CNUCTRAN: A C++ NUCLIDES TRANSMUTATION SIMULATION CODE

M. R. Omar

SCHOOL OF PHYSICS, UNIVERSITI SAINS MALAYSIA

1. Introduction

Welcome to the *CNUCTRAN* Beginner's Guide. This document will guide you through the essential aspects of using *CNUCTRAN* to perform nuclear burnup simulations. Throughout the process, you will go through the following key topics:

- Calculation Objective
- What is *CNUCTRAN?*
- InstallationHow does *CNUCTRAN* works?
- *CNUCTRAN* Installation
- The basics of Extensible Markup Language (XML)
- Creating an input file: Defining Zones & Simulation Settings
- Examples of an input file.
- Program output
- Theory

2. Calculation Objective

Suppose you have a system of I nuclides. As time progresses from $t=0$ to $t=T$, the initial concentration of these nuclides will evolve into the final concentration. Such an evolution is due to various transmutation reactions occurring within the time interval. *CNUCTRAN* aims to compute the final concentration of these transmuting nuclides. Some nuclides will increase in concentration, and some will deplete. Note also that the phrase “concentration of nuclide A” also means the nuclei count of nuclides A.

3. What is *CNUCTRAN*?

CNUCTRAN is a C++ code that simulates various nuclear transmutations such as decays, fissions, and neutron absorptions. Most importantly, it helps physicists avoid cumbersome numerical issues of solving the nuclide burnup equations using the probabilistic approach. Several other techniques, such as the Transmutation Trajectory Analysis and Chebychev Rational Approximation Method, are also available, and these methods impose several pros and cons. The primary purpose of *CNUCTRAN* is to explore the feasibility of using the probabilistic approach to solving nuclides burnup problems. When we talk about the probabilistic method, the method may sound stochastic in nature. In reality, the technique does not incorporate any means of random sampling, and the final concentration produced is definite and not contaminated with random errors.

CNUCTRAN works based on the probabilistic method, where it carefully tracks nuclide transformations into one another in a depletion problem. Interestingly, the probabilistic method does not directly solve Bateman equations. Instead, the technique reduces nuclide transformations into a sparse transfer matrix, $\mathbf{T}$, whose elements are made up of various transformation probabilities. Next, the transfer matrix serves as a multiplier of the initial nuclide concentration, $\mathbf{y}_0$, producing the final nuclide concentrations, $\mathbf{y}$.

Technically, the users may incorporate as many nuclides as possible into a burnup calculation. The calculation involves I nuclides, then, $\mathbf{y}_0$ and $\mathbf{y}$ are column vectors of I elements, and $\mathbf{T}$ is a $I \times I$ sparse square matrix. *CNUCTRAN* aims to compute matrix $\mathbf{T}$, so that the final nuclides concentration can be expressed as,

$$\boldsymbol{y} = \mathbf{T}\,\mathbf{y}_0 \tag{1}$$

Constructing the matrix $\mathbf{T}$ requires several important pieces of information. These include the half-life, λ, of the nuclides involved in the calculation, as well as the condition of the environment such as the neutron flux, ϕ, which will initiate neutron-induced transmutation reactions. Do not worry, these details will be discussed further in the upcoming sections.

Of course, the probabilistic approach has its pros and cons. The probabilistic method imposes fewer approximations in solving a problem. Therefore, it is plausible to use it as a benchmark for a solution of the same case by deterministic means. However, it comes at the expense of facing the limitations presented in Table 1.

Table 1: The pros and cons of the probabilistic method.

Pros	Cons
It is easy to understand. Understanding its method of solution does not require advanced knowledge in advanced mathematics.	The method requires arbitrary precision arithmetics. The conventional float/double data types may contaminate the accuracy of the final nuclides concentration.
The solution produced is an approximated nuclides concentration, and the accuracy of the approximation is heavily dictated by the user.	The method requires sparse matrix-matrix multiplication (SpMM) parallelization to improve its computational efficiency.
Its computational efficiency is practical. For instance, it can solve pure decay problems with ~1500 nuclides in less than 1 second using 3.2GHz 8 cores CPU.	For a system of nuclides under neutron irradiation, the neutron flux is assumed to be constant throughout the calculation time step.
The method can simulate complex burnup chains, and one does not need to linearize the burnup chains.	
The method faithfully simulates the transformation of nuclides and is free from any mathematical jargon. There is no risk of producing negative nuclides concentrations.	

4. How Does *CNUCTRAN* Works?

Before doing anything, the code first needs to have a model of some problem of interest. This could be one or more zones in a nuclear reactor or any other physical system containing the nuclides under study. You, as the code user, will need to describe the model so that the code can do something with it. A basic model consists of a few things:

- The description of zones – the problem must be split up into regions with different nuclides composition. For example, each nuclear fuel pin can serve as a single zone. The various fuel pins correspond to the different zones.
- Various simulation settings that tell the code how many precision digits need to include, the time step, the order of the calculation and what options to use.

CNUCTRAN retrieves these definitions in terms of XML markups written in a single input file (with the *.xml* extension). Therefore, the user must know how to use XML markups. But do not worry, XML is just a simple markup language and it is easy to learn and use. If you have no idea what XML is, please visit this XML tutorial website: https://www.w3schools.com/xml/.

5. How to Install

The current development of *CNUCTRAN* only provides a Windows executable. For other operating systems, you may have to compile it on your own. Unfortunately, you may have to replace the Microsoft Parallel Pattern Library (PPL) with any compatible infrastructure such as OpenMP. Note that *CNUCTRAN* requires PPL for its optimization.

The installation of *CNUCTRAN* involves the following simple steps:

Step 1: Visit https://github.com/rabieomar92/cnuctran/releases. Select the latest release and download the compressed file named *cnuctran.zip*.

Step 2: Extract the contents of *cnuctran.zip* into the same working directory.

Step 3: Double-click the *cmd* link file. This will pop up the command prompt console from the current working directory.

Step 4: Prepare the XML input file. You may rename the input XML file according to your needs. If you are a first-timer, you may skip this step because a predefined input file is already included in *cnuctran.zip*.

Step 5: If the input file name is "*input.xml*", then type in "`cnuctran`" in the command prompt console, then, press "enter". Or else, type in "`cnuctran [anyname].xml`".

When running the code, make sure that it is error-free. If it runs properly, then you are ready to go!

6. Extensible Markup Language (XML)

You may skip this section if you are used to XML.Unlike many other nuclear-related codes, which use an arbitrary-format ASCII file with "cards" to specify a particular geometry, materials, and associated run settings, the input files for *CNUCTRAN* are structured in an XML file. XML, which stands for eXtensible Markup Language, is a simple format that allows data to be exchanged efficiently between different programs and interfaces.

Anyone who has ever seen webpages written in HTML will be familiar with the structure of XML whereby "tags" enclosed in angle brackets denote that a particular piece of data will follow. Let us examine the following example:

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
  <age>27</age>
  <occupation>Health Physicist</occupation>
</person>
```

Here we see that the first tag indicates that the following data will describe a person. The nested tags `firstname`, `lastname`, `age`, and `occupation` indicate characteristics of the person being described. They are the child of `person` tag.

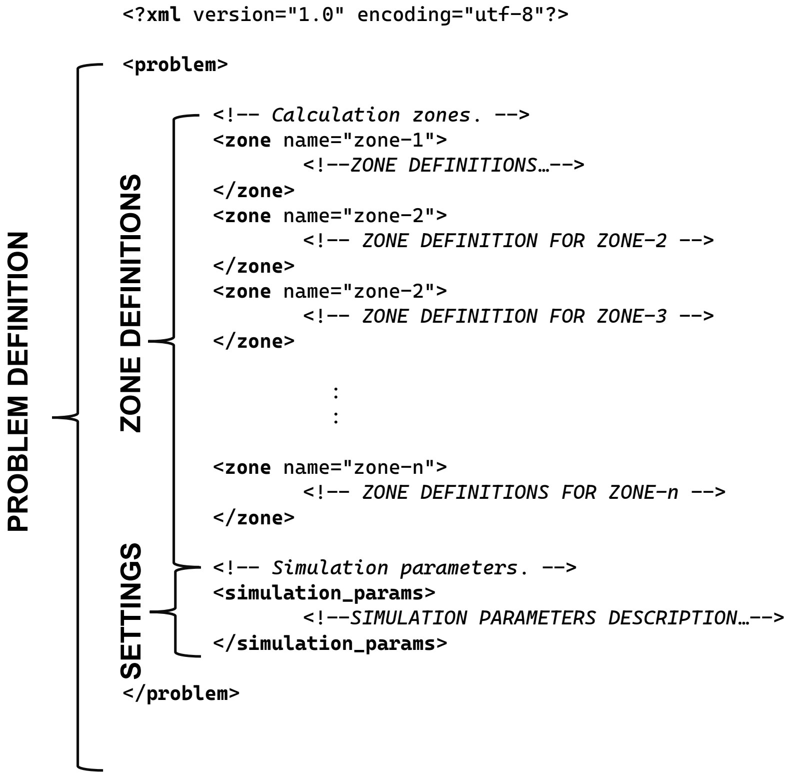
In much the same way, *CNUCTRAN* input uses XML tags to describe the calculation zones, nuclides list, and settings for a burnup calculation.

7. Creating an Input File

Before preparing the input file for CNUCTRAN, make sure that you have gone through Section 5. To put it simply, *CNUCTRAN* requires two input XML files in order to work:

a) An input XML file specifying the problem zones and computation settings. By default, *CNUCTRAN* assumes the input file as *input.xml*.
b) An XML file containing the nuclides library. The user does not need to prepare this XML file, as it is already included in *cnuctran.zip*. The name of the XML file is *chain_endfb71.xml*. The currently available nuclides data library is based on the Evaluated Nuclear Data File (ENDF) version B-VIII.0. To discover more about ENDF: https://www-nds.iaea.org/exfor/e4explorer.htm

The structure of the XML input file is given as follows.



```
<?xml version="1.0" encoding="utf-8"?>

<problem>

    <!-- Calculation zones. -->
    <zone name="zone-1">
            <!--ZONE DEFINITIONS…-->
    </zone>
    <zone name="zone-2">
            <!-- ZONE DEFINITION FOR ZONE-2 -->
    </zone>
    <zone name="zone-2">
            <!-- ZONE DEFINITION FOR ZONE-3 -->
    </zone>

                :
                :

    <zone name="zone-n">
            <!-- ZONE DEFINITIONS FOR ZONE-n -->
    </zone>

    <!-- Simulation parameters. -->
    <simulation_params>
            <!--SIMULATION PARAMETERS DESCRIPTION…-->
    </simulation_params>
</problem>
```

Notice that there should be at least one zone defined in the input file. Also, there is only one attribute for the **`<zone>`** tag, which is the name of the zone. All tags within **`<zone>`** tag are known as the child elements of **`<zone>`**. The specifications of all child tags within the **`<zone>`** tag are given in Table 2. In addition, Table 3 lists the specifications of all child tags within the **`<simulation_params>`** tag.

Table 2: Specifications of all child tags that appear within **`<zone>`** tag.

Tag Name	**Attributes**	**Description**	**Child Value**
`<species>` List all nuclides species involved in the calculation.	`source`	Sets the nuclides library location on the disk. This should be the location of the `chain_endfb71.xml` file.	The child value must be a list of nuclide names defined in the nuclides library. The names should be separated by a newline, whitespaces or commas. If `amin` and/or `amax` are prescribed, then *CNUCTRAN* will ignore the specified list of nuclide names (if any).
	`amin`	Sets the minimum atomic number, A, to be included in the calculation. Any nuclides with $A < A_{\mathrm{min}}$ will not be included in the calculation. Optional.	
	`amax`	Sets the maximum atomic number, A, to be included in the calculation. Any nuclides with $A > A_{\mathrm{max}}$ will not be included in the calculation. Optional.	
`<initial_concentrations>` Specifies the initial concentration of all nuclides involved in the calculation.	`source`	Sets the initial concentration XML file location on the disk. This file can be the XML output of the previous calculation. Optional.	See `<concentration>` tag.
	`override`	This attribute is useful if the `source` attribute is prescribed. There are two allowed values: `true` – all nuclides listed in the initial concentration source that is still not included in the code will be automatically registered. `false` – otherwise. Optional if `source` attribute is not set.	
`<concentration>` Defines a nuclide concentration value.	`species`	The name of the nuclide species.	n/a.
	`value`	The value of the concentration. The concentration unit can be anything including mol, kg, g, unitless (nuclei count), etc. However, the unit must be consistent throughout the input file.	
`<reaction_rates>` List all neutron-induced reactions associated with various nuclides involved in the calculation. Optional.	n/a	n/a	See `<reaction>` tag.
`<reaction>` Defines a reaction rate value with the type of reaction and its target nuclide. Optional.	`species`	The name of the target nuclide species undergoing the reaction.	n/a
	`type`	The reaction type. Values can be `fission`, `(n,gamma)`, `(n,2n)`, `(n,a)` depending on the prescribed nuclides data library.	
	`rate`	Reaction rate value in the unit of per second. This rate value can be calculated using the neutron flux obtained from neutronic simulations, i.e. Monte Carlo codes and deterministic transport codes.	

Table 3: Specifications of all child tags that appear within the **`<simulation_params>`** tag.

Tag Name	Child Value
`<n>` Sets the approximation order of the calculation. The precision is measured in terms of 10^{-n}. For more info on the approximation order, refer to Section 9.4.	The child value must be an integer greater than 0. A value greater than 10 is recommended. However, a higher value will increase the computational cost.
`<time_step>` Sets the time step of the calculation in seconds.	The child value must be a float number greater than 0.
`<precision_digits>` Sets the minimum number of accurate digits maintained in the arithmetics.	The child value must be an integer greater than 30. A higher value will increase the computational cost. Depending on the problems, it is recommended to set the precision to >40 for the approximation order, $n > 10$. The default value is 45.
`<output_digits>` Sets the number of decimal points to be displayed in the output.	The child value must be an integer greater than 0. The default value is 16.
`<output>` Sets the location of the output XML file on the disk.	The child value must be a valid target file name of the XML output file. The default value is "**`output.xml`**".
`<epsilon>` Sets the minimum nuclide concentration value threshold, such that all concentrations below the threshold value will be excluded from the output.	The child value must be a positive small floating points number..
`<verbosity>` Sets the verbosity level. A higher verbosity means more console output messages will be displayed.	The child value can be either 0 or 1. A higher value indicates a higher verbosity level. The default verbosity level is 0.

8. Sample Input File

Example #1: A burnup zone consists of nuclides with an atomic number between $A = 100$ and $A = 260$. The problem is a pure decay problem and we are interested to calculate the final nuclides concentration after 1 million years (~3.1556926×10^{13} seconds). The location of the nuclides data library is in the same working directory with the name *chain_endfb71.xml*. The order of the calculation is $n = 15$ and the number of precision digits is 50. The final nuclide concentrations should be stored in *output.xml* located within the same working directory. The initial concentration of Np237 is 1 mol.

```
  <?xml version="1.0" encoding="utf-8"?>

  <problem>

     <zone name="myzone">
       <species source=".\chain_endfb71.xml" amin="150" amax="250"/>
        <initial_concentrations>
           <concentration species="Np237" value="1.0" />
        </initial_concentrations>
     </zone>

     <simulation_params>
       <n>15</n>
        <time_step>3.1536e+13</time_step>
        <precision_digits>50</precision_digits>
        <output_digits>15</output_digits>
        <verbosity>1</verbosity>
        <output>.\output.xml</output>
     </simulation_params>

</problem>
```

Example #2: A burnup zone consists of nuclides with an atomic number between $A = 100$ and $A = 260$. The problem is a pure decay problem and we are interested to calculate the final nuclides concentration after 1 million years (~3.1556926×10^{13} seconds). The location of the nuclides data library is in the same working directory with the name *chain_endfb71.xml*. The order of the calculation is $n = 15$ and the number of precision digits is 50. The final nuclide concentrations should be stored in *output.xml* located within the same working directory. The initial concentration of Np237 is 1 mol. The defined zone is under neutron irradiation, such that Np237 experiences (n, a) and fission reactions with the rates at 0.0001 per sec. and 0.00001 per sec., respectively.

```
  <?xml version="1.0" encoding="utf-8"?>

  <problem>

     <zone name="myzone">
       <species source=".\chain_endfb71.xml" amin="150" amax="250"/>
        <initial_concentrations>
           <concentration species="Np237" value="1.0" />
        </initial_concentrations>
        <reaction_rates>
           <reaction species="Np237" type="(n,a)" rate="1E-4" />
           <reaction species="Np237" type="fission" rate="1E-5" />
        </reaction_rates>
     </zone>

     <simulation_params>
       <n>15</n>
        <time_step>3.1536e+13</time_step>
        <precision_digits>50</precision_digits>
        <output_digits>15</output_digits>
        <verbosity>1</verbosity>
        <output>.\output.xml</output>
     </simulation_params>

</problem>
```

9. Program Output

At the end of the calculation, *CNUCTRAN* will produce an output XML file containing the final nuclide concentrations. The name and the save location of the output XML can be set by the user manually through the input file. This can be done by using the **<output>** tag (See Table 3). An example of the output XML file is given as follows:

```
<?xml version="1.0" encoding="utf-8"?>

<output>
        <nuclide_concentrations zone="myzone" amin = "150" amax="250"
        total_nuclides="1489" time_step="3.1536e+13">
                <concentration species="Np237" value="7.237642260179204e-01" />
                <concentration species="Pa233" value="2.493122050609457e-08" />
                <concentration species="U233" value="5.702155930771045e-02" />
                <concentration species="Th229" value="2.635908035643954e-03" />
                <concentration species="Ra225" value="1.464973737813805e-08" />
                <concentration species="Ac225" value="9.832038604770428e-09" />
                <concentration species="Fr221" value="3.345624247467714e-12" />
                <concentration species="At217" value="3.675634802490104e-16" />
                <concentration species="Bi213" value="3.112578215430705e-11" />
                <concentration species="Po213" value="4.673987818740959e-20" />
                <concentration species="Tl209" value="3.304426094035331e-14" />
                <concentration species="Pb209" value="1.332557613880556e-10" />
                <concentration species="Bi209" value="2.165630965799592e-01" />
        </nuclide_concentrations>
</output>
```

Users can also set the verbosity level of the console output via the <verbosity> tag. The value of the tag can be either 0, 1 or 2, where a higher value indicates a higher verbosity level. Setting the verbosity level to zero causes *CNUCTRAN* to suppress all progress messages. However, it does not suppress the error messages. Setting the verbosity level to 1 allows *CNUCTRAN* to print several important messages to the console output. These messages include the time step, the total number of nuclides included in the calculation, and the CPU time of the computation.

10. Theory

10.1 Bateman Equations

Bateman's equation of a burnup problem is given as follows,

$$\frac{dy_i}{dt} = \underbrace{\sum_{j=1}^{I} b_{ij}\,\lambda_j y_j + \sum_{k=1}^{I} f_{ik}\Lambda_{ik} y_k}_{\text{production}} - \underbrace{(\lambda_i + \Lambda_i) y_i}_{\text{removal}} \tag{2}$$

y_i is the atom density of nuclide-*i*;
λ_i is the radioactive decay constant of isotope-*i* causing its removal from the system (the decay constant);
λ_{ij} is the decay constant of nuclide-*j* causing the production of nuclide-*i* in the system;
b_{ij} is the decay branching ratio from nuclide-*j* into nuclide-*i*;
f_{ik} is the production yield from nuclide-*k* into nuclide-*i*; and,
Λ_{ij} is the production rate of nuclide-*i* due to the removal of nuclide-*k* from the system.

Eq. (2) is a system of I linear differential equations that are dependent on one another. The solution of Eq. (2) is the concentration of I nuclides. The term on the LHS of the equation is known as the rate of change of nuclide-*i* concentration. The LHS of the equation is balanced with the terms on the RHS. Here, each term describes the various transmutations of nuclides that lead to the production and removal of nuclide-*i* within the system.

In particular, the first term on the RHS represents the natural decay of various nuclides that transforms themselves into nuclide-*i*. Whereas the second term of the RHS describes the formation of nuclides-i via various transmutation reactions of all nuclides excluding the natural decay reaction. Finally, the last term on the RHS is a combination of natural decay and various transmutation reactions that transform nuclide-i into other nuclides, which removes it from the system.

Alternatively, Eq. (2) can be expressed in its compact matrix form given by,

$$\frac{d\mathbf{y}}{dt} = \mathbf{A}\mathbf{y} \tag{3}$$

where $\mathbf{A}$ is the transmutation matrix such that its elements correspond to all coefficients defined in Eq. (2). In general, $\mathbf{A}$ is a sparse $I \times I$ square matrix, where most of its elements are zeros. On the other hand, $\mathbf{y}$ is a column vector corresponding to the concentration of all I nuclides defined in the calculation. Technically, the norm of $\mathbf{A}$ is in the order of $\sim 10^{21}$ and contains both off-diagonal elements and diagonal elements.

At a glance, obtaining the solution of Eq. (3) seems to be straightforward where the separation of variables is applicable, giving,

$$\mathbf{y} = \exp(\mathbf{A}t)\,\mathbf{y}_0 \tag{4}$$

Alas, the evaluation of the matrix exponential in Eq. (4) is not an easy job. Naively, one may approximate the matrix exponential by using the Taylor series,

$$\exp(\mathbf{A}t) \approx \sum_{k=0}^{\infty} \frac{1}{k!} \mathbf{A}^k t^k \tag{5}$$

However, burnup problems are stiff, such that matrix $\mathbf{A}$ consists of elements with extremely small and large numeric values. Thus, the evaluation of Eq. (5) for a transmutation matrix is not practical and does not always lead to an accurate and reliable result. Several other methods include the Padè approximation of the matrix exponential, which unfortunately suffers the same drawback as the Taylor series approximation.

(Maria, 2016) discovered that the eigenvalues of the matrix $\mathbf{A}$ lie near the negative real axis, thus enabling the approximation of the matrix exponential via the Chebychev approximants. Here, the technique is known as the Chebychev Rational Approximation Method (CRAM) where the matrix exponential approximation is given by

$$\exp(\mathbf{A}t) = \alpha_0 + 2\,\mathrm{Re}\left[\sum_{i=1}^{k/2} \alpha_i (\mathbf{A}t - \theta_i \mathbf{I})^{-1}\right] \tag{6}$$

where α_i and θ_i are complex numbers published by Maria, (2016) for the approximation order $k \in \{16, 32, 48\}$. Here, the computation of Eq. (5) requires solving $k/2$ linear systems. Due to the sparsity pattern of the burnup matrix, the linear systems can be solved efficiently using sparse Gaussian elimination (Pusa & Leppänen, 2013). To date, CRAM is the most efficient and able to handle large burnup problems. However, there are several limitations of the CRAM method which have been discussed in-depth by Calvin et al., (2021).

10.2 Discrete Probability Distribution of Transmutation Events

Suppose we select a nucleus during a particular time t. Depending on the environment, the nucleus has various transmutation event options that cause it to transform into another daughter nucleus. It is convenient to index these possible events as $j \in \{1,2,3,\ldots,J_i\}$. Conveniently, one may assign event-1 as neutron absorption, event-2 as fission, event-3 as radioactive decay, and so on. Also, J_i is the total number of events defined for nuclide i. For simplicity, $j = 0$ is designated as the case when no transmutation is occurring within the interval $(t, t + \Delta t)$.

Now, consider that the parameter Λ_{ij}, characterizing the frequency of each possible event-j is known. Also, $\Lambda_{i,j}$ is also known as the transmutation coefficient and it is assumed to be a fixed constant. For a decay reaction that transforms nuclide i into its daughter nuclide, $\Lambda_{i,j}$ is the decay rate constant or the decay constant. For systems of nuclides under neutron irradiation, $\Lambda_{i,j}$ will be a function of neutron flux. Therefore, it is necessary to assume that the flux is approximately constant over the time interval. The probability for event-j occurring within the interval $(t, t + \Delta t)$ is given by

$$p_{i,j} = 1 - e^{-\Lambda_{i,j}\Delta t} \tag{7}$$

Also, the probability of zero-chance of event-j to occur is the complement of the probability given in Eq. (7),

$$q_{i,j} = e^{-\Lambda_{i,j}\Delta t} \tag{8}$$

The probability, $p_{i,j}$ can be easily derived and theoretically exact.

So far, we have established the probability of a single event. Bear in mind that a parent nucleus is offered with various other possible transmutation events, $j \in \{0,1,2,\ldots,J_i\}$, and only one of these events will be finally selected. For simplicity, let E_1 be the condition where event-j is occurring, and E_2

be a condition where all events except removal event j are not occurring. The joint probability of both E_1 and E_2 are true is given by,

$$\mathrm{P}(\mathrm{E}_1 \cap \mathrm{E}_2) = p_{i,j} \times \prod_{\forall l \neq j} q_{i,l} \tag{9}$$

Substituting Eqs. (7) and (8) into Eq. (9) yield the un-normalized probability distribution,

$$\tilde{\pi}_{i,j} = \left(1 - e^{-\Lambda_{i,j}\Delta t}\right) \times \prod_{\forall l \neq j} e^{-\Lambda_{i,l}\Delta t} \tag{10}$$

The probability of no-removal, $j = 0$ is given by the product of the zero-chance probability of all possible events,

$$\tilde{\pi}_{i,0} = \prod_{\forall l} e^{-\Lambda_{i,l}\Delta t} \tag{11}$$

Therefore, the normalized joint Poisson distribution, $\pi(j; \Lambda_{i,1}, \ldots, \Lambda_{i,J_i})$, can be formally written as,

$$\pi_{i,j} \equiv \pi_j(\Delta t; \Lambda_{i,1}, \ldots, \Lambda_{i,J_i}) = c_i \prod_{l=1}^{J_i} \left(\delta_{jl} + (-1)^{\delta_{jl}} e^{-\Lambda_{i,l}\Delta t}\right) \tag{12}$$

where c_i is the normalization constant and δ_{jl} is the Kronecker delta. The value of c_i is given by,

$$c_i = \left(\sum_{j=0}^{J_i} \tilde{\pi}_{i,j}\right)^{-1} \tag{13}$$

10.3 The probabilistic method

So far, the establishment of $\pi_j(\Delta t; \Lambda_1, \ldots, \Lambda_J)$ in Eq. (12) allows us to estimate the probability of various transmutation events experienced by a nucleus. To verify the distribution, we aim to estimate the time evolution of nuclide concentrations using π_j. Generally, the concentration of nuclides i at time $t + \Delta t$ is contributed by (a) the un-reacted nuclides i within the interval $(t, t + \Delta t)$ and (b) the transmutation of other nuclides that occur within the interval $(t, t + \Delta t)$, which leads to the production of nuclide i. Of course, the concentration of nuclide i after the next sub-step, $t + \Delta t$, can be expressed as,

$$y_i(t + \Delta t) = \pi_{i,0} y_i(t) + \sum_{\forall l} \pi_{l \to i} y_l(t) \tag{14}$$

where $\pi_{i,0}$ is the probability of nuclide i not undergoing a transmutation event; $\pi_{l \to i}$ is the probability of any nuclides l undergoing a transmutation event that transforms themselves into nuclide i

$$\pi_{l \to i} = \sum_{\forall j \in \mathscr{R}} \pi_j \tag{15}$$

where $\mathscr{R}$ is a set of transmutation events that transform nuclide l into daughter nuclide i. If removal event j is a fission reaction, then π_j in Eq. (15) must be scaled to the fission yield of the daughter nuclide i. Eq. (14) can be systematically written in matrix form via the definition given in Eq. (15). Here, by considering all I nuclides in a depletion problem, then,

$$\begin{pmatrix} y_1(t+\Delta t) \\ y_2(t+\Delta t) \\ \vdots \\ y_I(t+\Delta t) \end{pmatrix} = \begin{pmatrix} \pi_{1,0} & \pi_{2\to1} & \cdots & \pi_{I\to1} \\ \pi_{1\to2} & \pi_{2,0} & \cdots & \pi_{I\to2} \\ \vdots & \vdots & \ddots & \vdots \\ \pi_{1\to I} & \pi_{2\to I} & \cdots & \pi_{I,0} \end{pmatrix} \begin{pmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_I(t) \end{pmatrix} \tag{16}$$

Or simply,

$$\mathbf{y}(t+\Delta t) = \mathbf{P}(\Delta t)\,\mathbf{y}(t) \tag{17}$$

Notice that the elements of P are calculated based on the substep interval, Δt. Eq. (17) is a recursion relation where the future nuclide concentrations, $\mathbf{y}(t+\Delta t)$, can be obtained by transforming the nuclide concentrations of the previous time interval via transfer matrix, $\mathbf{P}$. Here, $\mathbf{P}$ serves as the transformation operator. The column indices of $\mathbf{P}$ correspond to the various parent nuclides, and the row indices of $\mathbf{P}$ correspond to the various daughter nuclides. Notice that the diagonal elements of square matrix $\mathbf{P}$ are the probabilities of no transmutation events occurring. The elements of $\mathbf{P}$ can be directly calculated using Eq. (14), provided that all reaction rates associated with the nuclides involved are known.

Suppose that the evolution of nuclides concentration is monitored from the initial time, $t = 0$s to the final time, $t = T$. Then, a small time interval, Δt, is chosen such that the total number of sub-steps, $l = T/\Delta t \in \mathbb{N}$. Finally, $\mathbf{y}(t)$ can be obtained by iterating Eq. (17) for l times, giving,

$$\mathbf{y}(T) \equiv \mathbf{y}(l\,\Delta t) = \mathbf{P}^{l}\,\mathbf{y}(0) \tag{18}$$

where $\mathbf{y}(0)$ is the initial nuclides concentration at $t = 0$. Here, the smaller the value of Δt, the larger the number of sub-steps, thus, the better the accuracy. However, such a computation with small Δt requires arbitrary precision arithmetic to avoid the floating-point error. The evaluation of sparse matrix exponentiation, $\mathbf{P}^{l}$, can be accomplished at an incredible speed via the exponentiation by squaring. Evaluating $\mathbf{P}^{l}$ requires l repeated multiplications of $\mathbf{P}$. Such a naive approach will incur an unreasonably long CPU time, especially when l is large. Gratefully, the exponentiation by squaring helps to evaluate $\mathbf{P}^{l}$ with only $\log_2 l$ multiplications (Gordon, 1998). The trick is that $\mathbf{P}^{l}$ can be efficiently calculated by repeated squaring of $\mathbf{P}$. Since evaluating $\mathbf{P}^{l}$ also involves repeated matrix multiplications, using the naïve matrix multiplication algorithm will impose unnecessary operations on the zero elements of the sparse matrix, $\mathbf{P}$. Therefore, this work also implemented a sparse matrix multiplication algorithm proposed by Gustavson, (1978).

10.4 Approximation Order

Notice that the probabilistic method requires l self-multiplication of the transfer matrix $\mathbf{P}$. In the previous section, such a computation can be done at an incredible speed via exponentiation by squaring. To save computational time, the number of substeps is chosen as

$$l = 2^{k} \tag{19}$$

where k is the actual total number of self-multiplications of $\mathbf{P}$ required during the implementation of exponentiation by squaring. Our objective is to obtain the suitable value of k such that the substep interval Δt is in the order of 10^{-n}. Here, n is known as the *approximation order* which helps us to quantify how accurate is the approximation. Therefore, the time step, T, of the calculation can be expressed as,

$$T \approx 2^{k}\Delta t = 2^{k}10^{-n} \tag{20}$$

Finally, solving for k gives the corresponding number of self-multiplications needed so that $\Delta t = 10^{-n}$,

$$k = \lfloor \log_2 10^{n} T \rfloor \tag{21}$$

Of course, the floor operation is needed because k must be a positive integer number. To compensate for the error due to floor operation, Δt must be corrected,

$$\Delta t = \frac{T}{2^{\lfloor \log_2 10^n T \rfloor}} \tag{22}$$

11. Some Remarks

1. The code user is advised to study the probabilistic method before applying it to real burnup problems. The accuracy of the solution heavily depends on the choice of the approximation order, n, and the number of precision digits. A higher approximation order necessitates more precision digits, thus, eliminating round-off errors during the exponentiation of the transfer matrix $\mathbf{P}$.
2. *CNUCTRAN* efficiency is optimized through code parallelization. Currently, it uses Microsoft Parallel Pattern Library (PPL) to facilitate the parallelization of sparse matrix-matrix multiplication encountered during the computation of Eq. (18). Even though the probabilistic method is not efficient compared to the best available solver when it is implemented serially, parallelization helps to rectify this thus making *CNUCTRAN* a practical code.
3. Based on the verification result of *CNUCTRAN*, it could serve as a benchmarking tool for the verification of other burnup solvers.

References

Calvin, O., Schunert, S., & Ganapol, B. (2021). Global error analysis of the Chebyshev rational approximation method. *Annals of Nuclear Energy*, *150*, 107828. https://doi.org/https://doi.org/10.1016/j.anucene.2020.107828

Gordon, D. M. (1998). A Survey of Fast Exponentiation Methods. *Journal of Algorithms*, *27*(1), 129–146. https://doi.org/https://doi.org/10.1006/jagm.1997.0913

Gustavson, F. G. (1978). Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition. *ACM Trans. Math. Softw.*, *4*(3), 250–269. https://doi.org/10.1145/355791.355796

Maria, P. (2016). Higher-Order Chebyshev Rational Approximation Method and Application to Burnup Equations. *Nuclear Science and Engineering*, *182*(3), 297–318. https://doi.org/10.13182/NSE15-26

Pusa, M., & Leppänen, J. (2013). Solving Linear Systems with Sparse Gaussian Elimination in the Chebyshev Rational Approximation Method. *Nuclear Science and Engineering*, *175*(3), 250–258. https://doi.org/10.13182/NSE12-52