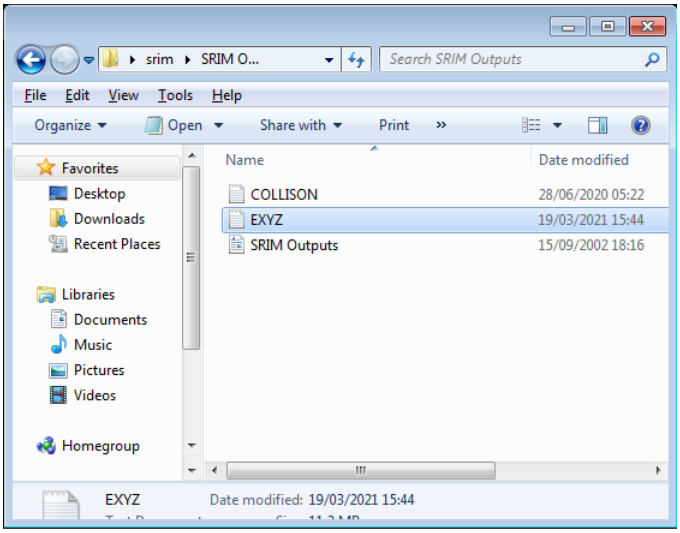
How To Use

3.1 SRIM

If you haven't done so already, run the required simulation in SRIM. When setting up the calculation, be sure to set an increment for the EXYZ file. Sane values that have been used in examples range from 10eV to 10,000KeV, but this will depend on the resolution of the data in the TENDL database, the thickness of your target and the energy of the projectiles.



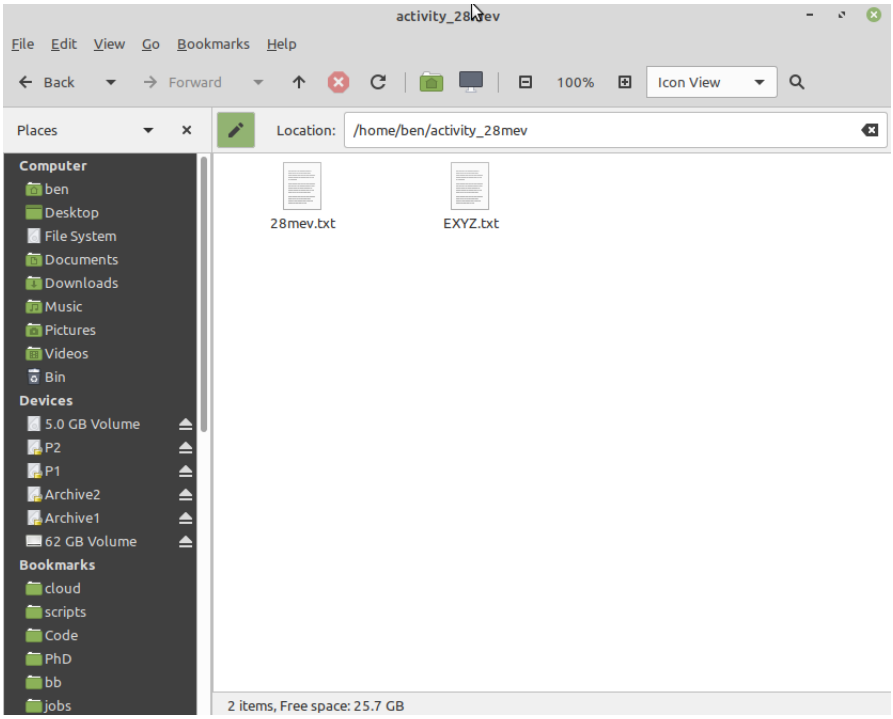




Take a copy of the EXYZ.txt file as it will be required by the activity program.

3.2 Activity

Create a directory to run the calculation. Copy in the EXYZ.txt file and create an input file.



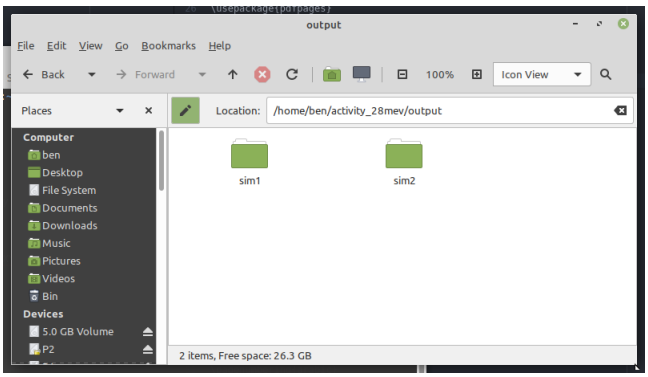
The input file is just a text file, and will specify the calculation details. The data file paths must be specified, and then the simulations can be detailed, each with it's own unique name. The example below is for two simulations - the first with a 300s beam, and the second with a 3000s beam.

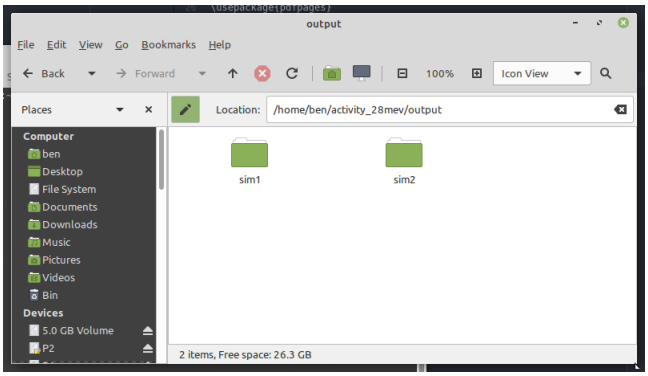
```
# Data files
data isotopes="/home/ben/activity/data/isotopes" xs="/home/ben/activity/data/xs"

# Sim 1
sim1 exyz="EXYZ.txt" target_composition=Fe,100.0 target_depth=0.5,mm target_density=7808,kgm3
    beam_projectile='proton' beam_energy=28,MeV beam_area=64,mm2 beam_duration=300,s beam_current=0.5,
    uA end_time=260000,s

# Sim 2
sim2 exyz="EXYZ.txt" target_composition=Fe,100.0 target_depth=0.5,mm target_density=7808,kgm3
    beam_projectile='proton' beam_energy=28,MeV beam_area=64,mm2 beam_duration=3000,s beam_current
    =5.0,uA end_time=260000,s
```

The program is run from the terminal/command line. Depending on your computer system, python3 might run through the command python3 or python, and on my computer the command is python3.

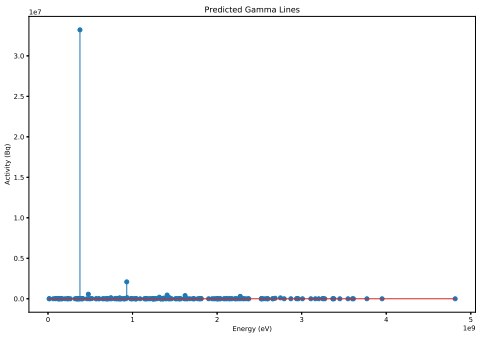
```
python3 /home/ben/activity/activity.py /home/ben/activity_28mev/28mev.txt
```

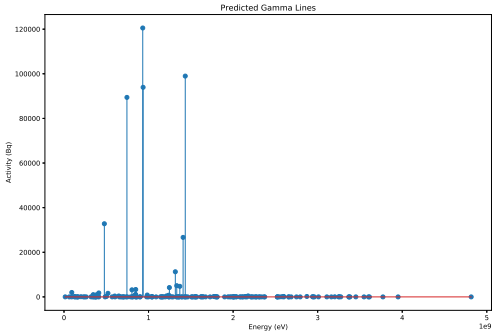
The program outputs into a directory for each simulation.

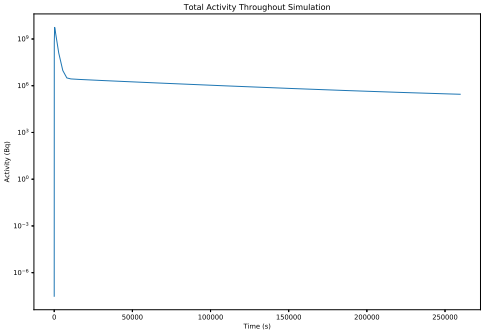




Within each directory are various plots and data files, including the activity over time and predicted gamma lines.




Predicted Gamma Lines
Activity (Bq)
Energy (eV)
1e9


Total Activity Throughout Simulation
Activity (Bq)
Time (s)

Chapter 4

Examples

4.1 Iron - 28MeV Protons

In this example, natural iron makes up the target .

```
# Data files
data isotopes="/home/ben/activity/data/isotopes" xs="/home/ben/activity/data/xs"

# Sim
sim1 exyz="EXYZ.txt" target_composition=Fe,100.0 target_depth=0.5,mm target_density=7808,kgm3
    beam_projectile='proton' beam_energy=28,MeV beam_area=64,mm2 beam_duration=300,s beam_current=0.5,
    uA end_time=260000,s
```

The simulation section defines what the target material is made of as well as the beam parameters.

4.2 Iron Fe56 (only) - 28MeV Protons

In this example, the target is made of Fe56 only, and contains none of the other naturally occurring stable isotopes.

```
# Data files
data isotopes="/home/ben/activity/data/isotopes" xs="/home/ben/activity/data/xs"

# Sim
sim1 exyz="EXYZ.txt" target_composition=Fe56,100.0 target_depth=0.5,mm target_density=7808,kgm3
    beam_projectile='proton' beam_energy=28,MeV beam_area=64,mm2 beam_duration=300,s beam_current=0.5,
    uA end_time=260000,s
```

4.3 Steel - 5MeV Protons

This example was provided by Alex Dickinson-Lomas, with a steel containing a wider range of elements.

```
# Data files
data isotopes="/home/ben/activity/data/isotopes" xs="/home/ben/activity/data/xs"

```

```
# Sim
sim1 exyz="EXYZ.txt" target_composition=Fe,96.375,C,0.772,Cu,0.024,Mn,1.36,Ni,0.698,Si,0.381,Cr,0.092,
    V,0.008,P,0.009,Si,0.003,Mo,0.278 target_depth=0.1,mm target_density=7808,kgm3 beam_projectile='
    proton' beam_energy=5,MeV beam_area=64,mm2 beam_duration=300,s beam_current=0.5,uA end_time
    =260000,s
```

Chapter 5

Decay Equation

5.1 Bateman Equation

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 5.1.

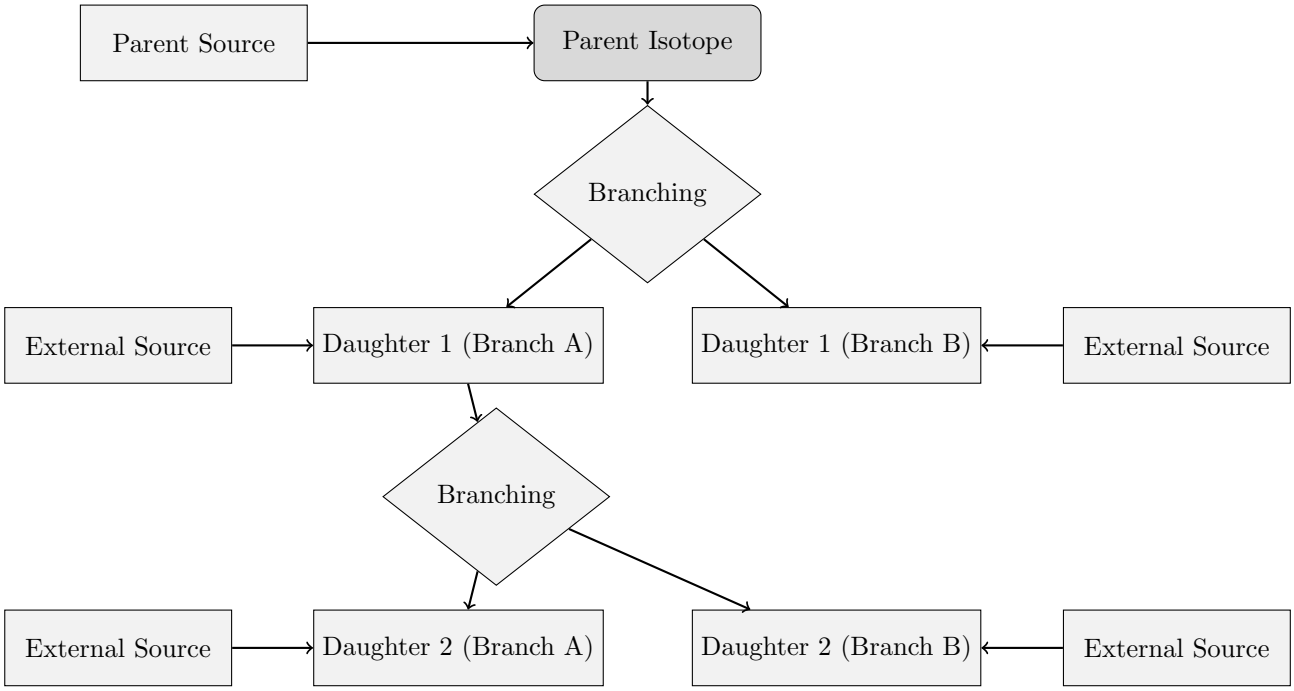


Figure 5.1: An example of several decay chains including branching factors and possible external source terms for each isotope on each chain.

5.1.1 Laplace Transform

Laplace Transforms (5.1) 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.

$$F(s) = \int_0^\infty f(t) \exp(-st) dt \tag{5.1}$$

5.1.2 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 i^{th} isotope
- $N_i(t)$ amount of the i^{th} isotope at time t
- $N_i'(t)$ change in amount of the i^{th} isotope, with respect to time, at time t

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

$$N_1'(t) = \omega_1 - \lambda_1 N_1(t) \tag{5.2}$$

$$N_i'(t) = \omega_i + b_{i-1}\lambda_{i-1}N_{i-1}(t) - \lambda_i N_i(t) \tag{5.3}$$

$$N_z'(t) = \omega_z + b_{z-1}\lambda_{z-1}N_{z-1}(t) \tag{5.4}$$

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

$$N_1(s) = \frac{1}{s+\lambda_1}N_1(0) + \frac{1}{s(s+\lambda_1)}\omega_1 \tag{5.5}$$

$$N_i(s) = \frac{1}{s(s+\lambda_i)}\left(\omega_i\right) + \frac{1}{s+\lambda_i}\left(b_{i-1}\lambda_{i-1}N_{i-1}(s)\right) + \frac{1}{s+\lambda_i}N_i(0) \tag{5.6}$$

$$N_z(s) = \frac{1}{s^2}\omega_z + \frac{1}{s}b_{z-1}\lambda_{z-1}N_{z-1}(s) + \frac{1}{s}N_z(0) \tag{5.7}$$

5.1.3 Numerical Inversion of the Laplace Transform

The Gaver-Stehfest[**stehfest**] 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 (figure **??**).

$$f(t) \approx f_n(t) = \frac{\ln(2)}{t} \sum_{k=1}^{2n} a_k(n) F(s) \text{ where } n \geq 1, t > 0 \tag{5.8}$$

$$s = \frac{k \ln(2)}{t} \tag{5.9}$$

$$a_k(n) = \frac{(-1)^{(n+k)}}{n!} \sum_{j=Floor(\frac{k+1}{2})} j^{n+1} \binom{n}{j} \binom{2j}{j} \binom{j}{k-j} \tag{5.10}$$

The equation for the i$^{\text{th}}$ 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 i$^{\text{th}}$ isotope is reached (changing the equations appropriately for the parent, unstable daugher and stable daughter isotopes).

5.1.4 Analytic Solution by Partial Fraction Expansion

The equation for the i$^{\text{th}}$ isotope in the s domain can be written in full by substituting the preceding equation until the parent isotope is reached, and this full equation 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):

$$\begin{aligned} N_4(s) = &\frac{1}{(s+\lambda_1)(s+\lambda_2)(s+\lambda_3)(s+\lambda_4)} b_2 b_3 b_4 \lambda_1 \lambda_2 \lambda_3 N_1(0) \\ &+ \frac{1}{(s+\lambda_2)(s+\lambda_3)(s+\lambda_4)} b_3 b_4 \lambda_2 \lambda_3 N_2(0) \\ &+ \frac{1}{(s+\lambda_3)(s+\lambda_4)} b_4 \lambda_3 N_3(0) \\ &+ \frac{1}{(s+\lambda_4)} N_4(0) \\ &+ \frac{1}{s(s+\lambda_1)(s+\lambda_2)(s+\lambda_3)(s+\lambda_4)} b_2 b_3 b_4 \lambda_1 \lambda_2 \lambda_3 \omega_1 \\ &+ \frac{1}{s(s+\lambda_2)(s+\lambda_3)(s+\lambda_4)} b_3 b_4 \lambda_2 \lambda_3 \omega_2 \\ &+ \frac{1}{s(s+\lambda_3)(s+\lambda_4)} b_4 \lambda_3 \omega_3 \\ &+ \frac{1}{s(s+\lambda_4)} \omega_4 \end{aligned} \tag{5.11}$$

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

$$
\begin{aligned}
N_4(s) = &\frac{1}{s(s+\lambda_1)(s+\lambda_2)(s+\lambda_3)} b_2 b_3 b_4 \lambda_1 \lambda_2 \lambda_3 N_1(0) \\
&+ \frac{1}{s(s+\lambda_2)(s+\lambda_3)} b_3 b_4 \lambda_2 \lambda_3 N_2(0) \\
&+ \frac{1}{s(s+\lambda_3)} b_4 \lambda_3 N_3(0) \\
&+ N_4(0) \\
&+ \frac{1}{s^2(s+\lambda_1)(s+\lambda_2)(s+\lambda_3)} b_2 b_3 b_4 \lambda_1 \lambda_2 \lambda_3 \omega_1 \\
&+ \frac{1}{s^2(s+\lambda_2)(s+\lambda_3)} b_3 b_4 \lambda_2 \lambda_3 \omega_2 \\
&+ \frac{1}{s^2(s+\lambda_3)} b_4 \lambda_3 \omega_3 \\
&+ \frac{1}{s^2} \omega_4
\end{aligned}
\tag{5.12}
$$

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

$$
N_m(t; \vec{\lambda}, \vec{b}, \vec{w}) = \sum_{k=1,m} r(k; \vec{\lambda}, \vec{b}) \left[f(t; k, m, \vec{\lambda}) N_k(0) + g(t; k, m, \vec{\lambda}) w_k \right] \tag{5.13}
$$

$$
r(k, m, \vec{\lambda}) = \begin{cases} \prod_{i=k,m-1} \left(b_{i+1} \lambda_i \right), & \text{if } k < m \\ 1, & \text{if } k = m \end{cases} \tag{5.14}
$$

$$
f(t; k, m, \vec{\lambda}) = (-1)^{m-k} \sum_{i=k,m} \left[\exp(-\lambda_i t) \prod_{j=k,m; j \neq i} \left(\frac{1}{\lambda_i - \lambda_j} \right) \right] \tag{5.15}
$$

$$
g(t; k, m, \vec{\lambda}) = \frac{1}{\prod_{i=k,m} \lambda_i} + (-1)^{m-k+1} \sum_{i=k,m} \left[\frac{1}{\lambda_i} \exp(-\lambda_i t) \prod_{j=k,m; j \neq i} \left(\frac{1}{\lambda_i - \lambda_j} \right) \right] \tag{5.16}
$$

The set of equations below is used to calculate the amount of the m^{th} isotope in the decay chain, where the m^{th} isotope is stable.

$$
N_m(t; \vec{\lambda}, \vec{b}, \vec{w}) = N_m + w_m t + \sum_{k=1,m-1} r(k; \vec{\lambda}, \vec{b}) \left[f(t; k, m-1, \vec{\lambda}) N_k(0) + g(t; k, m, \vec{\lambda}) w_k \right] \tag{5.17}
$$

$$
r(k, m, \vec{\lambda}) = \begin{cases} \prod_{i=k,m-1} \left(b_{i+1} \lambda_i \right), & \text{if } k < m \\ 1, & \text{if } k = m \end{cases} \tag{5.18}
$$

$$f(t;k,m,\vec{\lambda}) = \frac{1}{\prod_{i=k,m}\lambda_i} + (-1)^{m-k+1}\sum_{i=k,m}\left[\frac{1}{\lambda_i}\exp(-\lambda_i t)\prod_{j=k,m;j\neq i}\left(\frac{1}{\lambda_i-\lambda_j}\right)\right] \tag{5.19}$$

$$g(t;k,m,\vec{\lambda}) = \frac{1}{\prod_{i=k,m}\lambda_i}t + \frac{\sum_{i=k,m}\left[\prod_{j=k,m;j\neq i}\lambda_j\right]}{\prod_{i=k,m}\lambda_i^2} + (-1)^{m-k+1}\sum_{i=k,m}\left[\frac{1}{\lambda_i^2}\exp(-\lambda_i t)\prod_{j=k,m;j\neq i}\left(\frac{1}{\lambda_i-\lambda_j}\right)\right] \tag{5.20}$$

5.2 Python Isotopes Class

The decay equations are computed within the isotopes class in the isotopes.py file.

```
1
2  class isotopes:
3
4    ########################################
5    # DECAY EQUATIONS
6    ########################################
7
8    @staticmethod
9    def calculate_activity(t, l, b, w, n0):
10     nt = numpy.zeros((len(n0),),)
11     for m in range(0,len(n0)):
12       if(l[m] > 0.0):
13         nt[m] = isotopes.activity_unstable(t, l, b, w, n0, m)
14       elif(l[m] == 0.0):
15         nt[m] = isotopes.activity_stable(t, l, b, w, n0, m)
16     return nt
17
18   @staticmethod
19   def activity_unstable(t, l, b, w, n0, m):
20     s = 0.0
21     for k in range(0, m+1):
22       s = s + isotopes.r(k, m, b, l) * ( isotopes.f_unstable(t,k,m,l) * n0[k] + isotopes.g_unstable(t,
           k,m,l) * w[k])
23     return s
24
25   @staticmethod
26   def activity_stable(t, l, b, w, n0, m):
27     s = n0[m] + w[m] * t
28     for k in range(0, m):
29       s = s + isotopes.r(k, m, b, l) * (isotopes.f_stable(t,k,m,l) * n0[k] + isotopes.g_stable(t,k,m,l
           ) * w[k])
30     return s
31
32   @staticmethod
33   def r(k, m, b, l):
34     if(k == m):
35       return 1.0
```

```
    else:
      p = 1.0
      for i in range(k, m):
        p = p * (b[i] * l[i])
      return p

  @staticmethod
  def f_unstable(t,k,m,l):
    s = 0.0
    for i in range(k, m+1):
      p = 1.0
      for j in range(k, m+1):
        if(i != j):
          p = p * (1 / (l[i] - l[j]))
      s = s + numpy.exp(-1 * l[i] * t) * p
    s = (-1)**(m-k) * s
    return s

  @staticmethod
  def g_unstable(t,k,m,l):
    pa = 1.0
    for i in range(k,m+1):
      pa = pa * l[i]
    pa = 1.0 / pa
    s = 0.0
    for i in range(k, m+1):
      pb = 1.0
      for j in range(k, m+1):
        if(i != j):
          pb = pb * (1 / (l[i]-l[j]))
      s = s + (1/l[i]) * numpy.exp(-l[i]*t) * pb
    return pa + s * (-1)**(m-k+1)


  @staticmethod
  def f_stable(t,k,m_in,l):
    m = m_in - 1

    p = 1.0
    for i in range(k, m+1):
      p = p * l[i]

    s = 0.0
    for i in range(k, m+1):
      r = l[i]
      for j in range(k, m+1):
        if(i != j):
          r = r * (l[i] - l[j])
      s = s + (1/r)*numpy.exp(-1*l[i]*t)

    return (1.0/p) + s * (-1.0)**(m-k+1)


```

```
89    @staticmethod
90    def g_stable(t,k,m_in,l):
91      m = m_in - 1
92
93      pa = 1.0
94      for i in range(k,m+1):
95        pa = pa * l[i]
96      pa = 1.0 / pa
97
98      sa = 0.0
99      for i in range(k, m+1):
100       pb = 1.0
101       for j in range(k,m+1):
102         if(j != i):
103           pb = pb * l[j]
104       sa = sa + pb
105     pc = 1.0
106     for i in range(k, m+1):
107       pc = pc * l[i]**2
108
109     sb = 0.0
110     for i in range(k, m+1):
111       pd = 1.0
112       for j in range(k, m+1):
113         if(i != j):
114           pd = pd * (1 / (l[i]-l[j]))
115       sb = sb + (1/(l[i]**2)) * numpy.exp(-l[i]*t) * pd
116
117     return pa * t + sa / pc + sb * (-1)**(m-k+1)
```