Spring 4-11-2023

# Application of an Empirical Density Law via Python for Aqueous Plutonium Nitrate Systems

Tara L. Robertson
*University of New Mexico - Main Campus*

Tara Robertson

*Candidate*

Nuclear Engineering

*Department*

This thesis is approved, and it is acceptable in quality and form for publication:

*Approved by the Thesis Committee:*

Dr. Christopher Perfetti , Chairperson

Dr. Forrest Brown

Jennifer Alwin

Norann Calhoun

# APPLICATION OF AN EMPIRICAL DENSITY LAW VIA FOR AQUEOUS PLUTONIUM NITRATE SYSTEMS

**by**

**TARA ROBERTSON**

**BACHELOR OF SCIENCE IN NUCLEAR ENGINEERING**
**UNIVERSITY OF NEW MEXICO, 2021**

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

**Master of Science**
**Nuclear Engineering**

The University of New Mexico
Albuquerque, New Mexico

**May 2023**

# APPLICATION OF AN EMPIRICAL DENSITY LAW VIA PYTHON FOR AQUEOUS PLUTONIUM NITRATE SYSTEMS

**by**

**Tara Robertson**

**B.S., Nuclear Engineering, The University of New Mexico, 2021**

**M.S., Nuclear Engineering, The University of New Mexico, 2023**

## ABSTRACT

A predictive density tool has been developed to reduce bias and uncertainty in nuclear criticality safety calculations for aqueous plutonium nitrate systems. Multiple methods to calculate the density were reviewed and an empirical method was selected for use. The selected method was then implemented into a Python tool to calculate the solution density and the atom densities for an MCNP6.2 input file.

The Python tool has been validated and verified against the International Handbook of Evaluated Criticality Safety Benchmark Experiments to predict densities within ± 2.6%. The capability to calculate and utilize densities from this tool is a large improvement from current aqueous plutonium solution density modeling methods. This work will not only provide more accurate models but also lead the way towards a better understanding and potential relaxation in the conservatism of the current aqueous plutonium processing limits.

# TABLE OF CONTENTS

# CHAPTER 1 – INTRODUCTION

Plutonium nitrate is used in facilities which process plutonium from impure sources such as spent nuclear fuel. Since plutonium is produced by a nuclear reaction from uranium, it needs to be separated from the other actinides to be converted to pure plutonium metal or powder. This can be achieved by dissolving the mixed actinide metal into an acid and chemically separating the actinides. Common acids for this process are nitric acid and hydrochloric acid. Once the actinides are separated, a few methods may be used to convert the solution to plutonium metal or plutonium oxide.

Before performing these processes, it is important to develop nuclear criticality safety limits and controls. An unwanted criticality may place operational personnel in danger, damage the facility, and/or have environmental consequences. Therefore, nuclear criticality safety staff perform evaluations in accordance with ANSI/ANS 8.1 [1]: "*Before a new operation with fissionable material is begun, or before an existing operation is changed, it shall be determined that the entire process will be subcritical under both normal and credible abnormal conditions."* These evaluations may include criticality calculations for multiple normal and abnormal conditions and may be performed using a neutron transport code such as MCNP® or SCALE [7].

Geometry descriptions and potential simplifications can serve as a leading source of bias between k-effective results of a model calculated with neutron transport codes and a benchmark experiment k-effective. This source of bias may be combatted through precise knowledge of the process conditions and how they are best represented. Therefore, knowledge of the geometries, volumes, masses, densities, and composition of each material is essential. However, in the case of aqueous plutonium nitrate solutions, the solution density

can be difficult to model or calculate due to the ternary solution having multiple oxidation states and the formation of crystals or polymers at various acidities. Therefore, the need was identified for a predictive capability to calculate the density of aqueous plutonium nitrate solutions.

**Section 1.1. Methodology**

To develop this tool a literature review was conducted on various published density laws including the isopiestic method, the Pitzer method, and empirical modeling methods. The selected method was compared against published solution density data [2][3][4]. Then, the selected method was implemented into a Python tool designed for ease and efficiency for the intended users. Once completed, the tool was tested against the International Handbook of Evaluated Criticality Safety Benchmark Experiments and the existing SCALE fissile solution composition card [7]. Lastly, the density calculated by the tool was used in MCNP6.2 models in which the calculated k-effective results were compared to experimental k-effective results to provide quantification of bias for validation.

# CHAPTER 2 – DENSITY CALCULATION METHODS

## Section 2.1. Background and History

The current method used by Los Alamos National Laboratory for finding the density of aqueous plutonium solutions is a plutonium metal-water mixture approach. This method uses the theoretical alpha-phase density of plutonium metal (19.8 g/cm$^3$) and pure water (0.998 g/cm$^3$) to find the volume each occupies in an assumed total volume of the mixture with a given concentration of plutonium. The mass of the plutonium is known from the given concentration, therefore the volume occupied by that mass can be calculated and the remining volume is the water. Once the volume of water is calculated, the mass of water can be determined and summed to the plutonium mass to find the total mass in the given volume to calculate a density. It neglects the acid, for an aqueous plutonium nitrate system, $HNO_3$ in the solution completely.

This method is widely accepted to be conservative over a range of concentrations and adequate for nuclear criticality safety purposes. However, with this method the nitric acid or hydrochloric acid components of the aqueous solutions cannot be properly accounted for, which introduces a bias in the metal-water mixture as compared with actual plutonium solution systems. Nitrogen and chlorine are thermal neutron absorbers, therefore neglecting them in the system does not facilitate any potential credit for poisons. Chlorine has a much higher thermal neutron absorption cross section than nitrogen and will affect the criticality calculations much more. Updating the density calculation method is still important for plutonium nitrate solutions is still important and the scope of this work is limited to the nitrate system. The thermal neutron absorption cross section for nitrogen is not insignificant and can have some impact on the criticality calculations, if credited as a poison. In addition

to this effect, reducing the high conservatism in the density calculation will also impact the criticality calculations. Therefore, this study explored three models for calculating the density of plutonium nitrate solutions: the Isopiestic method, the Pitzer method, and empirical modeling methods.

**Section 2.2. Isopiestic Method**

The isopiestic method is built upon the behavior of the mixture of two solutions with the same water activity. As long as no specific interactions exists between the two solutions, this would form an isopiestic binary solution where a binary solution is the mixture of two initial solutions, and a ternary solution is the mixture of three initial solutions, if, each initial solution has no interaction between each other. Isopiestic is defined as “of, relating to, or marked by equal pressure”. When this is applied to binary or ternary solutions, this means that each initial solution has the same water activity as the total solution.

This concept was shown to be applicable to volumes as well for an isopiestic binary solution. Where the total volume of the mixture is equal to the sum of each initial solution volume. The final solution density may be easily calculated, if the binary data is known. The binary data is the density, concentration, and water activity for each initial solution, if the water activity is constant. Since solutions tend to be imperfect and a constant and consistent water activity between the initial solutions and the binary solution is unrealistic, a method to extrapolate the binary data may be used. This allows the isopiestic method to be applied to solutions without constant water activities.

The isopiestic method was limited by the solubility limits of the solutions due to crystal formation in the solutions at a maximum plutonium molality of 3 moles/kg. However, as part of the works by Leclaire [5], the extrapolated binary data range has been expanded for

plutonium nitrate solutions to a maximum plutonium molality of 3.668 moles/kg. Another limiting factor for plutonium nitrate solutions is at low acidities, the solution tends to form polymers. This was not adjusted for due to the assertion that the isopiestic density law would still be conservative due to the polymers having a lower solution density than the expected ternary solution. Lastly, the method is limited to temperatures of 25℃.

**Section 2.3. Pitzer Method**

The Pitzer method uses ion interaction parameters to model solution density by extending the Debye-Hückel theory with a virial expansion of the Gibbs energy. Empirical relations are also used to approximate two of the coefficient parameters, therefore terming it semi-empirical. Weber [6] defines the necessary equations and coefficients needed to use this method on multiple actinide solutions. The range of applicability for plutonium nitrate is listed as 0.1 to 1.5 moles/L plutonium, 0.5 to 6 moles/L acid, and 25 to 60℃.

**Section 2.4. Empirical Modeling Methods**

Empirical models are highly effective for a specified data range. However, if the source data has issues with the measurement data, range and diversity of data points, or any assumptions were made, it will directly affect the empirical model and these same biases will be apparent in the model.

There have been many empirical models that have attempted to model aqueous plutonium nitrate solutions. In a polynomial fit, some models use inputs of plutonium concentration and acid molarity while others use input units of plutonium molality and acid molality. Where molality is moles of the solute per kilogram of the solvent, water, for aqueous plutonium nitrate solutions. Using molality in these methods to calculate density is ideal because it does not change with temperature or pressure.

For the purposes of this work, the empirical relation outlined by Weber was considered [6]. The data used for fitting the equation was Charrin [2], Söhnel [3] ($HNO_3$ only), and Hofstetter [4]. Which provides an applicable range to a maximum plutonium molality of 0 to 3.3 moles/kg and acid molality of 0 to 6.27 moles/kg. Additionally, the formula covers temperatures of 25 to 60°C.

**Section 2.5. Method Selection**

When selecting a method to implement into a Python tool, the main considerations were usefulness, ease, and efficiency for the end users. Therefore, what solution data is commonly available for plutonium nitrate solutions? The content of plutonium and the acid in an aqueous plutonium solution is commonly measured and represented in plutonium concentration and acid molarity at Los Alamos National Laboratory. Where concentration is the weight in grams of the solute per liter of solution and molarity is the moles of the solute per liter of solution. On the other hand, all the methods discussed above use the input units of molality. To convert between units of molality and concentration it requires knowing the solution density.

Additionally, plutonium nitrate is a ternary solution ($Pu(NO_3)_4/HNO_3/H_2O$) and cannot exist in a true binary solution($Pu(NO_3)_4/H_2O$). However, a paper by Charrin [2] has regressed plutonium nitrate ternary data to a fictive binary solution of the plutonium nitrate in water. This data is limited to 25 °C and plutonium molality of 0.86 to 3 moles/kg. Since the isopiestic method requires the binary data, this approach would have to be used to calculate the density with the isopiestic method. However, while the Pitzer method requires the binary data to compute the coefficients, it does not require the binary data to calculate the density if the coefficients are known. Weber [6], calculates these coefficients using regressed pseudo-

data from the discussed empirical formula.

Another consideration was the expansion of this tool for aqueous plutonium chloride solutions. Simultaneous research is being performed by the University of New Mexico and Los Alamos National Laboratory to perform chlorine experiments and build a similar tool for aqueous plutonium chloride solutions. Ultimately, the empirical formula was selected for implementation into a Python tool with a possibility of expanding the tool to include the Pitzer method.

# CHAPTER 3 – Calculations

To implement this density law into a Python code which will be effective, easy to use and efficient for users, additional calculations will be needed. Because nuclear criticality safety staff at Los Alamos National Laboratory frequently use MCNP for criticality calculations, a feature which will create MCNP material card inputs was added. Therefore, the code takes multiple plutonium and acid content input units of concentration, molality, or molarity to calculate density. Then calculates number densities for each atom present in the solution for an MCNP material card input.

## Section 3.1. Density Calculations

The selected density law is shown below in terms of the density of water, $\rho_{H_2O}$, the molality of plutonium, $m_{Pu}$, the molality of the acid, $m_{Pu}$, and the temperature of the solution in Celsius, T [6].

$$\rho = \rho_{H_2O} + Am_{Pu} + Bm_H + CT + Dm_{Pu}m_H + Em_{Pu}T + Fm_{Pu}{}^2 + Gm_H{}^2 \tag{1}$$

Where:

$$A = 0.402234, \quad B = 0.029992, \quad C = 3.01282E-5, \quad D = -0.017735,$$

$$E = -4.7033E-4, \quad F = -0.026282, \quad G = -1.172E-3$$

As stated previously in Chapter 2, Los Alamos National Laboratory primarily uses units of concentration and molarity. Therefore, it was important to be able to convert between units easily. Their relationships can be shown below.

$$m_i = \frac{M_i}{C_{Solvent}}$$

(2)

Where m is the molality(mol/kg), M is the molarity(mol/L), W is the molar mass(g/mol) and C is the concentration(g/L). This equation applied to aqueous plutonium nitrate gives the following where the molarity of HNO3 is representing the free acidity of the acid in the solution.

$$m_{HNO_3} = \frac{1000 * M_{HNO_3}}{1000 * \rho_{Sol.\ Dens.} - W_{HNO_3} * M_{HNO_3} - W_{Pu(NO_3)_4} * M_{Pu(NO_3)_4}}$$

(3)

$$m_{Pu(NO_3)_4} = \frac{1000 * M_{Pu(NO_3)_4}}{1000 * \rho_{Sol.\ Dens.} - W_{HNO_3} * M_{HNO_3} - W_{Pu(NO_3)_4} * M_{Pu(NO_3)_4}}$$

(4)

The above equations provide a relationship to convert to molality from molarity but, it requires a known total solution density. Therefore, an iterative process was developed which would take an initial density guess, calculate the molality of plutonium and the acid based on that density and the input molarities, then recalculate density. It will then repeat using the new calculated density until the density converges within $10^{-4}$ %. Once a density is found then molality, molarity and concentration may be easily converted by the above equations or the below equations.

$$C_i = M_i * W_i$$

(5)

$$\frac{M_a}{M_b} = \frac{m_a}{m_b}$$

(6)

$$M_{HNO_3} = \frac{1000 * \rho_{Sol.\ Dens.} * m_{HNO_3}}{1000 + W_{HNO_3} * m_{HNO_3} + W_{Pu(NO_3)_4} * m_{Pu(NO_3)_4}}$$

(7)

$$M_{Pu(NO_3)_4} = \frac{1000 * \rho_{Sol.\ Dens.} * m_{Pu(NO_3)_4}}{1000 + W_{HNO_3} * m_{HNO_3} + W_{Pu(NO_3)_4} * m_{Pu(NO_3)_4}}$$

(8)

**Section 3.2. Material Card Calculations**

To begin calculations to find the MCNP® material card, the code first finds the solution density and the content inputs in units of molarity by the equations discussed in the previous section. Then the following equation will be used to find each of the number densities for each user specified isotope of plutonium.

$$N_{Pu_i} = wt.\%_i \frac{W_{Pu} * M_{Pu}}{W_{Pu_i}} * \frac{Av * 10^{-24}}{1000}$$

(9)

Where wt.%$_i$ is the weight percentage of that isotope in plutonium, $W_{Pu}$ is the total molar mass for plutonium, $W_{Pu,i}$ is the molar mass for the isotope, and Av is Avogadro's number. Then the number densities for the impurities may be calculated if the impurity concentration and molar mass.

$$N_{Imp_i} = \frac{C_{Imp_i}}{W_{Imp_i}} * \frac{Av * 10^{-24}}{1000}$$

(10)

Since $NO_3^-$ bonds with plutonium, hydrogen and the impurities, the nitrogen number density is calculated based on the plutonium molarity, acid molarity, and the number of bonds each impurity has with $NO_3^-$.

$$N_N = \left(4M_{Pu(NO_3)_4} + M_{HNO_3} + 3M_{Am_{241}} + \sum \frac{C_{Imp_i}}{W_{Imp_i}} * B_{Imp_i}\right) * \left(\frac{Av * 10^{-24}}{1000}\right)$$

(11)

The moles of water present must first be calculated before the number densities of hydrogen or oxygen. To calculate the moles of water, an assumption was made that once everything else has been accounted for, the rest must be water.

$$M_{H_2O} = \frac{1000\rho_{Sol.\ Dens.} - W_{HNO_3} * M_{HNO_3} - W_{Pu(NO_3)_4} * M_{Pu(NO_3)_4}}{W_{H_2O}}$$

(12)

Finally, the number densities for oxygen and hydrogen are shown below.

$$N_O = \left[M_{H_2O} + 3\left(4M_{Pu(NO_3)_4} + M_{HNO_3} + 3M_{Am_{241}} + \sum \frac{C_{Imp_i}}{W_{Imp_i}} * B_{Imp_i}\right)\right] * \left(\frac{Av * 10^{-24}}{1000}\right)$$

(13)

$$N_H = \left[M_{H_2O} + 3\left(4M_{Pu(NO_3)_4} + M_{HNO_3} + 3M_{Am_{241}} + \sum \frac{C_{Imp_i}}{W_{Imp_i}} * B_{Imp_i}\right)\right] * \left(\frac{Av * 10^{-24}}{1000}\right)$$

(14)

# CHAPTER 4- IMPLEMENTATION INTO PYTHON

These calculations have been implemented into Python to use multiple input units and various plutonium isotopes and impurity concentrations to find the solution density and material card. An additional feature was also added to allow users to add their MCNP6.2 input file directly to have the code place the density and material card into the input file.

To fit the needs of all users, the Python tool was separated into two codes: PuNSDensityFunctions for users with Python programming knowledge to run batches of data and PuNSDensitySolver for easy and quick individual density calculations. Both codes have the same equations and produce identical results, the differences are in user interface.

For both codes, a user would choose a folder on their computer to save the codes into. Once saved, any additional files in this folder will be able to be called without having to specify file paths. Also, neither codes use any Python modules or packages, so the only requirement for running these codes is Python. It is highly recommended however that a Python IDE be used such as SPYDER.

## Section 4.1. PuNSDensityFunctions User Interface

In the same folder PuNSDensityFunctions has been saved into, the user will create a new Python script and open with an IDE. At the beginning of the script file, the user will import PuNSDensityFunctions to call any of the functions as shown.

$import$ PuNSDensityFunctions

PuNSDensityFunctions.$Density1(mPu, mH, T)$

PuNSDensityFunctions is split into 15 functions, of which the first 11 are for density

calculations only and the other four solution functions are for density plus material card calculations. All are limited to a plutonium molality less than 3.3 and an acid content less than 6.27. If the inputs are not in the range of the density law, each function will return an error and a 0 zero density.

These functions were designed with the intention for users to harness them in numerous methods. Whatever data set and purpose the user has will affect how these functions get used. Therefore, they were made to be flexible to a wide range of scenarios.

**Section 4.2. PuNSDensityFunctions Density Functions**

Table 1 displays the function names and their inputs for all density functions. All inputs must be entered as floats or integers when using these functions and will return density as a float.

| ***Function*** | **Input 1** | **Input 2** | **Input 3** |
|---|---|---|---|
| *Density1* | Plutonium Molality (mol/kg) | Acid Molality (mol/kg) | Temperature (°C) |
| *Density2* | Plutonium Molality (mol/kg) | Acid Molarity (mol/L) | Temperature (°C) |
| *Density3* | Plutonium Molality (mol/kg) | Acid Concentration (g/L) | Temperature (°C) |
| *Density4* | Plutonium Molarity (mol/L) | Acid Molarity (mol/L) | Temperature (°C) |
| *Density5* | Plutonium Molarity (mol/L) | Acid Concentration (g/L) | Temperature (°C) |
| *Density6* | Plutonium Molarity (mol/L) | Acid Molality (mol/kg) | Temperature (°C) |
| *Density7* | Plutonium Concentration (g/L) | Acid Concentration (g/L) | Temperature (°C) |
| *Density8* | Plutonium Concentration (g/L) | Acid Molarity (mol/L) | Temperature (°C) |
| *Density9* | Plutonium Concentration (g/L) | Acid Molality (mol/kg) | Temperature (°C) |

Table 4.1. PuNSDensityFunctions density functions summary

An example showing a density calculation given a solution which contains 100 g/L

plutonium and 2 moles/L acid at 25°C is shown below. The script is on the top and the returned lines from the IPython Console are on the bottom.

```
1
2   import PuNSDensityFunctions as PUNS
3
4   CPu=100
5   H=2
6   T=25
7
8   print(PUNS.Density8(CPu, H, T))
```

```
In [9]: runfile('C:/Users/taral/Documents/Thesis/Thesis/Examples.py', wdir='C:/
Users/taral/Documents/Thesis/Thesis')
Reloaded modules: PuNSDensityFunctions
1.2184983711302875
```

Figure 4.1. Density calculation example

Additionally, these functions may be used to vary the input values as shown in the second example below.

```
1
2   import PuNSDensityFunctions as PUNS
3
4   CPu=[50, 100, 150]
5   H=[1.5, 2, 2.5]
6   T=25
7
8   for i in range(0,len(CPu)):
9       for j in range(0,len(H)):
10          print(PUNS.Density8(CPu[i], H[j], T))
11
```

```
In [10]: runfile('C:/Users/taral/Documents/Thesis/Thesis/Examples.py',
wdir='C:/Users/taral/Documents/Thesis/Thesis')
Reloaded modules: PuNSDensityFunctions
1.1234929921627903
1.1374686173517627
1.1511086495817417
1.2051470792444663
1.2184983711302875
1.2314024080363968
1.2870164493331295
1.299639655140221
1.3116846247367
```

Figure 4.2. Varied input density calculation example

**Section 4.3. PuNSDensityFunctions Density Editor Functions**

The density editor functions are designed for the user to enter a text file and a new file name. These should be formatted as shown below.

$$newfile = 'newfile.txt'$$

The functions will read the text file and then write a new text file which includes the solution density. The users will place an "@" symbol in the desired density location in the input text file and the editor will place the density there and the material card after the material number in the data card section in the new file. The editor function will not return the density into Python. Since the functions were designed for a user using them for MCNP6.2 calculations, it will skip any commented lines that begin with "c ". Additionally, the editor functions and solution functions only take two unit input options, the most common input at Los Alamos National Laboratory and the density law input units. The table below shows the function inputs.

| ***Function*** | **Input 1** | **Input 2** | **Input 3** | **Input 4** | **Input 5** |
|---|---|---|---|---|---|
| *DensityEditor1* | Input file | New file | Plutonium Molality (mol/kg) | Acid Molality (mol/kg) | Temperature (°C) |
| *DensityEditor2* | Input file | New file | Plutonium Concentration (g/L) | Acid Molarity (mol/L) | Temperature (°C) |

Table 4.2. PuNSDensityFunctions density editor functions summary

For the same solution as in the first example, 100 g/L plutonium and 2 moles/L acid

at 25°C, a MCNP input file, a Python script, and the new file is shown below. Nothing returns in the IPython Console therefore it is not shown.

```
MCNP Input Example
10      100     @           -1    u=1           imp:n=1
20      0                    2    u=1           imp:n=1
```

```python

import PuNSDensityFunctions as PUNS

CPu=100
H=2
T=25

inputfile="editthis1.txt"
newfile="newfile.txt"

PUNS.DensityEditor2(inputfile, newfile, CPu, H, T)
```

```
MCNP Input Example
10 100 -1.2184983711302875 -1 u=1 imp:n=1
20      0                    2    u=1           imp:n=1
```

Figure 4.3. Example of editing a MCNP file

**Section 4.4. PuNSDensityFunctions Solution Functions**

To calculate the material card number densities, the solution functions require the plutonium weight percents and impurities to be entered as a list of lists. The plutonium weight percent list should include each isotopes ZAID number followed by the isotope weight percent. This list may include any number of the following: plutonium-238, plutonium-239, plutonium-240, plutonium-241, plutonium-242, or americium-241. The impurity list should include, for each impurity, the impurity ZAID number, the concentration, the molar mass, and the number of bonds it will make with $NO_3^-$. The list may contain any number of impurities less than 11. However, it should be noted that a large concentration of impurities will affect the accuracy of the density calculation.

The code will return a tuple of the density as a float and the material card in a

dictionary. The dictionary keys will be the ZAID number for the atom the number density is for. The functions are listed in table 3.

| ***Function*** | **Input 1** | **Input 2** | **Input 3** | **Input 4** | **Input 5** |
|---|---|---|---|---|---|
| *Solution1* | Plutonium Molality (mol/kg) | Acid Molality (mol/kg) | Plutonium List | Impurity List | Temperature (°C) |
| *Solution2* | Plutonium Concentration (g/L) | Acid Molarity (mol/L) | Plutonium List | Impurity List | Temperature (°C) |

Table 4.3. PuNSDensityFunctions solution functions summary

An example showing the calculation of the material card and density for a solution with 2% plutonium-240, 100 g/L plutonium, 2 moles/L acid, and no impurities at 25°C is shown below.

```
1
2   import PuNSDensityFunctions as PUNS
3
4
5   CPu=100
6   H=2
7   T=25
8
9   wPu=[[94239,0.98],[94240,0.02]]
10  Imp=[]
11
12  S=PUNS.Solution2(CPu, H, wPu, Imp, T)
13
14  print(S[0])
15  print("")
16  print(S[1])
```

```
In [14]: runfile('C:/Users/taral/Documents/Thesis/Thesis/Examples.py',
wdir='C:/Users/taral/Documents/Thesis/Thesis')
Reloaded modules: PuNSDensityFunctions
1.2184983711302875

{94239: 0.0002468733146408552, 94240: 5.01720835397117e-06, 1001:
0.060621785153493354, 7014: 0.002212046182207572, 8016: 0.0363448311233694}
```

Figure 4.4. Density and material card example calculation

## Section 4.5. PuNSDensityFunctions Solution Editor Functions

The solution editor functions are a combination of all of the above functions. The table below lists the required inputs.

| ***Function*** | **Input 1** | **Input 2** | **Input 3** | **Input 4** | **Input 5** | **Input 6** | **Input 7** |
|---|---|---|---|---|---|---|---|
| *Solution1* | Input file | New file | Plutonium Molality (mol/kg) | Acid Molality (mol/kg) | Plutonium List | Impurity List | Temperature (°C) |
| *Solution2* | Input file | New file | Plutonium Concentration (g/L) | Acid Molarity (mol/L) | Plutonium List | Impurity List | Temperature (°C) |

Table 4.3. PuNSDensityFunctions solution functions summary

An example is shown below given a solution with 100 g/L plutonium and 2 moles/L acid at 25°C. Additionally the plutonium isotopes have weight percents of 75% plutonium-239, 19% plutonium-240, 5% plutonium 241, and 1% plutonium-242. There is also 2.3 g/L iron, 0.7 g/L nickel, and 0.5 g/L chromium in the solution. The input file is snipped to show the density and material card locations for the input file and the new file.

```
MCNP Input Example
10      100     @           -1    u=1            imp:n=1
m100
mt100  lwtr
```

```
import PuNSDensityFunctions as PUNS

CPu=100
H=2
T=25

wPu=[[94239,0.75],[94240,0.19],[94241,0.05],[94242,0.01],[95241,0.0061]]
Imp=[[26000,2.3,55.847,3],[24000,0.7,51.9961,3],[28000,0.6,58.6934,2]]

inputfile="editthis1.txt"
newfile="newfile.txt"

PUNS.SolutionEditor2(inputfile, newfile, CPu, H, wPu, Imp, T)
```

```
MCNP Input Example
10 100 -1.2184983711302875 -1 u=1 imp:n=1
m100 94239 0.0001889336591639198
     94240 4.766347936272613e-05
     94241 1.2490829357330395e-05
     94242 2.487825860095312e-06
     95241 1.5238813244639995e-06
     1001 0.060621785153493354
     7014 0.0023276543048082426
     8016 0.036691655491171406
     26000 2.480097408992425e-05
     24000 8.107146497525776e-06
     28000 6.156058432464297e-06
mt100  lwtr
```

Figure 4.5. Material card and density file editing example

**Section 4.6. PuNSDensitySolver User Interface**

PuNSDensitySolver was developed with the intention to be simple to use. Once the PuNSDensitySolver Python script has been saved onto the user's computer, the user will create a new file and type the line shown below.

$import\ Code1$

Then the Python script is ready to run. Rather than having rigid inputs, like PuNSDensityFunctions, PuNSDensitySolver will instead ask the user a series of questions through the built in Python input function. The answers to the questions should be typed directly into the IPython Console. Questions will require either be a yes or no answer, a selection out of a list of options, an integer or float input, or a file input. Based on the answers, it will either skip to a specific question, print a result, end the script, or print an error.

An example of calculating the density for a solution with 100 g/L plutonium and 2 moles/L acid at 25 °C is shown below.

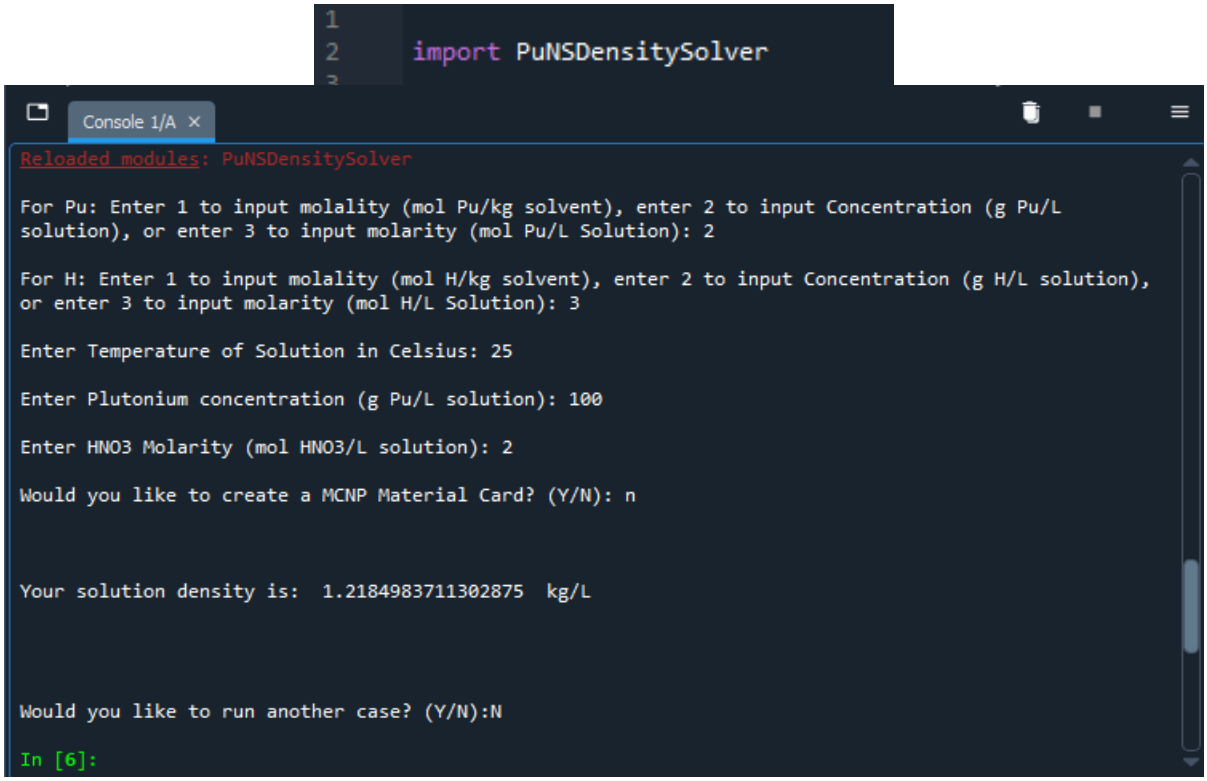

Figure 4.6. Density calculation example

An additional example is shown for the same case but, this time calculating the material card given 5% plutonium-240 and adding it to an MCNP file.

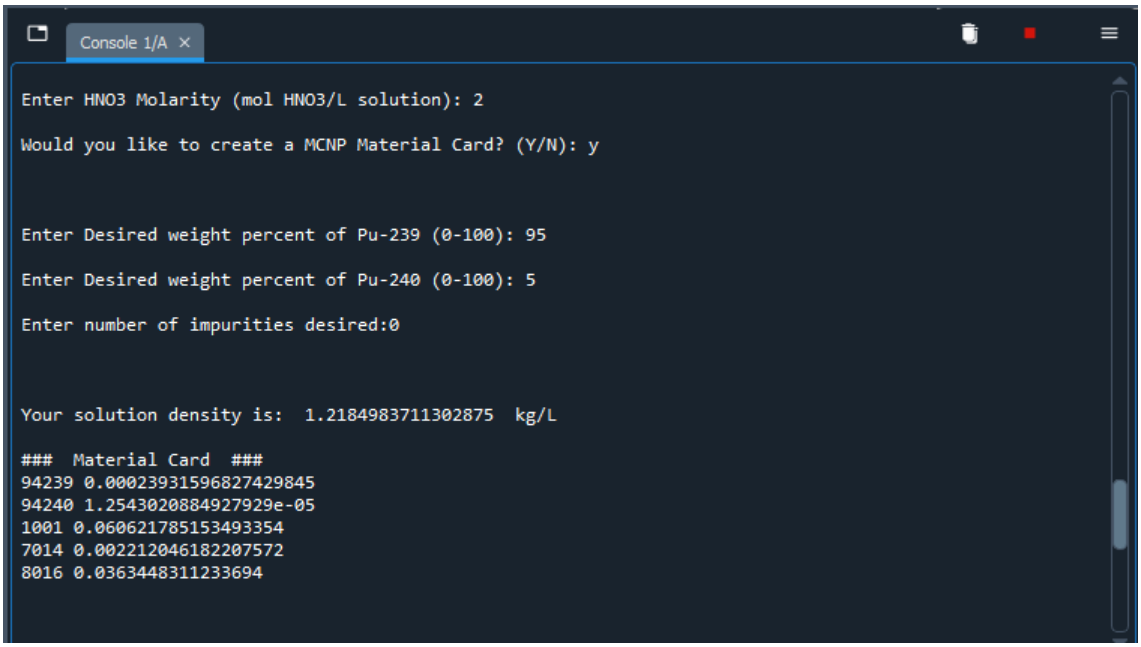


Figure 4.7. Material card calculation example

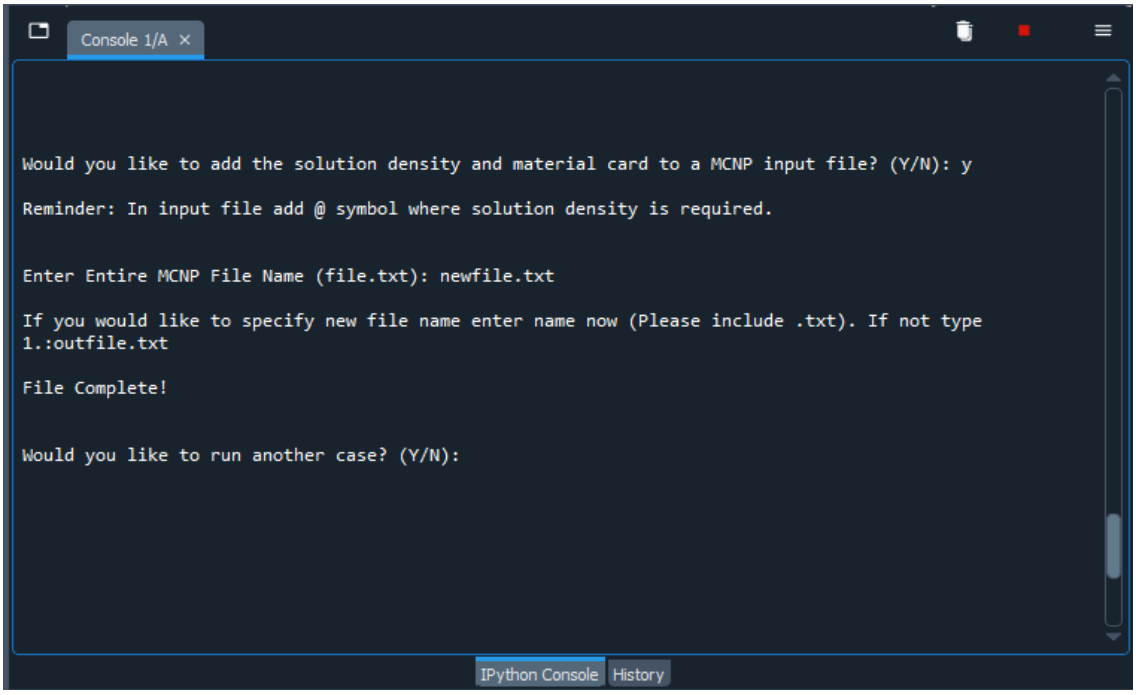


Figure 4.8. MCNP input file editing example

Next an example is shown for running another case and changing to temperature to 27 ℃ and adding 2.2 g/L of iron.

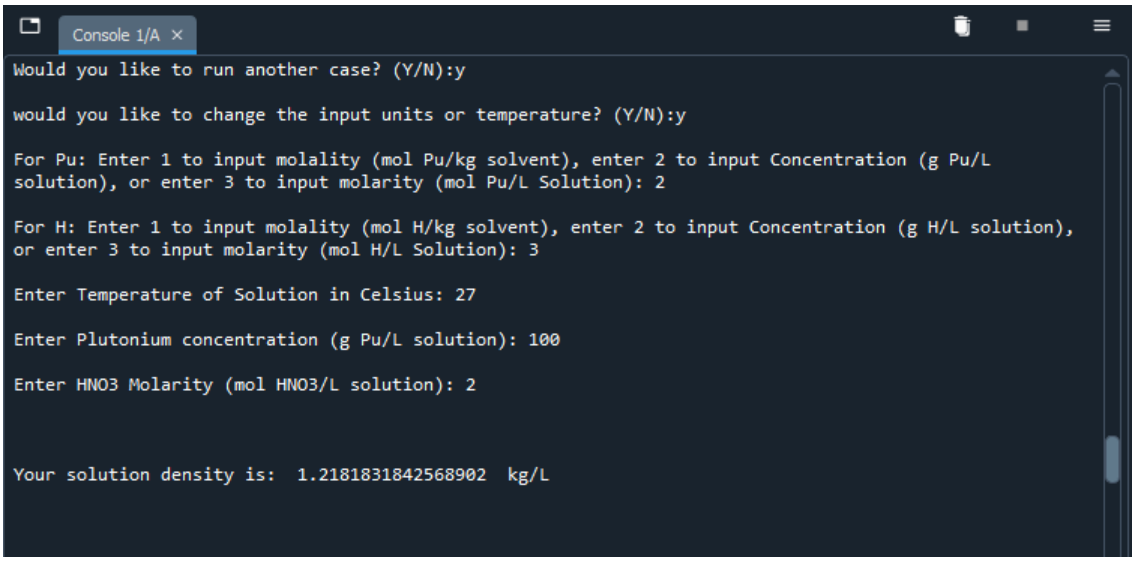


Figure 4.9. Example of running a new case with a different temperature

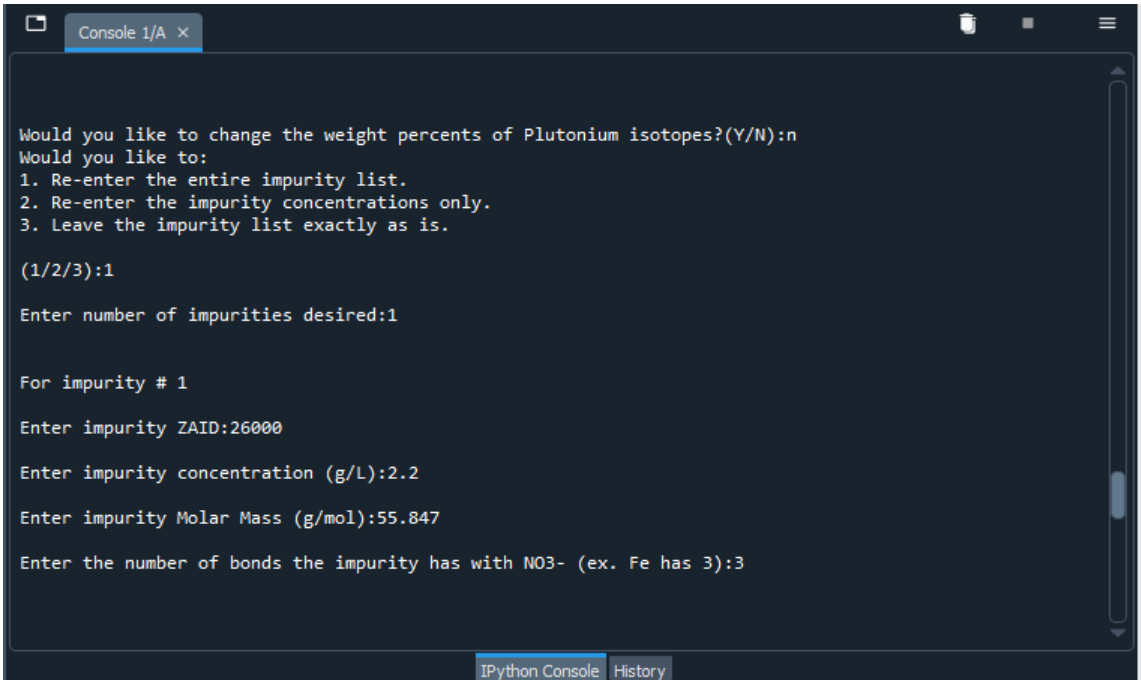


Figure 4.10. Adding an impurity example

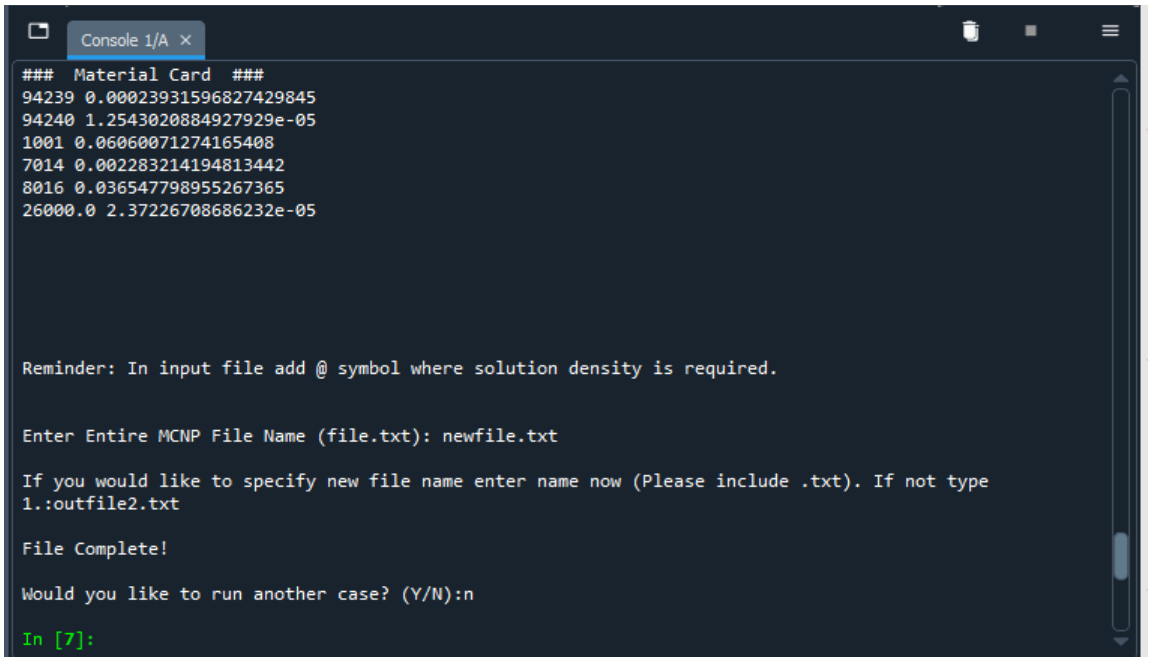


Figure 4.11. Results and editing MCNP input file example

# CHAPTER 5 – VERIFICATION AND VALIDATION

The Python tool was validated against the International Handbook of Evaluated Criticality Safety Benchmark Experiments (ICSBEP) and SCALE fissile solution composition card. A total of 46 cases were selected from ICSBEP benchmarks PU-SOL-THERM-001, PU-SOL-THERM-007, PU-SOL-THERM-012, PU-SOL-THERM-020, and PU-SOL-THERM-025. From each benchmark, multiple cases were reviewed and the experimental measurements for plutonium concentration and acid molarity were used in various functions in the code. The resulting density was then compared to the experimental density listed in the benchmarks. Additionally, in some cases the listed calculated atom number densities in the benchmarks were compared to the calculated number densities from the tool. Additionally, in order to understand how the density calculation relates to k-effective, a subset of the selected benchmarks was modeled using MCNP6.2 and the calculated k-effective from the calculated density compared to the calculated k-effective from the experimental density. The data distribution is shown below for all considered cases.

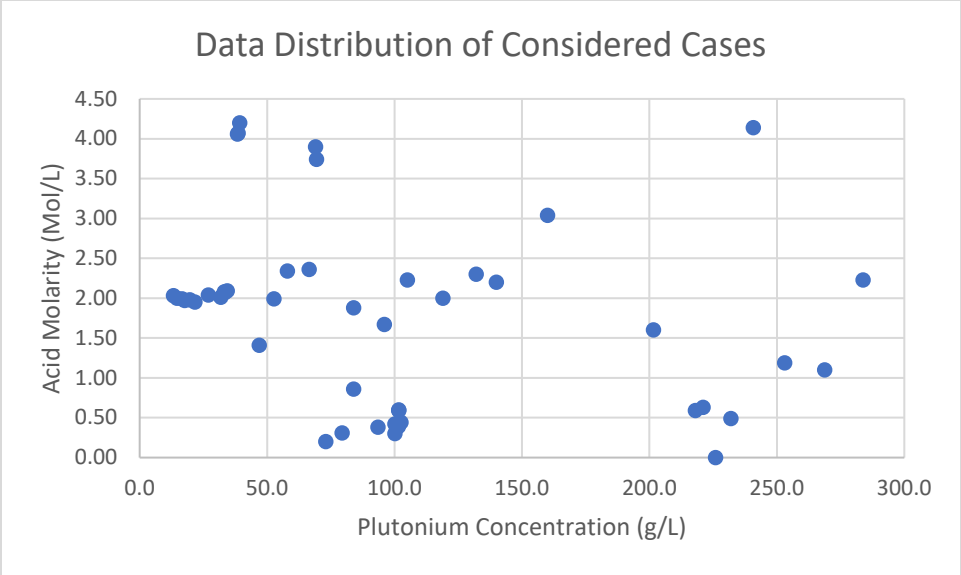


Figure 5.1. Data distribution of total considered cases

Of these cases, every function in each code of the Python tool was ran against 11 total cases. Each function with the same input units returned exactly the same results. Additionally, when each combination of units was tested it was found that the functions with molality plutonium and molality acid inputs returned the worst fit to density and the plutonium concentration or molarity and acid concentration or molarity inputs returned the best fit to density. Therefore, our two limiting functions were chosen to be PuNSDensityFunctions Density1 and Density8 for the density calculations. Conveniently, this aligns with the two unit input options chosen for all of the other function options.

**Section 5.1. Density Error Summary**

When compared to the benchmark data, the tool had a ± 2.6% error in predicting the density of aqueous plutonium nitrate for PuNSDensityFunctions Density1 and a ± 2.2% error in predicting the density of aqueous plutonium nitrate for PuNSDensityFunctions Density8. This can be seen in the following figures comparing the error in density versus either the acid or plutonium content inputs.

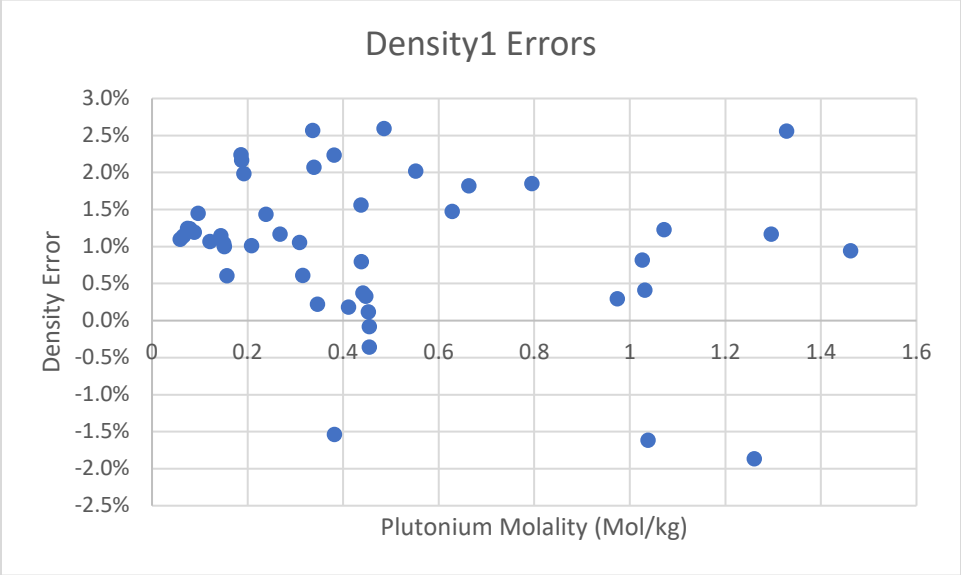


Figure 5.2. PuNSDensityFunctions Density1 error versus plutonium molality input

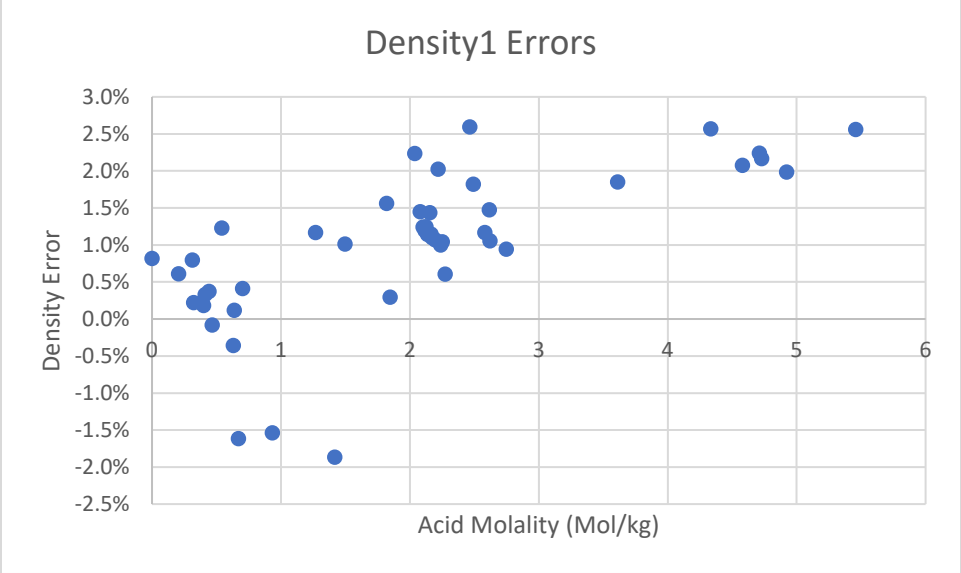


Figure 5.3. PuNSDensityFunctions Density1 error versus acid molality input

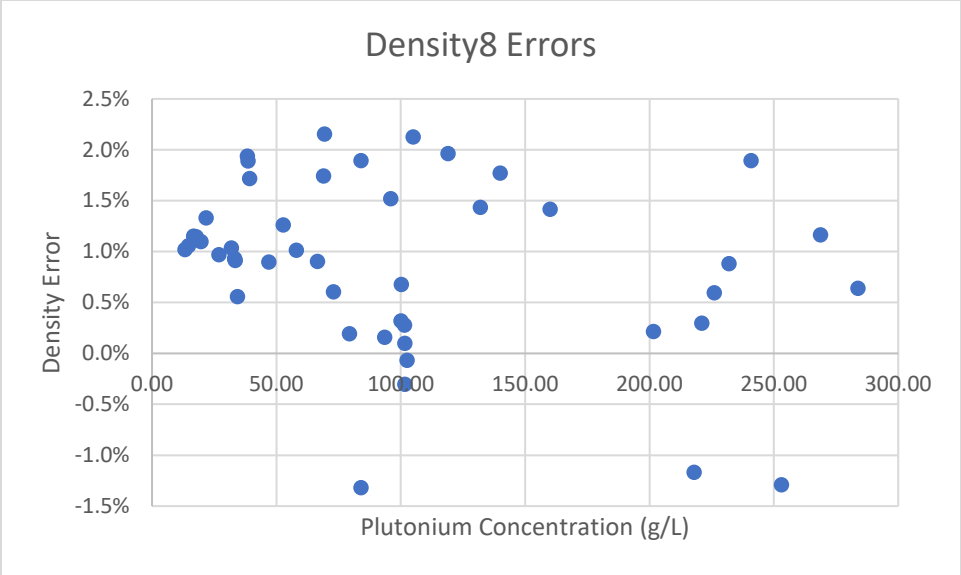


Figure 5.4. PuNSDensityFunctions Density8 error versus plutonium concentration input

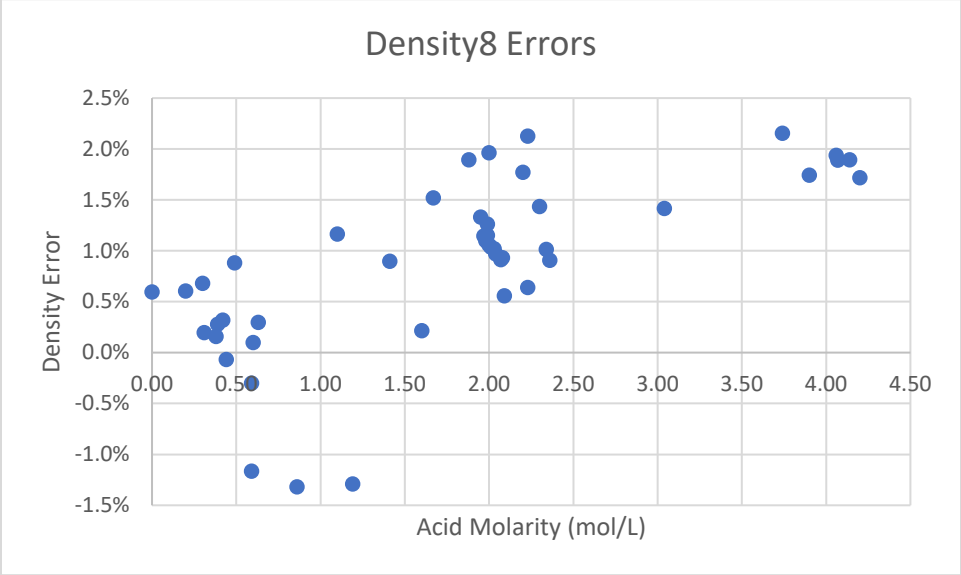


Figure 5.5. PuNSDensityFunctions Density8 error versus acid molarity input

In Figure 5.3. and Figure 5.5. a loose correlation of error with acid content may be observed. However, there appears to be no correlation with plutonium concentration.

**Section 5.2. SCALE Density Error Summary**

The SCALE fissile solution composition card takes inputs of plutonium

concentration, plutonium isotope weight percents, acid molarity, and temperature. However, it uses the Pitzer method to calculate density. Therefore, it was compared to PuNSDensityFunctions Soluion2 for both the density and the material card. The tool had a ± 2.3% error when compared to SCALE. Additionally, the correlation to acid content is seen in Figure 5.7.

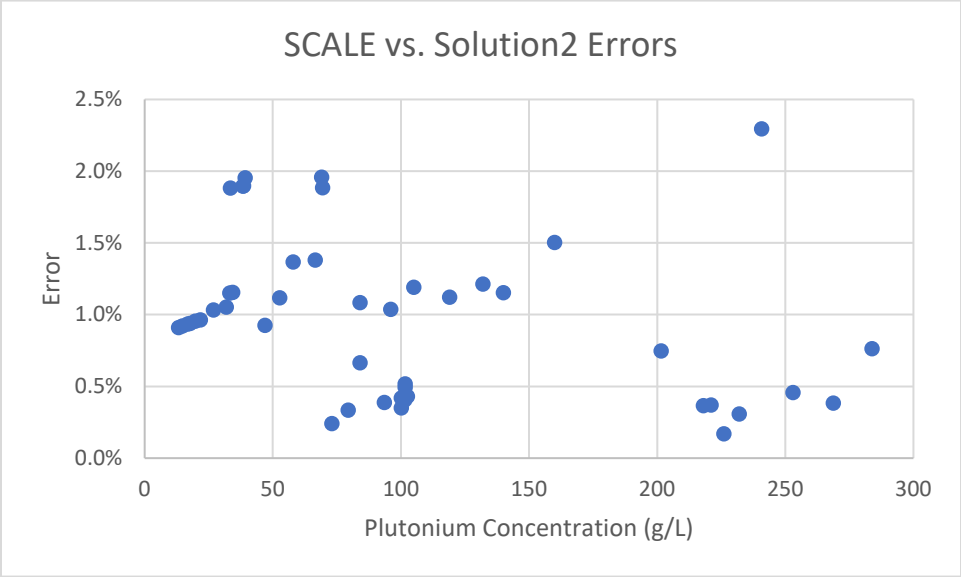


Figure 5.6. SCALE versus PuNSDensityFunctions Solution2 error versus plutonium concentration input

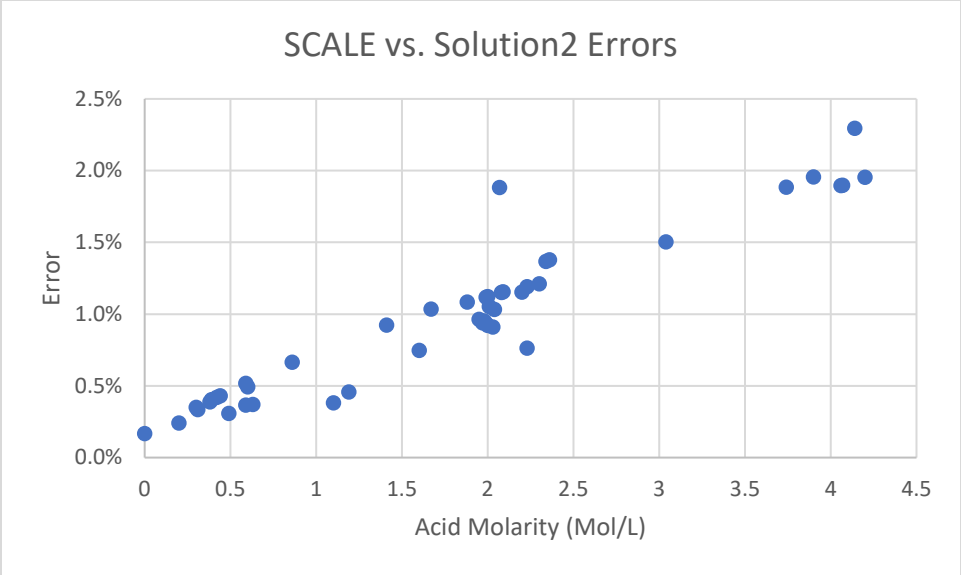


Figure 5.7. SCALE versus PuNSDensityFunctions Solution2 error versus acid molarity input

**Section 5.3. Material Card Error Summary**

The material card results from PuNSDensityFunctions Solution1 and PuNSDensityFunctions Solution2 were compared against the listed number densities in the ICSBEP benchmarks for 32 of the 46 cases. The data distribution is shown below in figure 5.8.

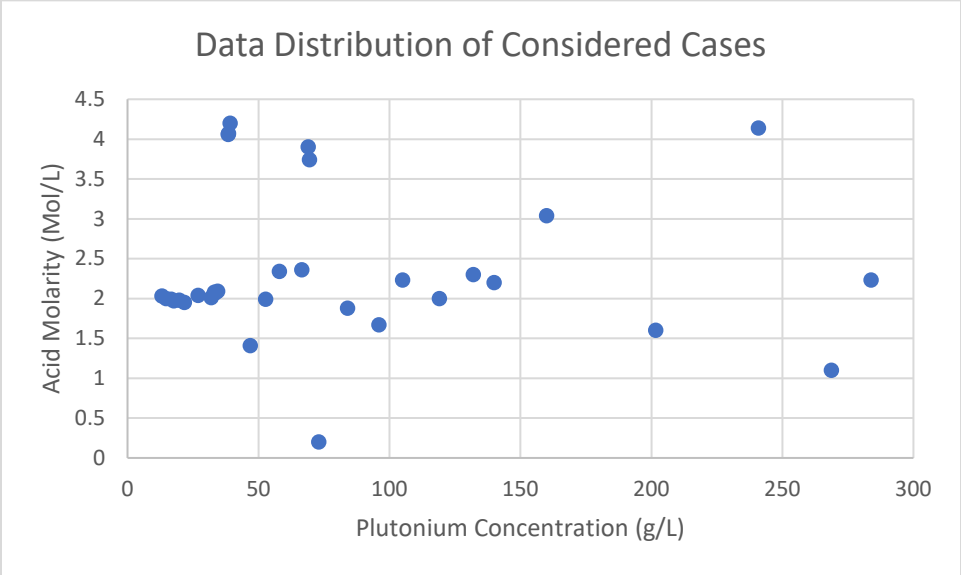


Figure 5.8. Data distribution of considered cases for the material cards

The error results for each atom in the material card is shown below for Solution1 and Solution2. It should be noted that only PU-SOL-THERM-012 included impurity concentration.

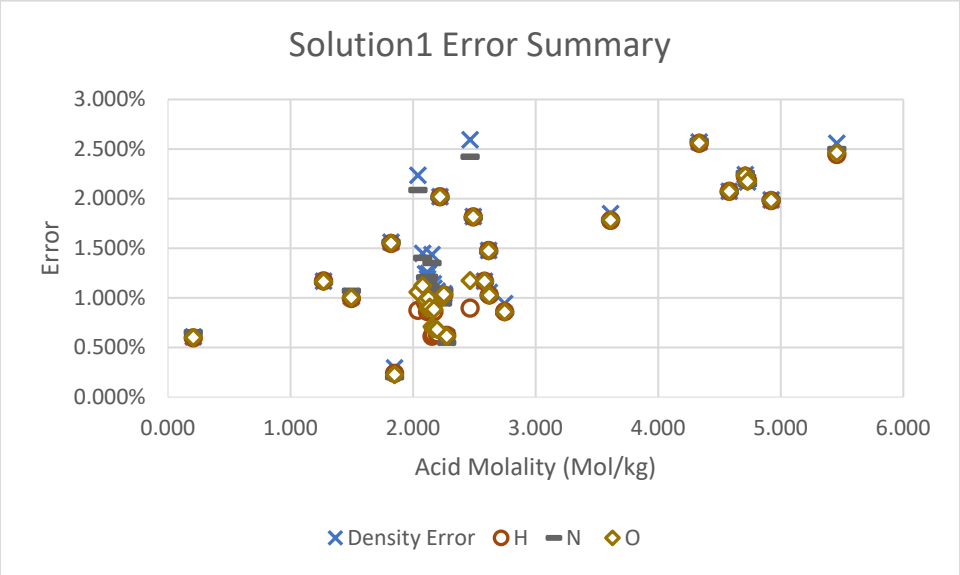


Figure 5.9. Solution1 errors for hydrogen, nitrogen, and oxygen

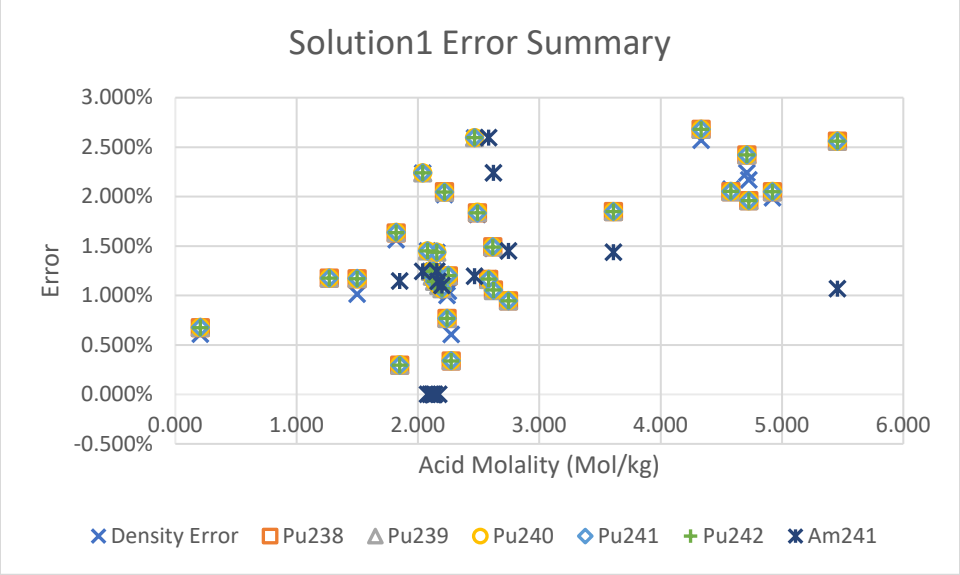


Figure 5.10. Solution1 errors for plutonium isotopes

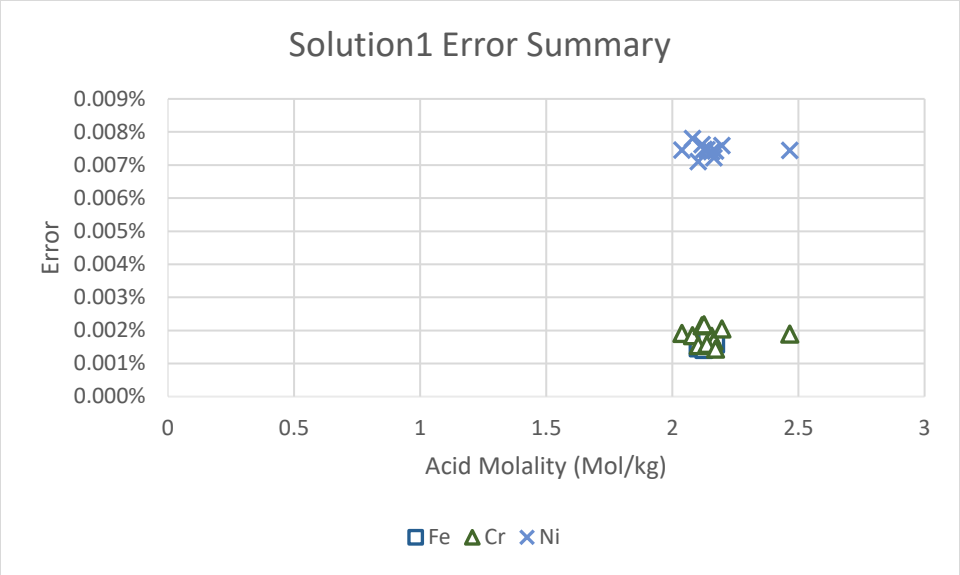


Figure 5.11. Solution1 errors for impurities

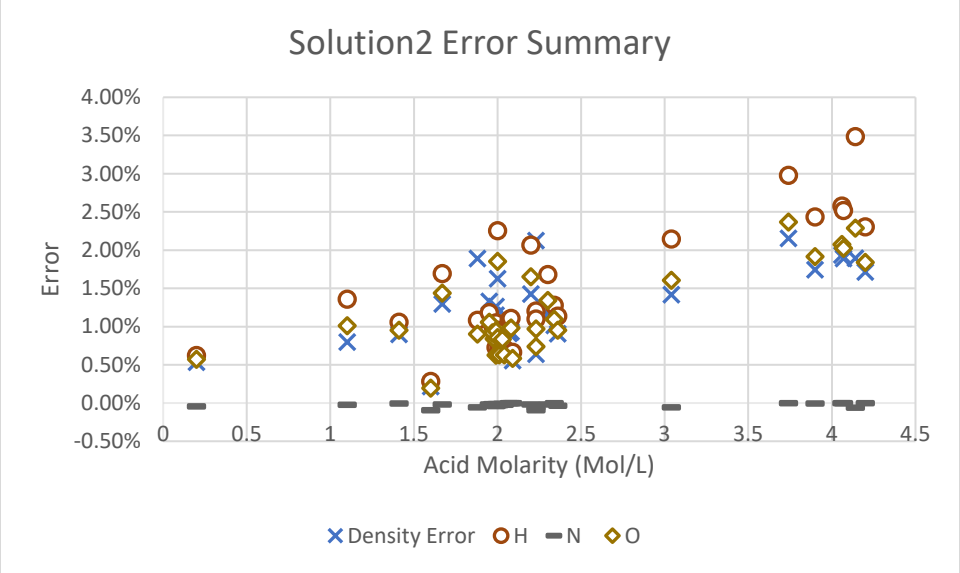


Figure 5.12. Solution2 errors for hydrogen, nitrogen, and oxygen

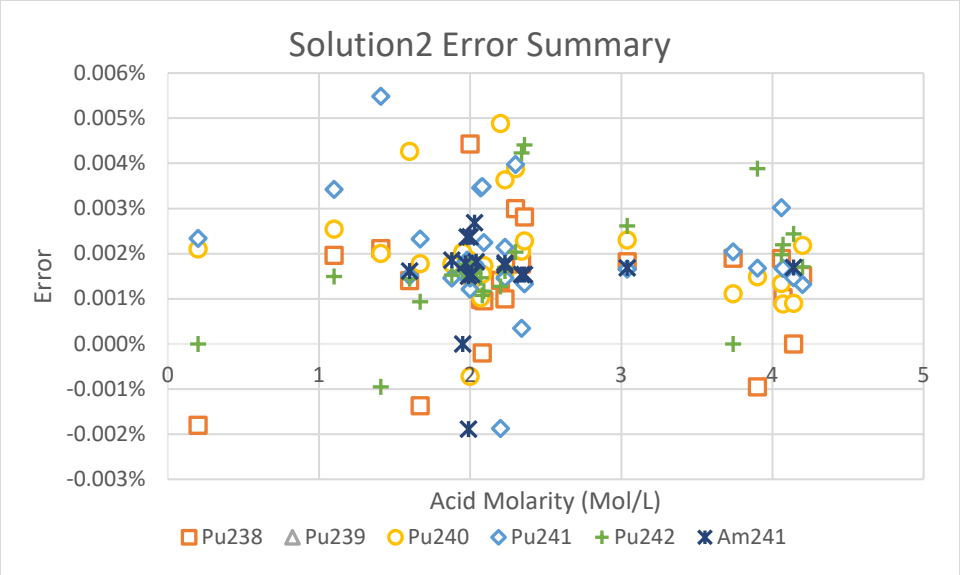


Figure 5.13. Solution2 errors for plutonium isotopes

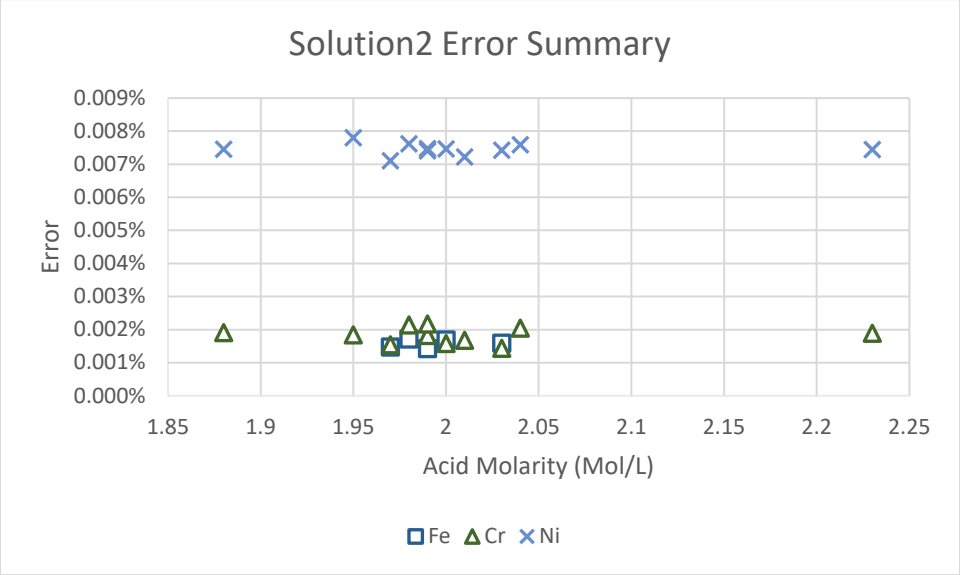


Figure 5.14. Solution2 errors for impurities

In Solution1, the errors in the plutonium isotope number densities are directly affected by the errors in density. This is due to a multiplication by density to find the number densities from the inputted units of molality. This step is not required when molarity and concentration are the inputs. Additionally, in both Solution1 and Solution2 functions,

hydrogen and oxygen errors are affected by the density errors due to the assumption that anything left in the solution is water. The effects on hydrogen and oxygen content are much more amplified for the Solution2 function with the largest error being up to 3.5% rather than 2.6% for Solution1. However, the plutonium isotope number densities see a large reduction in error down to under 0.0009% for Solution2 rather than 2.7% for Solution1.

**Section 5.4. SCALE Material Card Error Summary**

Since the SCALE fissile solution composition card does not consider impurities, the cases from PU-SOL-THERM-012 will look a little different. This is most clearly shown through the nitrogen content. For the material card, SCALE was again only compared to Solution2.

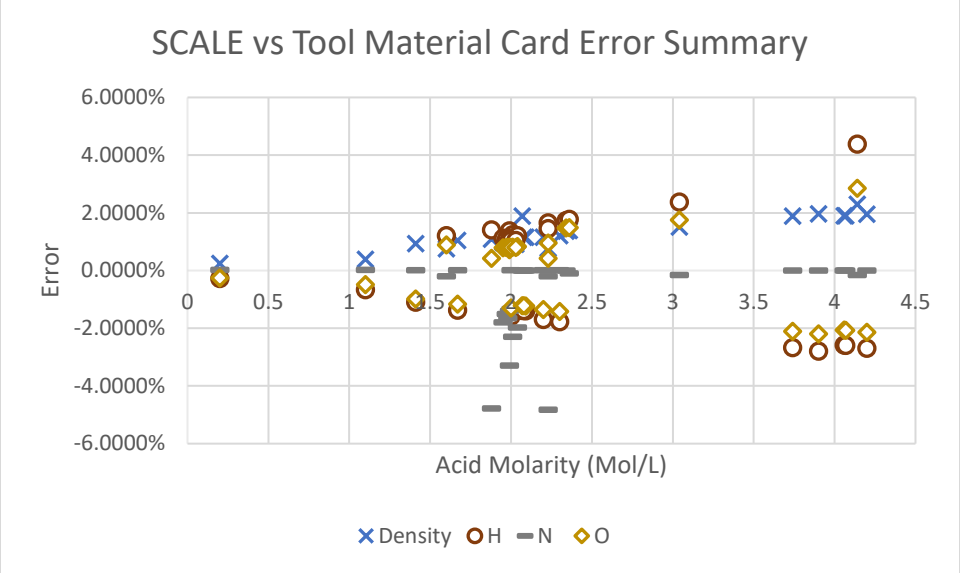


Figure 5.15. Solution2 versus SCALE errors for hydrogen, nitrogen, and oxygen

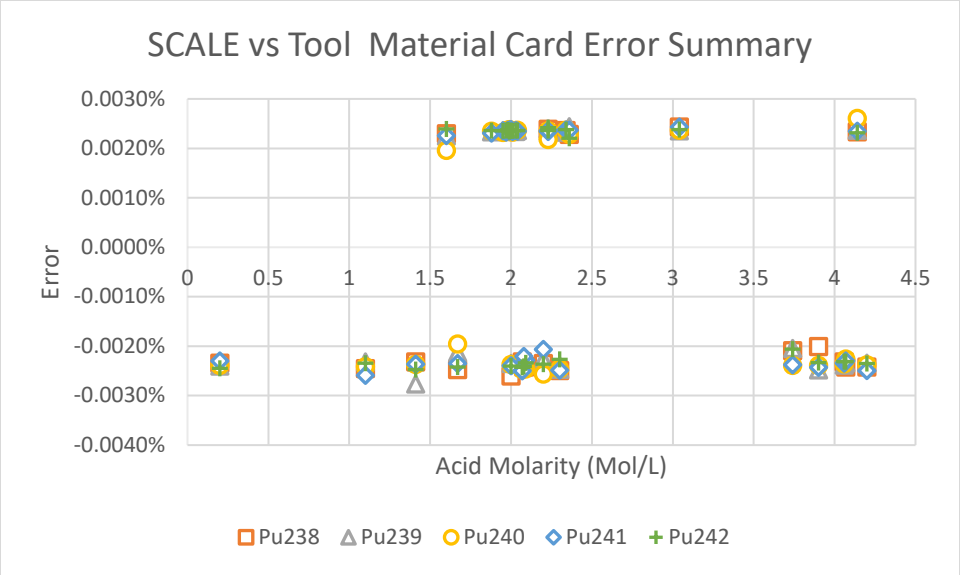


Figure 5.16. Solution2 versus SCALE errors for plutonium isotopes

The fit for the plutonium isotope number densities are within ± 0.003%. However, a large deviation from the SCALE number densities was found for hydrogen and oxygen content. The nitrogen content only varies for benchmark 12 which is due to impurities in the solution. The largest error observed for hydrogen and oxygen number densities was 4.4%.

**Section 5.5. MCNP6.2 Validation**

The affect the calculated densities from the plutonium concentration and acid molarity specified in the ICSBEP Handbook compared to the specified densities from the Handbook had on the k-effective value was quantified by running a few cases through MCNP6.2. The MCNP6.2 input cards were taken from the Whisper-1.1 benchmark library and compared to the input file generated from SolutionEditor2. The cases selected, were from benchmark PU-SOL-THERM-001, PU-SOL-THERM-007, PU-SOL-THERM-012, and PU-SOL-THERM-020. Benchmark PU-SOL-THERM-025 was not considered due to the complexity of the geometry and the MCNP6.2 input files not being verified by Los Alamos National Laboratory. However, this benchmark was the primary source of high acid content

and high plutonium content data points. The considered cases are shown below.

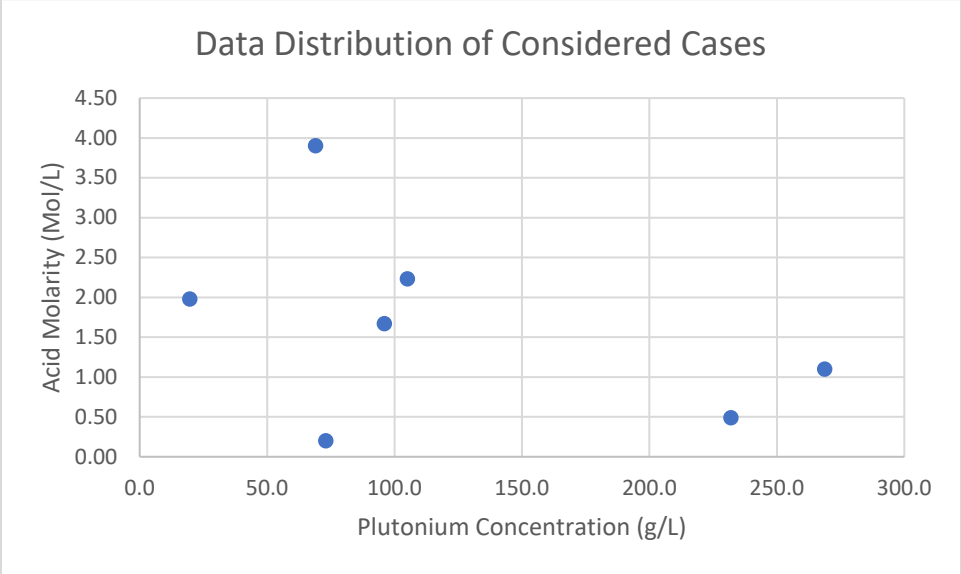


Figure 5.17. Data distribution of cases validated in MCNP6.2

The results of the validation are shown in the table below.

| *Benchmark* | *Plutonium Concentration* | *Acid Molarity* | *Benchmark Density* | *Benchmark Keff* | *Benchmark σ* | *Found Density* | *Found Keff* | *Found σ* | *Density Percent Error* | *Difference in Keff in pcm* | *Keff Percent Error* | *Difference in σ* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **001-001** | 73 | 0.2 | 1.13 | 1.00612 | 0.00087 | 1.12398 | 1.00224 | 0.00091 | 0.53% | 388 | 0.39% | -0.03177 |
| **001-002** | 96 | 1.67 | 1.219 | 1.00857 | 0.00092 | 1.20320 | 0.99861 | 0.0009 | 1.30% | 996 | 0.99% | 0.01554 |
| **001-006** | 268.7 | 1.1 | 1.484 | 1.00931 | 0.001 | 1.47215 | 1.00081 | 0.00102 | 0.80% | 850 | 0.84% | -0.014 |
| **007-002** | 232 | 0.49 | 1.409 | 1.0093 | 0.00013 | 1.39659 | 1.00206 | 0.00013 | 0.88% | 724 | 0.72% | 0 |
| **012-001** | 105 | 2.23 | 1.259 | 1.00818 | 0.00013 | 1.23223 | 0.99608 | 0.00013 | 2.13% | 1210 | 1.20% | 0 |
| **012-007** | 19.7 | 1.98 | 1.1 | 1.00553 | 0.00009 | 1.08792 | 1.00405 | 0.00009 | 1.10% | 148 | 0.15% | 0 |
| **020-015** | 69 | 3.9 | 1.238 | 0.99698 | 0.00112 | 1.21642 | 0.98019 | 0.00107 | 1.70% | 1679 | 1.68% | 0.03228 |

Table 5.1. Comparison of benchmark versus calculated keff.

For the data that was validated in MCNP6.2 against the Whisper 1.1 benchmark library cards, an observed effect on k-effective error from calculated density was reduced from the error in density for lower plutonium concentrations. As plutonium concentration grows, in case 001-006, there is a slight increase in k-effective error. This is shown in the figure below.

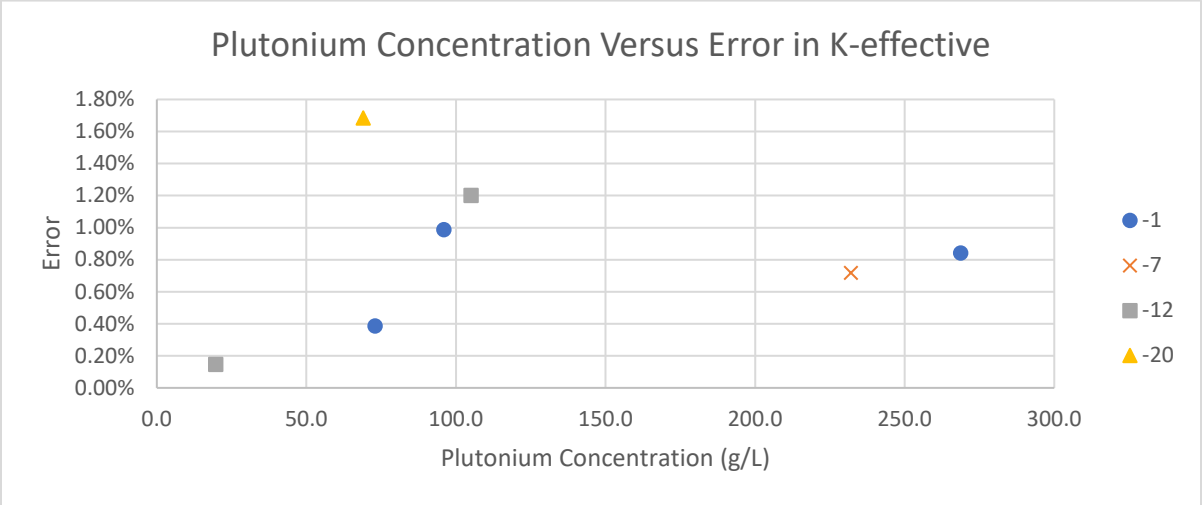


Figure 5.18. Plutonium concentration versus error in k-effective

However, when compared to the benchmark experimental k-effective of 1.000±0.005, every result returned within this range except for the case from benchmark 020. These variations are consistent with the selected method for density and k-effective calculations. The tool will work best over lower acid and plutonium concentration ranges but begin to deviate at high concentrations of either acid or plutonium.

# CHAPTER 6 – CONCLUSIONS

In summary, a Python tool was developed for plutonium nitrate density calculations to assist engineers at Los Alamos National Laboratory. The complete tool is attached in Appendix A and B, which uses an empirical density law and multiple other calculations to provide solution densities, atom number densities, and a text file editor. There are two main codes for the tool PuNSDensityFunctions and PuNSDensitySolver. Both include the same equations with separate user interfaces.

The calculated error for the density calculations is within ± 2.6% when the tool was compared to the International Handbook of Evaluated Criticality Safety Benchmark Experiments (ICSBEP). The density error was observed to be loosely correlated with acid concentration and may be reduced if the selected input units for plutonium and acid content are changed to molarity or concentration. The material card atom number densities have various errors depending on the error in density, input units, acidity, and impurity content. The error is reduced for density at lower acidity concentrations therefore reducing the error in the material card and k-effective calculations. Additionally, the error in k-effective was observed to be reduced by low plutonium concentrations.

The selected density law is sufficient for the current work but, future work for this tool should include an addition of multiple methods such as the Pitzer method or Isopiestic method, which will reduce the errors in the density calculations and atom number density calculations.

# APPENDICIES

## Appendix A

```
"""
PuNSDensityFunctions
Updated: 03/14/23
@author: Tara
"""
######################### EDIT GUIDE ##############################################
"""
mX is molality (mol X/kg solvent)
MX is Molar Mass (g X/mol X)
X  is molarity (mol X/L solution)
CX is concentration (g X/L solution)
wX is weight percent in decimal form Ex. wX=0.9 or 90% weight percent
NX is atom Density (atoms/barn*cm) that will be entered into the MCNP material card
A,B,C,D,E,F,G are coefficients from Weber Paper

Density is always in kg/l
T is in Celsius


Files should be formatted: 'inputfile.txt'

"""
######################### EDIT GUIDE ##############################################

#############################################################################################
#############################################################################################
######################
"""SECTION 1"""
#############################################################################################
#############################################################################################
######################

def Density1(mPu,mH,T):
    """mPu and mH are in molality (mol solute/kg solvent) and T is in Celsius"""
    #Density Coefficients
    if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
        print("ERROR: Data entered is out of bounds of equation range.")
        return 0
    else:
        A=0.402234
        B=0.029992
        C=3.01282*(10**-5)
        D=-0.017735
        E=-4.7033*(10**-4)
        F=-0.026282
        G=-1.172*(10**-3)
        do=0.997

        return do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH

def Density2(mPu,H,T):
    """mPu is in molality (mol solute/kg solvent), H is in molarity (mol solute/L solution), and T is in Celsius"""
    #Density Coefficients
    A=0.402234
    B=0.029992
    C=3.01282*(10**-5)
    D=-0.017735
    E=-4.7033*(10**-4)
    F=-0.026282
    G=-1.172*(10**-3)
    do=0.997

    #Molar Masses
```

```
    MPu239=239.0521636
    MH=1.0078
    MO=15.999
    MN=14.007
    MPUNO34=MPu239+(MN+3*MO)*4
    MHNO3=MH+(MN+3*MO)

    error=1
    NewDens=1.2

    while error>0.000001:
        GuessDens=NewDens
        Pu=(1000*NewDens-MHNO3*H)*mPu/(1000+MPUNO34*mPu)
        mH=mPu*H/Pu
        Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
        error=(abs(Dens1-GuessDens))/(Dens1)
        NewDens=Dens1

    if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
        print("ERROR: Data entered is out of bounds of equation range.")
        return 0
    else:
        FinalDens=NewDens

        return FinalDens

def Density3(mPu,CH,T):
    """mPu is in molality (mol solute/kg solvent), CH is in concentration (g solute/L solution), and T is in Celsius"""

    #Density Coefficients
    A=0.402234
    B=0.029992
    C=3.01282*(10**-5)
    D=-0.017735
    E=-4.7033*(10**-4)
    F=-0.026282
    G=-1.172*(10**-3)
    do=0.997

    #Molar Masses
    MPu239=239.0521636
    MH=1.0078
    MO=15.999
    MN=14.007
    MPUNO34=MPu239+(MN+3*MO)*4
    MHNO3=MH+(MN+3*MO)

    H=CH/(MH)

    error=1
    NewDens=1.2

    while error>0.000001:
        GuessDens=NewDens
        Pu=(1000*NewDens-MHNO3*H)*mPu/(1000+MPUNO34*mPu)
        mH=mPu*H/Pu
        Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
        error=(abs(Dens1-GuessDens))/(Dens1)
        NewDens=Dens1
    if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
        print("ERROR: Data entered is out of bounds of equation range.")
        return 0
    else:
        FinalDens=NewDens

        return FinalDens

def Density4(Pu,H,T):
    """Pu and H are in molarity (mol solute/L solution) and T is in Celsius"""
```

```
    #Density Coefficients
    A=0.402234
    B=0.029992
    C=3.01282*(10**-5)
    D=-0.017735
    E=-4.7033*(10**-4)
    F=-0.026282
    G=-1.172*(10**-3)
    do=0.997

    #Molar Masses
    MPu239=239.0521636
    MH=1.0078
    MO=15.999
    MN=14.007
    MPUNO34=MPu239+(MN+3*MO)*4
    MHNO3=MH+(MN+3*MO)

    error=1
    NewDens=1.2

    while error>0.000001:
        GuessDens=NewDens
        mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
        mH=mPu*H/Pu
        Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
        error=((((Dens1-GuessDens))**2)**0.5)/(Dens1)
        NewDens=Dens1
    if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
        print("ERROR: Data entered is out of bounds of equation range.")
        return 0

    else:
        FinalDens=NewDens

        return FinalDens

def Density5(Pu,CH,T):
    """Pu is in molarity (mol solute/L solution), CH is in concentration (g solute/L solution), and T is in Celsius"""
    #Density Coefficients
    A=0.402234
    B=0.029992
    C=3.01282*(10**-5)
    D=-0.017735
    E=-4.7033*(10**-4)
    F=-0.026282
    G=-1.172*(10**-3)
    do=0.997

    #Molar Masses
    MPu239=239.0521636
    MH=1.0078
    MO=15.999
    MN=14.007
    MPUNO34=MPu239+(MN+3*MO)*4
    MHNO3=MH+(MN+3*MO)

    H=CH/(MH)

    error=1
    NewDens=1.2

    while error>0.000001:
        GuessDens=NewDens
        mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
        mH=mPu*H/Pu
        Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
        error=(abs(Dens1-GuessDens))/(Dens1)
        NewDens=Dens1
```

```
    if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
        print("ERROR: Data entered is out of bounds of equation range.")
        return 0
    else:
        FinalDens=NewDens

        return FinalDens

def Density6(Pu,mH,T):
    """Pu is in molarity (mol solute/L solution), mH is in molality (mol solute/kg solvent), and T is in Celsius"""
    #Density Coefficients
    A=0.402234
    B=0.029992
    C=3.01282*(10**-5)
    D=-0.017735
    E=-4.7033*(10**-4)
    F=-0.026282
    G=-1.172*(10**-3)
    do=0.997

    #Molar Masses
    MPu239=239.0521636
    MH=1.0078
    MO=15.999
    MN=14.007
    MPUNO34=MPu239+(MN+3*MO)*4
    MHNO3=MH+(MN+3*MO)

    error=1
    NewDens=1.2

    while error>0.000001:
        GuessDens=NewDens
        H=(1000*NewDens-MPUNO34*Pu)*mH/(1000+MHNO3*mH)
        mPu=Pu*mH/H
        Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
        error=(abs(Dens1-GuessDens))/(Dens1)
        NewDens=Dens1
    if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
        print("ERROR: Data entered is out of bounds of equation range.")
        return 0
    else:
        FinalDens=NewDens

        return FinalDens

def Density7(CPu,CH,T):
    """CPu and CH are in concentration (g solute/L solution) and T is in Celsius"""
    #Density Coefficients
    A=0.402234
    B=0.029992
    C=3.01282*(10**-5)
    D=-0.017735
    E=-4.7033*(10**-4)
    F=-0.026282
    G=-1.172*(10**-3)
    do=0.997

    #Molar Masses
    MPu239=239.0521636
    MH=1.0078
    MO=15.999
    MN=14.007
    MPUNO34=MPu239+(MN+3*MO)*4
    MHNO3=MH+(MN+3*MO)

    Pu=CPu/(MPu239)
    H=CH/(MH)
```

```
    error=1
    NewDens=1.2

    while error>0.000001:
        GuessDens=NewDens
        mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
        mH=mPu*H/Pu
        Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
        error=(abs(Dens1-GuessDens))/(Dens1)
        NewDens=Dens1
    if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
        print("ERROR: Data entered is out of bounds of equation range.")
        return 0
    else:
        FinalDens=NewDens

        return FinalDens

def Density8(CPu,H,T):
    """CPu is in concentration (g solute/L solution), H is in molarity (mol solute/L solution), and T is in Celsius"""
    #Density Coefficients
    A=0.402234
    B=0.029992
    C=3.01282*(10**-5)
    D=-0.017735
    E=-4.7033*(10**-4)
    F=-0.026282
    G=-1.172*(10**-3)
    do=0.997

    #Molar Masses
    MPu239=239.0521636
    MH=1.0078
    MO=15.999
    MN=14.007
    MPUNO34=MPu239+(MN+3*MO)*4
    MHNO3=MH+(MN+3*MO)

    Pu=CPu/(MPu239)

    error=1
    NewDens=1.2

    while error>0.000001:
        GuessDens=NewDens
        mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
        mH=mPu*H/Pu
        Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
        error=(abs(Dens1-GuessDens))/(Dens1)
        NewDens=Dens1
    if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
        print("ERROR: Data entered is out of bounds of equation range.")
        return 0
    else:
        FinalDens=NewDens

        return FinalDens

def Density9(CPu,mH,T):
    """CPu is in concentration (g solute/L solution), mH is in molality (mol solute/kg solvent), and T is in Celsius"""
    #Density Coefficients
    A=0.402234
    B=0.029992
    C=3.01282*(10**-5)
    D=-0.017735
    E=-4.7033*(10**-4)
    F=-0.026282
    G=-1.172*(10**-3)
    do=0.997
```

```
  #Molar Masses
  MPu239=239.0521636
  MH=1.0078
  MO=15.999
  MN=14.007
  MPUNO34=MPu239+(MN+3*MO)*4
  MHNO3=MH+(MN+3*MO)

  Pu=CPu/(MPu239)

  error=1
  NewDens=1.2

  while error>0.000001:
    GuessDens=NewDens
    H=(1000*NewDens-MPUNO34*Pu)*mH/(1000+MHNO3*mH)
    mPu=mH*Pu/H
    Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
    error=(abs(Dens1-GuessDens))/(Dens1)
    NewDens=Dens1
  if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
    print("ERROR: Data entered is out of bounds of equation range.")
    return 0
  else:
    FinalDens=NewDens

    return FinalDens

##########################################################################################################
##########################################################################################################
######################
"""SECTION 2"""
##########################################################################################################
##########################################################################################################
######################

def DensityEditor1(inputfile,newfile,mPu,mH,T):
  """Place @ where density value is desired in inputfile. Units are in molality (mol solute/kg solvent) and temperature in celsius.
NOTE: To edit file, the file must be in the same folder as the code and typed in the format of "inputfile.txt"."""
  if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
    print("ERROR: Data entered is out of bounds of equation range.")
    return 0
  else:


    #Density Coefficients
    A=0.402234
    B=0.029992
    C=3.01282*(10**-5)
    D=-0.017735
    E=-4.7033*(10**-4)
    F=-0.026282
    G=-1.172*(10**-3)
    do=0.997

    #Variables to help with file editing
    glue=' '



    Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
    Dens=str(-Dens1)

    with open(inputfile) as f:
      lines=f.readlines()
    L=len(lines)

    Rd=[0]*L
```

```
    for i in range(0,L):
        strng=lines[i]
        new=strng.split()
        Rd[i]=new
    Flines=lines
    MarkerI=0
    MarkerJ=0
    for i in range(0, L):

        if (Rd[i]==[]):
            continue
        else:
            Temp=Rd[i][0]
            if (Temp=='c'):
                continue
            else:
                Temp1=Rd[i]

        for j in range(0, len(Temp1)):
            if (Rd[i][j]=='@'):
                MarkerI=i
                MarkerJ=j

        if (MarkerI>0):
            Rd[MarkerI][MarkerJ]=Dens

            newl=glue.join(Rd[i])+'\n'
            Flines[MarkerI]=newl

            break
    with open(newfile,'w') as f:
        for i in range(0,L):
            strg=Flines[i]
            f.writelines(strg)

def DensityEditor2(inputfile,newfile,CPu,H,T):
    """ Place @ where density value is desired in the file. CPu is in concentration (g solute/L solution) and H is in molarity (mol solute/L solution). NOTE: To edit file, the file must be in the same folder as the code and typed in the format of "inputfile.txt"."""


    #Density Coefficients
    A=0.402234
    B=0.029992
    C=3.01282*(10**-5)
    D=-0.017735
    E=-4.7033*(10**-4)
    F=-0.026282
    G=-1.172*(10**-3)
    do=0.997

    #Molar Masses
    MPu239=239.0521636
    MH=1.0078
    MO=15.999
    MN=14.007
    MPUNO34=MPu239+(MN+3*MO)*4
    MHNO3=MH+(MN+3*MO)

    #Variables to help with file editing
    glue=' '


    Pu=CPu/(MPu239)


    error=1
    NewDens=1.2
```

```
  while error>0.000001:
    GuessDens=NewDens
    mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
    mH=mPu*H/Pu
    Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
    error=(abs(Dens1-GuessDens))/(Dens1)
    NewDens=Dens1
  Dens1=NewDens

  if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
    print("ERROR: Data entered is out of bounds of equation range.")
    return 0
  else:

    Dens=str(-Dens1)

    with open(inputfile) as f:
      lines=f.readlines()
    L=len(lines)

    Rd=[0]*L

    for i in range(0,L):
      strng=lines[i]
      new=strng.split()
      Rd[i]=new
    Flines=lines
    MarkerI=0
    MarkerJ=0
    for i in range(0, L):

      if (Rd[i]==[]):
        continue
      else:
        Temp=Rd[i][0]
        if (Temp=='c'):
          continue
        else:
          Temp1=Rd[i]

      for j in range(0, len(Temp1)):
        if (Rd[i][j]=='@'):
          MarkerI=i
          MarkerJ=j

      if (MarkerI>0):
        Rd[MarkerI][MarkerJ]=Dens
        newl=glue.join(Rd[i])+'\n'
        Flines[MarkerI]=newl

        break
    with open(newfile,'w') as f:
      for i in range(0,L):
        strg=Flines[i]
        f.writelines(strg)


########################################################################################################
########################################################################################################
######################
"""SECTION 3"""
########################################################################################################
########################################################################################################
######################
def Solution1(mPu,mH,wPu,Imp,T):
  """For wPu type a list of lists that contains the ZAID # of the isotope and the decimal weight percent for Pu-238,239,240,241,242
or Am-241 in the format of wPu=[[ZAID,wt.%],[ZAID, wt.%]]. Imp is also a list of lists containing the isotope ZAID, concentration
(g/l) in the solution, the molar mass, and how many bonds it forms with NO3- in the format Imp=[[ZAID, Concentration, Molar
Mass, Binding#],[etc]]. Both wPu and Imp may be empty and added as wPu=[] and/or Imp=[]. This is used for 100% Pu-239 and/or
```

```
no impurities. Units are in molality for mPu and mH (mole solute/kg solvent)."""
  if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
    print("ERROR: Data entered is out of bounds of equation range.")
    return 0
  else:

    #Density Coefficients
    A=0.402234
    B=0.029992
    C=3.01282*(10**-5)
    D=-0.017735
    E=-4.7033*(10**-4)
    F=-0.026282
    G=-1.172*(10**-3)
    do=0.997

    #Molar Masses
    MPu238=238.0495601
    MPu239=239.0521636
    MPu240=240.0538138
    MPu241=241.0568517
    MPu242=242.0587428
    MAm241=241.0568291
    MH=1.0078
    MO=15.999
    MN=14.007
    MPUNO34=MPu239+(MN+3*MO)*4
    MHNO3=MH+(MN+3*MO)
    MSolvent=MH+MH+MO

    #Conversions
    Na=6.022*(10**23)
    Barntocm2=10**24
    Cm3toL=10**3



    LwPu=len(wPu)

    w94238=0
    w94239=0
    w94240=0
    w94241=0
    w94242=0
    w95241=0

    ZAID94238=0
    ZAID94239=0
    ZAID94240=0
    ZAID94241=0
    ZAID94242=0
    ZAID95241=0

    if LwPu>0:
      for i in range(0,LwPu):
        if wPu[i][0]==94238:
          w94238=wPu[i][1]
          ZAID94238=94238
        if wPu[i][0]==94239:
          w94239=wPu[i][1]
          ZAID94239=94239
        if wPu[i][0]==94240:
          w94240=wPu[i][1]
          ZAID94240=94240
        if wPu[i][0]==94241:
          w94241=wPu[i][1]
          ZAID94241=94241
        if wPu[i][0]==94242:
          w94242=wPu[i][1]
```

```
        ZAID94242=94242
      if wPu[i][0]==95241:
        w95241=wPu[i][1]
        ZAID95241=95241
else:
  w94239=1
  ZAID94239=94239

numImp=len(Imp)
if numImp>0:
  ZAIDImp1=Imp[0][0]
  CImp1=Imp[0][1]
  MImp1=Imp[0][2]
  BImp1=Imp[0][3]
  Imp1=CImp1/MImp1

  if numImp>1:
    ZAIDImp2=Imp[1][0]
    CImp2=Imp[1][1]
    MImp2=Imp[1][2]
    BImp2=Imp[1][3]
    Imp2=CImp2/MImp2

    if numImp>2:
      ZAIDImp3=Imp[2][0]
      CImp3=Imp[2][1]
      MImp3=Imp[2][2]
      BImp3=Imp[2][3]
      Imp3=CImp3/MImp3

      if numImp>3:
        ZAIDImp4=Imp[3][0]
        CImp4=Imp[3][1]
        MImp4=Imp[3][2]
        BImp4=Imp[3][3]
        Imp4=CImp4/MImp4

        if numImp>4:
          ZAIDImp5=Imp[5][0]
          CImp5=Imp[4][1]
          MImp5=Imp[4][2]
          BImp5=Imp[4][3]
          Imp5=CImp5/MImp5

          if numImp>5:
            ZAIDImp6=Imp[5][0]
            CImp6=Imp[5][1]
            MImp6=Imp[5][2]
            BImp6=Imp[5][3]
            Imp6=CImp6/MImp6

            if numImp>6:
              ZAIDImp7=Imp[6][0]
              CImp7=Imp[6][1]
              MImp7=Imp[6][2]
              BImp7=Imp[6][3]
              Imp7=CImp7/MImp7

              if numImp>7:
                ZAIDImp8=Imp[7][0]
                CImp8=Imp[7][1]
                MImp8=Imp[7][2]
                BImp8=Imp[7][3]
                Imp8=CImp8/MImp8

                if numImp>8:
                  ZAIDImp9=Imp[8][0]
                  CImp9=Imp[8][1]
                  MImp9=Imp[8][2]
```

```
            BImp9=Imp[8][3]
            Imp9=CImp9/MImp9

            if numImp>9:
              ZAIDImp10=Imp[9][0]
              CImp10=Imp[9][1]
              MImp10=Imp[9][2]
              BImp10=Imp[9][3]
              Imp10=CImp10/MImp10

              if numImp>10:
                ZAIDImp11=Imp[10][0]
                CImp11=Imp[10][1]
                MImp11=Imp[10][2]
                BImp11=Imp[10][3]
                Imp11=CImp11/MImp11
              else:
                Imp11=0
                BImp11=0
                ZAIDImp11=0
            else:
              Imp10=0
              BImp10=0
              ZAIDImp10=0
              Imp11=0
              BImp11=0
              ZAIDImp11=0
          else:
            Imp9=0
            BImp9=0
            ZAIDImp9=0
            Imp10=0
            BImp10=0
            ZAIDImp10=0
            Imp11=0
            BImp11=0
            ZAIDImp11=0
        else:
          Imp8=0
          BImp8=0
          ZAIDImp8=0
          Imp9=0
          BImp9=0
          ZAIDImp9=0
          Imp10=0
          BImp10=0
          ZAIDImp10=0
          Imp11=0
          BImp11=0
          ZAIDImp11=0

      else:
        Imp7=0
        BImp7=0
        ZAIDImp7=0
        Imp8=0
        BImp8=0
        ZAIDImp8=0
        Imp9=0
        BImp9=0
        ZAIDImp9=0
        Imp10=0
        BImp10=0
        ZAIDImp10=0
        Imp11=0
        BImp11=0
        ZAIDImp11=0
    else:
      Imp6=0
```

```
            BImp6=0
            ZAIDImp6=0
            Imp7=0
            BImp7=0
            ZAIDImp7=0
            Imp8=0
            BImp8=0
            ZAIDImp8=0
            Imp9=0
            BImp9=0
            ZAIDImp9=0
            Imp10=0
            BImp10=0
            ZAIDImp10=0
            Imp11=0
            BImp11=0
            ZAIDImp11=0

        else:
          Imp5=0
          BImp5=0
          ZAIDImp5=0
          Imp6=0
          BImp6=0
          ZAIDImp6=0
          Imp7=0
          BImp7=0
          ZAIDImp7=0
          Imp8=0
          BImp8=0
          ZAIDImp8=0
          Imp9=0
          BImp9=0
          ZAIDImp9=0
          Imp10=0
          BImp10=0
          ZAIDImp10=0
          Imp11=0
          BImp11=0
          ZAIDImp11=0
      else:
        Imp4=0
        BImp4=0
        ZAIDImp4=0
        Imp5=0
        BImp5=0
        ZAIDImp5=0
        Imp6=0
        BImp6=0
        ZAIDImp6=0
        Imp7=0
        BImp7=0
        ZAIDImp7=0
        Imp8=0
        BImp8=0
        ZAIDImp8=0
        Imp9=0
        BImp9=0
        ZAIDImp9=0
        Imp10=0
        BImp10=0
        ZAIDImp10=0
        Imp11=0
        BImp11=0
        ZAIDImp11=0
    else:
      Imp3=0
      BImp3=0
      ZAIDImp3=0
```

```
        Imp4=0
        BImp4=0
        ZAIDImp4=0
        Imp5=0
        BImp5=0
        ZAIDImp5=0
        Imp6=0
        BImp6=0
        ZAIDImp6=0
        Imp7=0
        BImp7=0
        ZAIDImp7=0
        Imp8=0
        BImp8=0
        ZAIDImp8=0
        Imp9=0
        BImp9=0
        ZAIDImp9=0
        Imp10=0
        BImp10=0
        ZAIDImp10=0
        Imp11=0
        BImp11=0
        ZAIDImp11=0
    else:
      Imp2=0
      BImp2=0
      ZAIDImp2=0
      Imp3=0
      BImp3=0
      ZAIDImp3=0
      Imp4=0
      BImp4=0
      ZAIDImp4=0
      Imp5=0
      BImp5=0
      ZAIDImp5=0
      Imp6=0
      BImp6=0
      ZAIDImp6=0
      Imp7=0
      BImp7=0
      ZAIDImp7=0
      Imp8=0
      BImp8=0
      ZAIDImp8=0
      Imp9=0
      BImp9=0
      ZAIDImp9=0
      Imp10=0
      BImp10=0
      ZAIDImp10=0
      Imp11=0
      BImp11=0
      ZAIDImp11=0
else:
  Imp1=0
  BImp1=0
  ZAIDImp1=0
  Imp2=0
  BImp2=0
  ZAIDImp2=0
  Imp3=0
  BImp3=0
  ZAIDImp3=0
  Imp4=0
  BImp4=0
  ZAIDImp4=0
  Imp5=0
```

```
    BImp5=0
    ZAIDImp5=0
    Imp6=0
    BImp6=0
    ZAIDImp6=0
    Imp7=0
    BImp7=0
    ZAIDImp7=0
    Imp8=0
    BImp8=0
    ZAIDImp8=0
    Imp9=0
    BImp9=0
    ZAIDImp9=0
    Imp10=0
    BImp10=0
    ZAIDImp10=0
    Imp11=0
    BImp11=0
    ZAIDImp11=0

  Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
  PuNO34=Dens1*mPu/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)   #molPu/Lsolution
  HNO3=Dens1*mH/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)

  Pu=PuNO34*1
  CPu=Pu*MPu239
  H=HNO3*1

  Pu238=CPu*w94238/MPu238
  Pu239=CPu*w94239/MPu239
  Pu240=CPu*w94240/MPu240
  Pu241=CPu*w94241/MPu241
  Pu242=CPu*w94242/MPu242
  Am241=CPu*w95241/MAm241

  AmNO33=Am241
  Am=Am241


N=(4*PuNO34)+HNO3+(3*AmNO33)+(BImp1*Imp1)+(BImp2*Imp2)+(BImp3*Imp3)+(BImp4*Imp4)+(BImp5*Imp5)+(BImp
6*Imp6)+(BImp7*Imp7)+(BImp8*Imp8)+(BImp9*Imp9)+(BImp10*Imp10)+(BImp11*Imp11)

  rhoPUNO34=Pu*MPUNO34/1000
  rhoHNO3=H*MHNO3/1000
  rhoSolvent=Dens1-rhoPUNO34-rhoHNO3
  Solvent=rhoSolvent*1000/MSolvent

  Htotal=Solvent*2+HNO3*1
  Ototal=(Solvent*1)+N*3

  NPu238=0
  NPu239=0
  NPu240=0
  NPu241=0
  NPu242=0
  NAm241=0

  if (w94238>0):
    NPu238=((Pu238/Cm3toL)*Na/Barntocm2)
  if (w94239>0):
    NPu239=((Pu239/Cm3toL)*Na/Barntocm2)
  if (w94240>0):
    NPu240=((Pu240/Cm3toL)*Na/Barntocm2)
  if (w94241>0):
    NPu241=((Pu241/Cm3toL)*Na/Barntocm2)
  if (w94242>0):
    NPu242=((Pu242/Cm3toL)*Na/Barntocm2)
  if (w95241>0):
```

```
    NAm241=((Am241/Cm3toL)*Na/Barntocm2)

    NImp1=((Imp1/Cm3toL)*Na/Barntocm2)
    NImp2=((Imp2/Cm3toL)*Na/Barntocm2)
    NImp3=((Imp3/Cm3toL)*Na/Barntocm2)
    NImp4=((Imp4/Cm3toL)*Na/Barntocm2)
    NImp5=((Imp5/Cm3toL)*Na/Barntocm2)
    NImp6=((Imp6/Cm3toL)*Na/Barntocm2)
    NImp7=((Imp7/Cm3toL)*Na/Barntocm2)
    NImp8=((Imp8/Cm3toL)*Na/Barntocm2)
    NImp9=((Imp9/Cm3toL)*Na/Barntocm2)
    NImp10=((Imp10/Cm3toL)*Na/Barntocm2)
    NImp11=((Imp11/Cm3toL)*Na/Barntocm2)

    NAm241=((Am/Cm3toL)*Na/Barntocm2)


    NH=((Htotal/Cm3toL)*Na/Barntocm2)
    NN=((N/Cm3toL)*Na/Barntocm2)
    NO=((Ototal/Cm3toL)*Na/Barntocm2)


    MaterialCard={ZAID94238:NPu238,
            ZAID94239:NPu239,
            ZAID94240:NPu240,
            ZAID94241:NPu241,
            ZAID94242:NPu242,
            ZAID95241:NAm241,
            1001:NH,
            7014:NN,
            8016:NO,
            ZAIDImp1:NImp1,
            ZAIDImp2:NImp2,
            ZAIDImp3:NImp3,
            ZAIDImp4:NImp4,
            ZAIDImp5:NImp5,
            ZAIDImp6:NImp6,
            ZAIDImp7:NImp7,
            ZAIDImp8:NImp8,
            ZAIDImp9:NImp9,
            ZAIDImp10:NImp10,
            ZAIDImp11:NImp11}
    del MaterialCard[0]


  return Dens1,MaterialCard


def Solution2(CPu,H,wPu,Imp,T):
  """For wPu type a list of lists that contains the ZAID # of the isotope and the decimal weight percent for Pu-238,239,240,241,242
or Am-241 in the format of wPu=[[ZAID,wt.%],[ZAID, wt.%]]. Imp is also a list of lists containing the isotope ZAID, concentration
(g/l) in the solution, the molar mass, and how many bonds it forms with NO3- in the format Imp=[[ZAID, Concentration, Molar
Mass, Binding#],[etc]]. Both wPu and Imp may be empty and added as wPu=[] and/or Imp=[]. This is used for 100% Pu-239 and/or
no impurities. CPu is in concentration (g solute/L solution) and H is in molarity (mol solute/L solution)."""


  #Density Coefficients
  A=0.402234
  B=0.029992
  C=3.01282*(10**-5)
  D=-0.017735
  E=-4.7033*(10**-4)
  F=-0.026282
  G=-1.172*(10**-3)
  do=0.997

  #Molar Masses
  MPu238=238.0495601
  MPu239=239.0521636
```

```
MPu240=240.0538138
MPu241=241.0568517
MPu242=242.0587428
MAm241=241.0568291
MH=1.0078
MO=15.999
MN=14.007
MPUNO34=MPu239+(MN+3*MO)*4
MHNO3=MH+(MN+3*MO)
MSolvent=MH+MH+MO

#Conversions
Na=6.022*(10**23)
Barntocm2=10**24
Cm3toL=10**3



LwPu=len(wPu)

w94238=0
w94239=0
w94240=0
w94241=0
w94242=0
w95241=0

ZAID94238=0
ZAID94239=0
ZAID94240=0
ZAID94241=0
ZAID94242=0
ZAID95241=0

if LwPu>0:
   for i in range(0,LwPu):
      if wPu[i][0]==94238:
         w94238=wPu[i][1]
         ZAID94238=94238
      if wPu[i][0]==94239:
         w94239=wPu[i][1]
         ZAID94239=94239
      if wPu[i][0]==94240:
         w94240=wPu[i][1]
         ZAID94240=94240
      if wPu[i][0]==94241:
         w94241=wPu[i][1]
         ZAID94241=94241
      if wPu[i][0]==94242:
         w94242=wPu[i][1]
         ZAID94242=94242
      if wPu[i][0]==95241:
         w95241=wPu[i][1]
         ZAID95241=95241

numImp=len(Imp)
if numImp>0:
   ZAIDImp1=Imp[0][0]
   CImp1=Imp[0][1]
   MImp1=Imp[0][2]
   BImp1=Imp[0][3]
   Imp1=CImp1/MImp1

   if numImp>1:
      ZAIDImp2=Imp[1][0]
      CImp2=Imp[1][1]
      MImp2=Imp[1][2]
      BImp2=Imp[1][3]
      Imp2=CImp2/MImp2
```

```
if numImp>2:
   ZAIDImp3=Imp[2][0]
   CImp3=Imp[2][1]
   MImp3=Imp[2][2]
   BImp3=Imp[2][3]
   Imp3=CImp3/MImp3

   if numImp>3:
      ZAIDImp4=Imp[3][0]
      CImp4=Imp[3][1]
      MImp4=Imp[3][2]
      BImp4=Imp[3][3]
      Imp4=CImp4/MImp4

      if numImp>4:
         ZAIDImp5=Imp[5][0]
         CImp5=Imp[4][1]
         MImp5=Imp[4][2]
         BImp5=Imp[4][3]
         Imp5=CImp5/MImp5

         if numImp>5:
            ZAIDImp6=Imp[5][0]
            CImp6=Imp[5][1]
            MImp6=Imp[5][2]
            BImp6=Imp[5][3]
            Imp6=CImp6/MImp6

            if numImp>6:
               ZAIDImp7=Imp[6][0]
               CImp7=Imp[6][1]
               MImp7=Imp[6][2]
               BImp7=Imp[6][3]
               Imp7=CImp7/MImp7

               if numImp>7:
                  ZAIDImp8=Imp[7][0]
                  CImp8=Imp[7][1]
                  MImp8=Imp[7][2]
                  BImp8=Imp[7][3]
                  Imp8=CImp8/MImp8

                  if numImp>8:
                     ZAIDImp9=Imp[8][0]
                     CImp9=Imp[8][1]
                     MImp9=Imp[8][2]
                     BImp9=Imp[8][3]
                     Imp9=CImp9/MImp9

                     if numImp>9:
                        ZAIDImp10=Imp[9][0]
                        CImp10=Imp[9][1]
                        MImp10=Imp[9][2]
                        BImp10=Imp[9][3]
                        Imp10=CImp10/MImp10

                        if numImp>10:
                           ZAIDImp11=Imp[10][0]
                           CImp11=Imp[10][1]
                           MImp11=Imp[10][2]
                           BImp11=Imp[10][3]
                           Imp11=CImp11/MImp11
                        else:
                           Imp11=0
                           BImp11=0
                           ZAIDImp11=0
                     else:
                        Imp10=0
```

```
                BImp10=0
                ZAIDImp10=0
                Imp11=0
                BImp11=0
                ZAIDImp11=0
            else:
              Imp9=0
              BImp9=0
              ZAIDImp9=0
              Imp10=0
              BImp10=0
              ZAIDImp10=0
              Imp11=0
              BImp11=0
              ZAIDImp11=0
          else:
            Imp8=0
            BImp8=0
            ZAIDImp8=0
            Imp9=0
            BImp9=0
            ZAIDImp9=0
            Imp10=0
            BImp10=0
            ZAIDImp10=0
            Imp11=0
            BImp11=0
            ZAIDImp11=0

        else:
          Imp7=0
          BImp7=0
          ZAIDImp7=0
          Imp8=0
          BImp8=0
          ZAIDImp8=0
          Imp9=0
          BImp9=0
          ZAIDImp9=0
          Imp10=0
          BImp10=0
          ZAIDImp10=0
          Imp11=0
          BImp11=0
          ZAIDImp11=0
      else:
        Imp6=0
        BImp6=0
        ZAIDImp6=0
        Imp7=0
        BImp7=0
        ZAIDImp7=0
        Imp8=0
        BImp8=0
        ZAIDImp8=0
        Imp9=0
        BImp9=0
        ZAIDImp9=0
        Imp10=0
        BImp10=0
        ZAIDImp10=0
        Imp11=0
        BImp11=0
        ZAIDImp11=0

  else:
    Imp5=0
    BImp5=0
    ZAIDImp5=0
```

```
        Imp6=0
        BImp6=0
        ZAIDImp6=0
        Imp7=0
        BImp7=0
        ZAIDImp7=0
        Imp8=0
        BImp8=0
        ZAIDImp8=0
        Imp9=0
        BImp9=0
        ZAIDImp9=0
        Imp10=0
        BImp10=0
        ZAIDImp10=0
        Imp11=0
        BImp11=0
        ZAIDImp11=0
    else:
      Imp4=0
      BImp4=0
      ZAIDImp4=0
      Imp5=0
      BImp5=0
      ZAIDImp5=0
      Imp6=0
      BImp6=0
      ZAIDImp6=0
      Imp7=0
      BImp7=0
      ZAIDImp7=0
      Imp8=0
      BImp8=0
      ZAIDImp8=0
      Imp9=0
      BImp9=0
      ZAIDImp9=0
      Imp10=0
      BImp10=0
      ZAIDImp10=0
      Imp11=0
      BImp11=0
      ZAIDImp11=0
  else:
    Imp3=0
    BImp3=0
    ZAIDImp3=0
    Imp4=0
    BImp4=0
    ZAIDImp4=0
    Imp5=0
    BImp5=0
    ZAIDImp5=0
    Imp6=0
    BImp6=0
    ZAIDImp6=0
    Imp7=0
    BImp7=0
    ZAIDImp7=0
    Imp8=0
    BImp8=0
    ZAIDImp8=0
    Imp9=0
    BImp9=0
    ZAIDImp9=0
    Imp10=0
    BImp10=0
    ZAIDImp10=0
    Imp11=0
```

```
        BImp11=0
        ZAIDImp11=0
    else:
        Imp2=0
        BImp2=0
        ZAIDImp2=0
        Imp3=0
        BImp3=0
        ZAIDImp3=0
        Imp4=0
        BImp4=0
        ZAIDImp4=0
        Imp5=0
        BImp5=0
        ZAIDImp5=0
        Imp6=0
        BImp6=0
        ZAIDImp6=0
        Imp7=0
        BImp7=0
        ZAIDImp7=0
        Imp8=0
        BImp8=0
        ZAIDImp8=0
        Imp9=0
        BImp9=0
        ZAIDImp9=0
        Imp10=0
        BImp10=0
        ZAIDImp10=0
        Imp11=0
        BImp11=0
        ZAIDImp11=0
else:
    Imp1=0
    BImp1=0
    ZAIDImp1=0
    Imp2=0
    BImp2=0
    ZAIDImp2=0
    Imp3=0
    BImp3=0
    ZAIDImp3=0
    Imp4=0
    BImp4=0
    ZAIDImp4=0
    Imp5=0
    BImp5=0
    ZAIDImp5=0
    Imp6=0
    BImp6=0
    ZAIDImp6=0
    Imp7=0
    BImp7=0
    ZAIDImp7=0
    Imp8=0
    BImp8=0
    ZAIDImp8=0
    Imp9=0
    BImp9=0
    ZAIDImp9=0
    Imp10=0
    BImp10=0
    ZAIDImp10=0
    Imp11=0
    BImp11=0
    ZAIDImp11=0

MPu=MPu239
```

```
  Pu=CPu/(MPu)

  error=1
  NewDens=1.2

  while error>0.000001:
    GuessDens=NewDens
    mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
    mH=mPu*H/Pu
    Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
    error=(abs(Dens1-GuessDens))/(Dens1)
    NewDens=Dens1
  Dens1=NewDens
  if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
    print("ERROR: Data entered is out of bounds of equation range.")
    return 0
  else:
    PuNO34=Pu
    HNO3=H

    Pu238=CPu*w94238/MPu238
    Pu239=CPu*w94239/MPu239
    Pu240=CPu*w94240/MPu240
    Pu241=CPu*w94241/MPu241
    Pu242=CPu*w94242/MPu242
    Am241=CPu*w95241/MAm241

    AmNO33=Am241
    Am=Am241


N=(4*PuNO34)+HNO3+(3*AmNO33)+(BImp1*Imp1)+(BImp2*Imp2)+(BImp3*Imp3)+(BImp4*Imp4)+(BImp5*Imp5)+(BImp
6*Imp6)+(BImp7*Imp7)+(BImp8*Imp8)+(BImp9*Imp9)+(BImp10*Imp10)+(BImp11*Imp11)

    rhoPUNO34=Pu*MPUNO34/1000
    rhoHNO3=H*MHNO3/1000
    rhoSolvent=Dens1-rhoPUNO34-rhoHNO3
    Solvent=rhoSolvent*1000/MSolvent

    Htotal=Solvent*2+HNO3*1
    Ototal=(Solvent*1)+N*3

    NPu238=0
    NPu239=0
    NPu240=0
    NPu241=0
    NPu242=0
    NAm241=0

    if (w94238>0):
      NPu238=((Pu238/Cm3toL)*Na/Barntocm2)
    if (w94239>0):
      NPu239=((Pu239/Cm3toL)*Na/Barntocm2)
    if (w94240>0):
      NPu240=((Pu240/Cm3toL)*Na/Barntocm2)
    if (w94241>0):
      NPu241=((Pu241/Cm3toL)*Na/Barntocm2)
    if (w94242>0):
      NPu242=((Pu242/Cm3toL)*Na/Barntocm2)
    if (w95241>0):
      NAm241=((Am241/Cm3toL)*Na/Barntocm2)

    NImp1=((Imp1/Cm3toL)*Na/Barntocm2)
    NImp2=((Imp2/Cm3toL)*Na/Barntocm2)
    NImp3=((Imp3/Cm3toL)*Na/Barntocm2)
    NImp4=((Imp4/Cm3toL)*Na/Barntocm2)
    NImp5=((Imp5/Cm3toL)*Na/Barntocm2)
    NImp6=((Imp6/Cm3toL)*Na/Barntocm2)
    NImp7=((Imp7/Cm3toL)*Na/Barntocm2)
```

```
    NImp8=((Imp8/Cm3toL)*Na/Barntocm2)
    NImp9=((Imp9/Cm3toL)*Na/Barntocm2)
    NImp10=((Imp10/Cm3toL)*Na/Barntocm2)
    NImp11=((Imp11/Cm3toL)*Na/Barntocm2)

    NAm241=((Am/Cm3toL)*Na/Barntocm2)


    NH=((Htotal/Cm3toL)*Na/Barntocm2)
    NN=((N/Cm3toL)*Na/Barntocm2)
    NO=((Ototal/Cm3toL)*Na/Barntocm2)


    MaterialCard={ZAID94238:NPu238,
            ZAID94239:NPu239,
            ZAID94240:NPu240,
            ZAID94241:NPu241,
            ZAID94242:NPu242,
            ZAID95241:NAm241,
            1001:NH,
            7014:NN,
            8016:NO,
            ZAIDImp1:NImp1,
            ZAIDImp2:NImp2,
            ZAIDImp3:NImp3,
            ZAIDImp4:NImp4,
            ZAIDImp5:NImp5,
            ZAIDImp6:NImp6,
            ZAIDImp7:NImp7,
            ZAIDImp8:NImp8,
            ZAIDImp9:NImp9,
            ZAIDImp10:NImp10,
            ZAIDImp11:NImp11}
    del MaterialCard[0]

    return Dens1,MaterialCard

##########################################################################################################
##########################################################################################################
#####################
"""SECTION 4"""
##########################################################################################################
##########################################################################################################
#####################

def SolutionEditor1(inputfile,newfile,mPu,mH,wPu,Imp,T):
   """Place @ where density value is desired in the file and ensure there is a material number (ex. m100) in the data card that is
empty. For wPu type a list of lists that contains the ZAID # of the isotope and the decimal weight percent for Pu-
238,239,240,241,242 or Am-241 in the format of wPu=[[ZAID,wt.%],[ZAID, wt.%]]. Imp is also a list of lists containing the isotope
ZAID, concentration (g/l) in the solution, the molar mass, and how many bonds it forms with NO3- in the format Imp=[[ZAID,
Concentration, Molar Mass, Binding#],[etc]]. Both wPu and Imp may be empty and added as wPu=[] and/or Imp=[]. This is used for
100% Pu-239 and/or no impurities. Units are in molality for mPu and mH (mole solute/kg solvent). NOTE: To edit file, the file must
be in the same folder as the code and typed in the format of "inputfile.txt". The editor will automatically find the assigned material
number prior to density location and look for the m# in the data card to place the material card. Ensure numbers match and are in
the correct locations for a MCNP input file."""
   if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
      print("ERROR: Data entered is out of bounds of equation range.")
      return 0
   else:

      #Density Coefficients
      A=0.402234
      B=0.029992
      C=3.01282*(10**-5)
      D=-0.017735
      E=-4.7033*(10**-4)
      F=-0.026282
      G=-1.172*(10**-3)
      do=0.997
```

```
#Molar Masses
MPu238=238.0495601
MPu239=239.0521636
MPu240=240.0538138
MPu241=241.0568517
MPu242=242.0587428
MAm241=241.0568291
MH=1.0078
MO=15.999
MN=14.007
MPUNO34=MPu239+(MN+3*MO)*4
MHNO3=MH+(MN+3*MO)
MSolvent=MH+MH+MO

#Conversions
Na=6.022*(10**23)
Barntocm2=10**24
Cm3toL=10**3

#Variables to help with file editing
glue=' '

LwPu=len(wPu)

w94238=0
w94239=0
w94240=0
w94241=0
w94242=0
w95241=0

if LwPu>0:
   for i in range(0,LwPu):
      if wPu[i][0]==94238:
         w94238=wPu[i][1]
      if wPu[i][0]==94239:
         w94239=wPu[i][1]
      if wPu[i][0]==94240:
         w94240=wPu[i][1]
      if wPu[i][0]==94241:
         w94241=wPu[i][1]
      if wPu[i][0]==94242:
         w94242=wPu[i][1]
      if wPu[i][0]==95241:
         w95241=wPu[i][1]

numImp=len(Imp)
if numImp>0:
   ZAIDImp1=Imp[0][0]
   CImp1=Imp[0][1]
   MImp1=Imp[0][2]
   BImp1=Imp[0][3]
   Imp1=CImp1/MImp1

   if numImp>1:
      ZAIDImp2=Imp[1][0]
      CImp2=Imp[1][1]
      MImp2=Imp[1][2]
      BImp2=Imp[1][3]
      Imp2=CImp2/MImp2

      if numImp>2:
         ZAIDImp3=Imp[2][0]
         CImp3=Imp[2][1]
         MImp3=Imp[2][2]
         BImp3=Imp[2][3]
         Imp3=CImp3/MImp3
```

```
if numImp>3:
   ZAIDImp4=Imp[3][0]
   CImp4=Imp[3][1]
   MImp4=Imp[3][2]
   BImp4=Imp[3][3]
   Imp4=CImp4/MImp4

   if numImp>4:
      ZAIDImp5=Imp[5][0]
      CImp5=Imp[4][1]
      MImp5=Imp[4][2]
      BImp5=Imp[4][3]
      Imp5=CImp5/MImp5

      if numImp>5:
         ZAIDImp6=Imp[5][0]
         CImp6=Imp[5][1]
         MImp6=Imp[5][2]
         BImp6=Imp[5][3]
         Imp6=CImp6/MImp6

         if numImp>6:
            ZAIDImp7=Imp[6][0]
            CImp7=Imp[6][1]
            MImp7=Imp[6][2]
            BImp7=Imp[6][3]
            Imp7=CImp7/MImp7

            if numImp>7:
               ZAIDImp8=Imp[7][0]
               CImp8=Imp[7][1]
               MImp8=Imp[7][2]
               BImp8=Imp[7][3]
               Imp8=CImp8/MImp8

               if numImp>8:
                  ZAIDImp9=Imp[8][0]
                  CImp9=Imp[8][1]
                  MImp9=Imp[8][2]
                  BImp9=Imp[8][3]
                  Imp9=CImp9/MImp9

                  if numImp>9:
                     ZAIDImp10=Imp[9][0]
                     CImp10=Imp[9][1]
                     MImp10=Imp[9][2]
                     BImp10=Imp[9][3]
                     Imp10=CImp10/MImp10

                     if numImp>10:
                        ZAIDImp11=Imp[10][0]
                        CImp11=Imp[10][1]
                        MImp11=Imp[10][2]
                        BImp11=Imp[10][3]
                        Imp11=CImp11/MImp11
                     else:
                        Imp11=0
                        BImp11=0
                        ZAIDImp11=0
                  else:
                     Imp10=0
                     BImp10=0
                     ZAIDImp10=0
                     Imp11=0
                     BImp11=0
                     ZAIDImp11=0
               else:
                  Imp9=0
                  BImp9=0
```

```
            ZAIDImp9=0
            Imp10=0
            BImp10=0
            ZAIDImp10=0
            Imp11=0
            BImp11=0
            ZAIDImp11=0
        else:
          Imp8=0
          BImp8=0
          ZAIDImp8=0
          Imp9=0
          BImp9=0
          ZAIDImp9=0
          Imp10=0
          BImp10=0
          ZAIDImp10=0
          Imp11=0
          BImp11=0
          ZAIDImp11=0

      else:
        Imp7=0
        BImp7=0
        ZAIDImp7=0
        Imp8=0
        BImp8=0
        ZAIDImp8=0
        Imp9=0
        BImp9=0
        ZAIDImp9=0
        Imp10=0
        BImp10=0
        ZAIDImp10=0
        Imp11=0
        BImp11=0
        ZAIDImp11=0
    else:
      Imp6=0
      BImp6=0
      ZAIDImp6=0
      Imp7=0
      BImp7=0
      ZAIDImp7=0
      Imp8=0
      BImp8=0
      ZAIDImp8=0
      Imp9=0
      BImp9=0
      ZAIDImp9=0
      Imp10=0
      BImp10=0
      ZAIDImp10=0
      Imp11=0
      BImp11=0
      ZAIDImp11=0

  else:
    Imp5=0
    BImp5=0
    ZAIDImp5=0
    Imp6=0
    BImp6=0
    ZAIDImp6=0
    Imp7=0
    BImp7=0
    ZAIDImp7=0
    Imp8=0
    BImp8=0
```

```
            ZAIDImp8=0
            Imp9=0
            BImp9=0
            ZAIDImp9=0
            Imp10=0
            BImp10=0
            ZAIDImp10=0
            Imp11=0
            BImp11=0
            ZAIDImp11=0
        else:
          Imp4=0
          BImp4=0
          ZAIDImp4=0
          Imp5=0
          BImp5=0
          ZAIDImp5=0
          Imp6=0
          BImp6=0
          ZAIDImp6=0
          Imp7=0
          BImp7=0
          ZAIDImp7=0
          Imp8=0
          BImp8=0
          ZAIDImp8=0
          Imp9=0
          BImp9=0
          ZAIDImp9=0
          Imp10=0
          BImp10=0
          ZAIDImp10=0
          Imp11=0
          BImp11=0
          ZAIDImp11=0
      else:
        Imp3=0
        BImp3=0
        ZAIDImp3=0
        Imp4=0
        BImp4=0
        ZAIDImp4=0
        Imp5=0
        BImp5=0
        ZAIDImp5=0
        Imp6=0
        BImp6=0
        ZAIDImp6=0
        Imp7=0
        BImp7=0
        ZAIDImp7=0
        Imp8=0
        BImp8=0
        ZAIDImp8=0
        Imp9=0
        BImp9=0
        ZAIDImp9=0
        Imp10=0
        BImp10=0
        ZAIDImp10=0
        Imp11=0
        BImp11=0
        ZAIDImp11=0
    else:
      Imp2=0
      BImp2=0
      ZAIDImp2=0
      Imp3=0
      BImp3=0
```

```
        ZAIDImp3=0
        Imp4=0
        BImp4=0
        ZAIDImp4=0
        Imp5=0
        BImp5=0
        ZAIDImp5=0
        Imp6=0
        BImp6=0
        ZAIDImp6=0
        Imp7=0
        BImp7=0
        ZAIDImp7=0
        Imp8=0
        BImp8=0
        ZAIDImp8=0
        Imp9=0
        BImp9=0
        ZAIDImp9=0
        Imp10=0
        BImp10=0
        ZAIDImp10=0
        Imp11=0
        BImp11=0
        ZAIDImp11=0
else:
    Imp1=0
    BImp1=0
    ZAIDImp1=0
    Imp2=0
    BImp2=0
    ZAIDImp2=0
    Imp3=0
    BImp3=0
    ZAIDImp3=0
    Imp4=0
    BImp4=0
    ZAIDImp4=0
    Imp5=0
    BImp5=0
    ZAIDImp5=0
    Imp6=0
    BImp6=0
    ZAIDImp6=0
    Imp7=0
    BImp7=0
    ZAIDImp7=0
    Imp8=0
    BImp8=0
    ZAIDImp8=0
    Imp9=0
    BImp9=0
    ZAIDImp9=0
    Imp10=0
    BImp10=0
    ZAIDImp10=0
    Imp11=0
    BImp11=0
    ZAIDImp11=0

MPu238=238.0495601
MPu239=239.0521636
MPu240=240.0538138
MPu241=241.0568517
MPu242=242.0587428
MAm241=241.0568291
MPu=MPu239
MH=1.0078
MO=15.999
```

```
    MN=14.007   #g/mol
    MPUNO34=MPu+(MN+3*MO)*4
    MHNO3=MH+(MN+3*MO)
    MSolvent=MH+MH+MO

    Na=6.022*(10**23)
    Barntocm2=10**24
    Cm3toL=10**3

    Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
    Dens=str(-Dens1)

    PuNO34=Dens1*mPu/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)   #molPu/Lsolution
    HNO3=Dens1*mH/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)

    Pu=PuNO34*1
    H=HNO3*1
    CPu=Pu*MPu

    Pu238=CPu*w94238/MPu238
    Pu239=CPu*w94239/MPu239
    Pu240=CPu*w94240/MPu240
    Pu241=CPu*w94241/MPu241
    Pu242=CPu*w94242/MPu242
    Am241=CPu*w95241/MAm241

    AmNO33=Am241
    Am=Am241


N=(4*PuNO34)+HNO3+(3*AmNO33)+(BImp1*Imp1)+(BImp2*Imp2)+(BImp3*Imp3)+(BImp4*Imp4)+(BImp5*Imp5)+(BImp
6*Imp6)+(BImp7*Imp7)+(BImp8*Imp8)+(BImp9*Imp9)+(BImp10*Imp10)+(BImp11*Imp11)

    rhoPUNO34=Pu*MPUNO34/1000
    rhoHNO3=H*MHNO3/1000

    rhoSolvent=Dens1-rhoPUNO34-rhoHNO3
    Solvent=rhoSolvent*1000/MSolvent

    Htotal=Solvent*2+HNO3*1
    Ototal=(Solvent*1)+N*3

    Flines=[]
    MaterialCard=[]
    FMatCard=[0]
    Md=[0]

    if (w94238>0):
       NPu238=str((Pu238/Cm3toL)*Na/Barntocm2)
       MaterialCard.append(['94238',NPu238])
    if (w94239>0):
       NPu239=str((Pu239/Cm3toL)*Na/Barntocm2)
       MaterialCard.append(['94239',NPu239])
    if (w94240>0):
       NPu240=str((Pu240/Cm3toL)*Na/Barntocm2)
       MaterialCard.append(['94240',NPu240])
    if (w94241>0):
       NPu241=str((Pu241/Cm3toL)*Na/Barntocm2)
       MaterialCard.append(['94241',NPu241])
    if (w94242>0):
       NPu242=str((Pu242/Cm3toL)*Na/Barntocm2)
       MaterialCard.append(['94242',NPu242])
    if Am>0:
       NAm241=str((Am/Cm3toL)*Na/Barntocm2)
       MaterialCard.append(['95241',NAm241])

    NH=str((Htotal/Cm3toL)*Na/Barntocm2)
    MaterialCard.append(['1001',NH])
    NN=str((N/Cm3toL)*Na/Barntocm2)
```

```
MaterialCard.append(['7014',NN])
NO=str((Ototal/Cm3toL)*Na/Barntocm2)
MaterialCard.append(['8016',NO])
if Imp1>0:
   ZAID1=str(ZAIDImp1)
   NImp1=str((Imp1/Cm3toL)*Na/Barntocm2)
   MaterialCard.append([ZAID1,NImp1])
   if Imp2>0:
      ZAID2=str(ZAIDImp2)
      NImp2=str((Imp2/Cm3toL)*Na/Barntocm2)
      MaterialCard.append([ZAID2,NImp2])
      if Imp3>0:
         ZAID3=str(ZAIDImp3)
         NImp3=str((Imp3/Cm3toL)*Na/Barntocm2)
         MaterialCard.append([ZAID3,NImp3])
         if Imp4>0:
            ZAID4=str(ZAIDImp4)
            NImp4=str((Imp4/Cm3toL)*Na/Barntocm2)
            MaterialCard.append([ZAID4,NImp4])
            if Imp5>0:
               ZAID5=str(ZAIDImp5)
               NImp5=str((Imp5/Cm3toL)*Na/Barntocm2)
               MaterialCard.append([ZAID5,NImp5])
               if Imp6>0:
                  ZAID6=str(ZAIDImp6)
                  NImp6=str((Imp6/Cm3toL)*Na/Barntocm2)
                  MaterialCard.append([ZAID6,NImp6])
                  if Imp7>0:
                     ZAID7=str(ZAIDImp7)
                     NImp7=str((Imp7/Cm3toL)*Na/Barntocm2)
                     MaterialCard.append([ZAID7,NImp7])
                     if Imp8>0:
                        ZAID8=str(ZAIDImp8)
                        NImp8=str((Imp8/Cm3toL)*Na/Barntocm2)
                        MaterialCard.append([ZAID8,NImp8])
                        if Imp9>0:
                           ZAID9=str(ZAIDImp9)
                           NImp9=str((Imp9/Cm3toL)*Na/Barntocm2)
                           MaterialCard.append([ZAID9,NImp9])
                           if Imp10>0:
                              ZAID10=str(ZAIDImp10)
                              NImp10=str((Imp10/Cm3toL)*Na/Barntocm2)
                              MaterialCard.append([ZAID10,NImp10])
                              if Imp11>0:
                                 ZAID11=str(ZAIDImp11)
                                 NImp11=str((Imp11/Cm3toL)*Na/Barntocm2)
                                 MaterialCard.append([ZAID11,NImp11])


with open(inputfile) as f:
   lines=f.readlines()
L=len(lines)
Lm=len(MaterialCard)
Rd=[0]*L

for i in range(0,L):
   strng=lines[i]
   new=strng.split()
   Rd[i]=new


for i in range(0, L):

   if (Rd[i]==[]):
      continue
   else:
      Temp=Rd[i][0]
      if (Temp=='c'):
         continue
```

```
            else:
                Temp1=Rd[i]

                for j in range(0, len(Temp1)):
                    if (Rd[i][j]=='@'):
                        DensI=i
                        DensJ=j
                        break

        Rd[DensI][DensJ]=Dens
        Matnum=str(Rd[DensI][DensJ-1])
        MatKey='m'+Matnum
        mtnm=float(Rd[DensI][DensJ-1])
        Space='     '
        if mtnm<10:
            Space1="   "
        elif mtnm<100:
            Space1="  "
        elif mtnm<1000:
            Space1=' '

        for i in range(0,Lm):
            if i==0:
                Md[i]=[MatKey,MaterialCard[0][0],MaterialCard[0][1]]
                new1=Md[i][0]+Space1+Md[i][1]+glue+Md[i][2]+'\n'
                FMatCard[i]=new1
            else:
                new1=Space+glue.join(MaterialCard[i])+'\n'
                FMatCard.append(new1)

        for i in range(DensI, L):

            if (Rd[i]==[]):
                continue
            else:
                Temp=Rd[i][0]
                if (Temp=='c'):
                    continue
                else:
                    Temp1=Rd[i]

                    for j in range(0, len(Temp1)):
                        if (Rd[i][j]==MatKey):
                            MatI=i

        for i in range(0,L):
            if i==DensI:
                newdens=glue.join(Rd[i])+'\n'
                Flines.append(newdens)
            if i==MatI:
                for k in range(0,Lm):
                    Flines.append(FMatCard[k])

            else:
                Flines.append(lines[i])

        with open(newfile,'w') as f:
            for i in range(0,L+Lm-1):
                strg=Flines[i]
                f.writelines(strg)

def SolutionEditor2(inputfile,newfile,CPu,H,wPu,Imp,T):
    """Place @ where density value is desired in the file and ensure there is a material number (ex. m100) in the data card that is
empty. For wPu type a list of lists that contains the ZAID # of the isotope and the decimal weight percent for Pu-
238,239,240,241,242 or Am-241 in the format of wPu=[[ZAID,wt.%],[ZAID, wt.%]]. Imp is also a list of lists containing the isotope
ZAID, concentration (g/l) in the solution, the molar mass, and how many bonds it forms with NO3- in the format Imp=[[ZAID,
Concentration, Molar Mass, Binding#],[etc]]. Both wPu and Imp may be empty and added as wPu=[] and/or Imp=[]. This is used for
100% Pu-239 and/or no impurities. CPu is in concentration (g solute/L solution) and H is in molarity (mol solute/L solution).
NOTE: To edit file, the file must be in the same folder as the code and typed in the format of "inputfile.txt". The editor will
```

```
automatically find the assigned material number prior to density location and look for the m# in the data card to place the material
card. Ensure numbers match and are in the correct locations for a MCNP input file."""
   #Density Coefficients

   A=0.402234
   B=0.029992
   C=3.01282*(10**-5)
   D=-0.017735
   E=-4.7033*(10**-4)
   F=-0.026282
   G=-1.172*(10**-3)
   do=0.997

   #Molar Masses
   MPu238=238.0495601
   MPu239=239.0521636
   MPu240=240.0538138
   MPu241=241.0568517
   MPu242=242.0587428
   MAm241=241.0568291
   MH=1.0078
   MO=15.999
   MN=14.007
   MPUNO34=MPu239+(MN+3*MO)*4
   MHNO3=MH+(MN+3*MO)
   MSolvent=MH+MH+MO

   #Conversions
   Na=6.022*(10**23)
   Barntocm2=10**24
   Cm3toL=10**3

   #Variables to help with file editing
   glue=' '


   LwPu=len(wPu)

   w94238=0
   w94239=0
   w94240=0
   w94241=0
   w94242=0
   w95241=0

   if LwPu>0:
      for i in range(0,LwPu):
         if wPu[i][0]==94238:
            w94238=wPu[i][1]
         if wPu[i][0]==94239:
            w94239=wPu[i][1]
         if wPu[i][0]==94240:
            w94240=wPu[i][1]
         if wPu[i][0]==94241:
            w94241=wPu[i][1]
         if wPu[i][0]==94242:
            w94242=wPu[i][1]
         if wPu[i][0]==95241:
            w95241=wPu[i][1]

   numImp=len(Imp)
   if numImp>0:
      ZAIDImp1=Imp[0][0]
      CImp1=Imp[0][1]
      MImp1=Imp[0][2]
      BImp1=Imp[0][3]
      Imp1=CImp1/MImp1

      if numImp>1:
```

```
ZAIDImp2=Imp[1][0]
CImp2=Imp[1][1]
MImp2=Imp[1][2]
BImp2=Imp[1][3]
Imp2=CImp2/MImp2

if numImp>2:
  ZAIDImp3=Imp[2][0]
  CImp3=Imp[2][1]
  MImp3=Imp[2][2]
  BImp3=Imp[2][3]
  Imp3=CImp3/MImp3

  if numImp>3:
    ZAIDImp4=Imp[3][0]
    CImp4=Imp[3][1]
    MImp4=Imp[3][2]
    BImp4=Imp[3][3]
    Imp4=CImp4/MImp4

    if numImp>4:
      ZAIDImp5=Imp[5][0]
      CImp5=Imp[4][1]
      MImp5=Imp[4][2]
      BImp5=Imp[4][3]
      Imp5=CImp5/MImp5

      if numImp>5:
        ZAIDImp6=Imp[5][0]
        CImp6=Imp[5][1]
        MImp6=Imp[5][2]
        BImp6=Imp[5][3]
        Imp6=CImp6/MImp6

        if numImp>6:
          ZAIDImp7=Imp[6][0]
          CImp7=Imp[6][1]
          MImp7=Imp[6][2]
          BImp7=Imp[6][3]
          Imp7=CImp7/MImp7

          if numImp>7:
            ZAIDImp8=Imp[7][0]
            CImp8=Imp[7][1]
            MImp8=Imp[7][2]
            BImp8=Imp[7][3]
            Imp8=CImp8/MImp8

            if numImp>8:
              ZAIDImp9=Imp[8][0]
              CImp9=Imp[8][1]
              MImp9=Imp[8][2]
              BImp9=Imp[8][3]
              Imp9=CImp9/MImp9

              if numImp>9:
                ZAIDImp10=Imp[9][0]
                CImp10=Imp[9][1]
                MImp10=Imp[9][2]
                BImp10=Imp[9][3]
                Imp10=CImp10/MImp10

                if numImp>10:
                  ZAIDImp11=Imp[10][0]
                  CImp11=Imp[10][1]
                  MImp11=Imp[10][2]
                  BImp11=Imp[10][3]
                  Imp11=CImp11/MImp11
                else:
```

```
                Imp11=0
                BImp11=0
                ZAIDImp11=0
            else:
              Imp10=0
              BImp10=0
              ZAIDImp10=0
              Imp11=0
              BImp11=0
              ZAIDImp11=0
          else:
            Imp9=0
            BImp9=0
            ZAIDImp9=0
            Imp10=0
            BImp10=0
            ZAIDImp10=0
            Imp11=0
            BImp11=0
            ZAIDImp11=0
        else:
          Imp8=0
          BImp8=0
          ZAIDImp8=0
          Imp9=0
          BImp9=0
          ZAIDImp9=0
          Imp10=0
          BImp10=0
          ZAIDImp10=0
          Imp11=0
          BImp11=0
          ZAIDImp11=0

      else:
        Imp7=0
        BImp7=0
        ZAIDImp7=0
        Imp8=0
        BImp8=0
        ZAIDImp8=0
        Imp9=0
        BImp9=0
        ZAIDImp9=0
        Imp10=0
        BImp10=0
        ZAIDImp10=0
        Imp11=0
        BImp11=0
        ZAIDImp11=0
    else:
      Imp6=0
      BImp6=0
      ZAIDImp6=0
      Imp7=0
      BImp7=0
      ZAIDImp7=0
      Imp8=0
      BImp8=0
      ZAIDImp8=0
      Imp9=0
      BImp9=0
      ZAIDImp9=0
      Imp10=0
      BImp10=0
      ZAIDImp10=0
      Imp11=0
      BImp11=0
      ZAIDImp11=0
```

```
      else:
        Imp5=0
        BImp5=0
        ZAIDImp5=0
        Imp6=0
        BImp6=0
        ZAIDImp6=0
        Imp7=0
        BImp7=0
        ZAIDImp7=0
        Imp8=0
        BImp8=0
        ZAIDImp8=0
        Imp9=0
        BImp9=0
        ZAIDImp9=0
        Imp10=0
        BImp10=0
        ZAIDImp10=0
        Imp11=0
        BImp11=0
        ZAIDImp11=0
    else:
      Imp4=0
      BImp4=0
      ZAIDImp4=0
      Imp5=0
      BImp5=0
      ZAIDImp5=0
      Imp6=0
      BImp6=0
      ZAIDImp6=0
      Imp7=0
      BImp7=0
      ZAIDImp7=0
      Imp8=0
      BImp8=0
      ZAIDImp8=0
      Imp9=0
      BImp9=0
      ZAIDImp9=0
      Imp10=0
      BImp10=0
      ZAIDImp10=0
      Imp11=0
      BImp11=0
      ZAIDImp11=0
  else:
    Imp3=0
    BImp3=0
    ZAIDImp3=0
    Imp4=0
    BImp4=0
    ZAIDImp4=0
    Imp5=0
    BImp5=0
    ZAIDImp5=0
    Imp6=0
    BImp6=0
    ZAIDImp6=0
    Imp7=0
    BImp7=0
    ZAIDImp7=0
    Imp8=0
    BImp8=0
    ZAIDImp8=0
    Imp9=0
    BImp9=0
```

```
            ZAIDImp9=0
            Imp10=0
            BImp10=0
            ZAIDImp10=0
            Imp11=0
            BImp11=0
            ZAIDImp11=0
        else:
          Imp2=0
          BImp2=0
          ZAIDImp2=0
          Imp3=0
          BImp3=0
          ZAIDImp3=0
          Imp4=0
          BImp4=0
          ZAIDImp4=0
          Imp5=0
          BImp5=0
          ZAIDImp5=0
          Imp6=0
          BImp6=0
          ZAIDImp6=0
          Imp7=0
          BImp7=0
          ZAIDImp7=0
          Imp8=0
          BImp8=0
          ZAIDImp8=0
          Imp9=0
          BImp9=0
          ZAIDImp9=0
          Imp10=0
          BImp10=0
          ZAIDImp10=0
          Imp11=0
          BImp11=0
          ZAIDImp11=0
    else:
      Imp1=0
      BImp1=0
      ZAIDImp1=0
      Imp2=0
      BImp2=0
      ZAIDImp2=0
      Imp3=0
      BImp3=0
      ZAIDImp3=0
      Imp4=0
      BImp4=0
      ZAIDImp4=0
      Imp5=0
      BImp5=0
      ZAIDImp5=0
      Imp6=0
      BImp6=0
      ZAIDImp6=0
      Imp7=0
      BImp7=0
      ZAIDImp7=0
      Imp8=0
      BImp8=0
      ZAIDImp8=0
      Imp9=0
      BImp9=0
      ZAIDImp9=0
      Imp10=0
      BImp10=0
      ZAIDImp10=0
```

```
    Imp11=0
    BImp11=0
    ZAIDImp11=0


  MPu238=238.0495601
  MPu239=239.0521636
  MPu240=240.0538138
  MPu241=241.0568517
  MPu242=242.0587428
  MAm241=241.0568291
  MPu=MPu239
  #(w94238*MPu238)+(w94239*MPu239)+(w94240*MPu240)+(w94241*MPu241)+(w94242*MPu242)
  MH=1.0078
  MO=15.999
  MN=14.007   #g/mol
  MPUNO34=MPu+(MN+3*MO)*4
  MHNO3=MH+(MN+3*MO)
  MSolvent=MH+MH+MO

  Pu=CPu/(MPu)

  Na=6.022*(10**23)
  Barntocm2=10**24
  Cm3toL=10**3

  error=1
  NewDens=1.2

  while error>0.000001:
    GuessDens=NewDens
    mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
    mH=mPu*H/Pu
    Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
    error=(abs(Dens1-GuessDens))/(Dens1)
    NewDens=Dens1
  Dens1=NewDens
  Dens=str(-Dens1)
  if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
    print("ERROR: Data entered is out of bounds of equation range.")
    return 0
  else:

    PuNO34=Pu
    HNO3=H

    Pu238=CPu*w94238/MPu238
    Pu239=CPu*w94239/MPu239
    Pu240=CPu*w94240/MPu240
    Pu241=CPu*w94241/MPu241
    Pu242=CPu*w94242/MPu242
    Am241=CPu*w95241/MAm241

    AmNO33=Am241
    Am=Am241


N=(4*PuNO34)+HNO3+(3*AmNO33)+(BImp1*Imp1)+(BImp2*Imp2)+(BImp3*Imp3)+(BImp4*Imp4)+(BImp5*Imp5)+(BImp
6*Imp6)+(BImp7*Imp7)+(BImp8*Imp8)+(BImp9*Imp9)+(BImp10*Imp10)+(BImp11*Imp11)

    rhoPUNO34=Pu*MPUNO34/1000
    rhoHNO3=H*MHNO3/1000

    rhoSolvent=Dens1-rhoPUNO34-rhoHNO3
    Solvent=rhoSolvent*1000/MSolvent

    Htotal=Solvent*2+HNO3*1
    Ototal=(Solvent*1)+N*3
```

```
Flines=[]
MaterialCard=[]
FMatCard=[0]
Md=[0]

if (w94238>0):
   NPu238=str((Pu238/Cm3toL)*Na/Barntocm2)
   MaterialCard.append(['94238',NPu238])
if (w94239>0):
   NPu239=str((Pu239/Cm3toL)*Na/Barntocm2)
   MaterialCard.append(['94239',NPu239])
if (w94240>0):
   NPu240=str((Pu240/Cm3toL)*Na/Barntocm2)
   MaterialCard.append(['94240',NPu240])
if (w94241>0):
   NPu241=str((Pu241/Cm3toL)*Na/Barntocm2)
   MaterialCard.append(['94241',NPu241])
if (w94242>0):
   NPu242=str((Pu242/Cm3toL)*Na/Barntocm2)
   MaterialCard.append(['94242',NPu242])
if Am>0:
   NAm241=str((Am/Cm3toL)*Na/Barntocm2)
   MaterialCard.append(['95241',NAm241])

NH=str((Htotal/Cm3toL)*Na/Barntocm2)
MaterialCard.append(['1001',NH])
NN=str((N/Cm3toL)*Na/Barntocm2)
MaterialCard.append(['7014',NN])
NO=str((Ototal/Cm3toL)*Na/Barntocm2)
MaterialCard.append(['8016',NO])
if Imp1>0:
   ZAID1=str(ZAIDImp1)
   NImp1=str((Imp1/Cm3toL)*Na/Barntocm2)
   MaterialCard.append([ZAID1,NImp1])
   if Imp2>0:
      ZAID2=str(ZAIDImp2)
      NImp2=str((Imp2/Cm3toL)*Na/Barntocm2)
      MaterialCard.append([ZAID2,NImp2])
      if Imp3>0:
         ZAID3=str(ZAIDImp3)
         NImp3=str((Imp3/Cm3toL)*Na/Barntocm2)
         MaterialCard.append([ZAID3,NImp3])
         if Imp4>0:
            ZAID4=str(ZAIDImp4)
            NImp4=str((Imp4/Cm3toL)*Na/Barntocm2)
            MaterialCard.append([ZAID4,NImp4])
            if Imp5>0:
               ZAID5=str(ZAIDImp5)
               NImp5=str((Imp5/Cm3toL)*Na/Barntocm2)
               MaterialCard.append([ZAID5,NImp5])
               if Imp6>0:
                  ZAID6=str(ZAIDImp6)
                  NImp6=str((Imp6/Cm3toL)*Na/Barntocm2)
                  MaterialCard.append([ZAID6,NImp6])
                  if Imp7>0:
                     ZAID7=str(ZAIDImp7)
                     NImp7=str((Imp7/Cm3toL)*Na/Barntocm2)
                     MaterialCard.append([ZAID7,NImp7])
                     if Imp8>0:
                        ZAID8=str(ZAIDImp8)
                        NImp8=str((Imp8/Cm3toL)*Na/Barntocm2)
                        MaterialCard.append([ZAID8,NImp8])
                        if Imp9>0:
                           ZAID9=str(ZAIDImp9)
                           NImp9=str((Imp9/Cm3toL)*Na/Barntocm2)
                           MaterialCard.append([ZAID9,NImp9])
                           if Imp10>0:
                              ZAID10=str(ZAIDImp10)
                              NImp10=str((Imp10/Cm3toL)*Na/Barntocm2)
```

```
                    MaterialCard.append([ZAID10,NImp10])
                    if Imp11>0:
                       ZAID11=str(ZAIDImp11)
                       NImp11=str((Imp11/Cm3toL)*Na/Barntocm2)
                       MaterialCard.append([ZAID11,NImp11])


with open(inputfile) as f:
   lines=f.readlines()
L=len(lines)
Lm=len(MaterialCard)
Rd=[0]*L

for i in range(0,L):
   strng=lines[i]
   new=strng.split()
   Rd[i]=new


for i in range(0, L):

   if (Rd[i]==[]):
      continue
   else:
      Temp=Rd[i][0]
      if (Temp=='c'):
         continue
      else:
         Temp1=Rd[i]

         for j in range(0, len(Temp1)):
            if (Rd[i][j]=='@'):
               DensI=i
               DensJ=j
               break

Rd[DensI][DensJ]=Dens
Matnum=str(Rd[DensI][DensJ-1])
MatKey='m'+Matnum
mtnm=float(Rd[DensI][DensJ-1])
Space='    '
if mtnm<10:
   Space1="   "
elif mtnm<100:
   Space1="  "
elif mtnm<1000:
   Space1=' '

for i in range(0,Lm):
   if i==0:
      Md[i]=[MatKey,MaterialCard[0][0],MaterialCard[0][1]]
      new1=Md[i][0]+Space1+Md[i][1]+glue+Md[i][2]+'\n'
      FMatCard[i]=new1
   else:
      new1=Space+glue.join(MaterialCard[i])+'\n'
      FMatCard.append(new1)

for i in range(DensI, L):

   if (Rd[i]==[]):
      continue
   else:
      Temp=Rd[i][0]
      if (Temp=='c'):
         continue
      else:
         Temp1=Rd[i]

         for j in range(0, len(Temp1)):
```

```
            if (Rd[i][j]==MatKey):
               MatI=i

for i in range(0,L):
   if i==DensI:
      newdens=glue.join(Rd[i])+'\n'
      Flines.append(newdens)
   elif i==MatI:
      for k in range(0,Lm):
         Flines.append(FMatCard[k])
   else:
      Flines.append(lines[i])

with open(newfile,'w') as f:
   for i in range(0,L+Lm-1):
      strg=Flines[i]
      f.writelines(strg)
```

## Appendix B

```
# -*- coding: utf-8 -*-
"""
PuNSDensitySolver
Updated 03/24/2023
@author: Tara
"""
######################### EDIT GUIDE ##############################################
"""
mX is molality (mol X/kg solvent)
MX is Molar Mass (g X/mol X)
X  is molarity (mol X/L solution)
CX is concentration (g X/L solution)
wX is weight percent in percentage form Ex. wX=90 for 90% weight percent
NX is (g X/ g solution) the solution overall weight percent that will be
                entered into the MCNP material card
A,B,C,D,E,F,G are coefficients from Weber Paper


Density is always in kg/l
T is in Celsius


Files should be formatted: 'newfile1.txt'

"""
######################### EDIT GUIDE ##############################################



##############################
""" SECTION 1: Constants"""
##############################

#Density Coefficients
A=0.402234
B=0.029992
C=3.01282*(10**-5)
D=-0.017735
E=-4.7033*(10**-4)
F=-0.026282
G=-1.172*(10**-3)
do=0.997

#Molar Masses
MPu238=238.0495601
MPu239=239.0521636
MPu240=240.0538138
MPu241=241.0568517
MPu242=242.0587428
MAm241=241.0568291
MH=1.0078
MO=15.999
MN=14.007
MPu=MPu239
MPUNO34=MPu+(MN+3*MO)*4
MHNO3=MH+(MN+3*MO)
MSolvent=MH+MH+MO

#Conversions
Na=6.022*(10**23)
Barntocm2=10**24
Cm3toL=10**3

#Variables to help with file editing
glue=' '
error=1
```

```
NewDens=1.2
w94238=0
w94239=0
w94240=0
w94241=0
w94242=0
w95241=0
Implist=[]
NImpCont=[]
MaterialCard=[]
Flines=[]
MaterialCard=[]
FMatCard=[0]
Md=[0]
wPuMC=[]
implist2=[]

################################
"""SECTION 2: Initial Inputs"""
################################


###Units###
PuUnitVariable=input("For Pu: Enter 1 to input molality (mol Pu/kg solvent), enter 2 to input Concentration (g Pu/L solution), or
enter 3 to input molarity (mol Pu/L Solution): ")

HUnitVariable=input("For H: Enter 1 to input molality (mol H/kg solvent), enter 2 to input Concentration (g H/L solution), or enter
3 to input molarity (mol H/L Solution): ")

if (PuUnitVariable=="1" or PuUnitVariable=="2" or PuUnitVariable=="3") and (HUnitVariable=="1" or HUnitVariable=="2" or
HUnitVariable=="3"):

  ###Temperature###
  TemperatureUnit=input("Enter Temperature of Solution in Celsius: ")
  T=float(TemperatureUnit)
  ######################################################################
  """SECTION 3: Calculate Density and Pu&H values"""
  ######################################################################

  ### Molality Molality Input###
  if PuUnitVariable=="1":
    if HUnitVariable=="1":
      molalPu=input("Enter Plutonium Molality (mol Pu/kg solvent): ")
      mPu=float(molalPu)
      molalH=input("Enter HNO3 Molality (mol HNO3/kg solvent): ")
      mH=float(molalH)

      Dens=do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH
      Pu=Dens*mPu/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)   #molPu/Lsolution
      H=Dens*mH/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)

    ### Molality Concentration Input###
    if HUnitVariable=="2":
      molalPu=input("Enter Plutonium Molality (mol Pu/kg solvent): ")
      mPu=float(molalPu)
      ConcentrationH=input("Enter H concentration (g H/L solution): ")
      CH=float(ConcentrationH)
      H=CH/(MH)

      while error>0.000001:
        GuessDens=NewDens
        Pu=(1000*NewDens-MHNO3*H)*mPu/(1000+MPUNO34*mPu)
        mH=mPu*H/Pu
        Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
        error=(abs(Dens1-GuessDens))/(Dens1)
        NewDens=Dens1
      Dens=NewDens
      Pu=Dens*mPu/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)   #molPu/Lsolution
```

```
    ### Molality Molarity Input###
    if HUnitVariable=="3":
        molalPu=input("Enter Plutonium Molality (mol Pu/kg solvent): ")
        mPu=float(molalPu)

        MolarH=input("Enter HNO3 Molarity (mol HNO3/L solution): ")
        H=float(MolarH)

        while error>0.000001:
            GuessDens=NewDens
            Pu=(1000*NewDens-MHNO3*H)*mPu/(1000+MPUNO34*mPu)
            mH=mPu*H/Pu
            Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
            error=(abs(Dens1-GuessDens))/(Dens1)
            NewDens=Dens1
        Dens=NewDens
        Pu=Dens*mPu/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)   #molPu/Lsolution

### Molarity Molarity Input###
if PuUnitVariable=="3":
    if HUnitVariable=="3":
        MolarPu=input("Enter Plutonium Molarity (mol Pu/L solution): ")
        Pu=float(MolarPu)

        MolarH=input("Enter HNO3 Molarity (mol HNO3/L solution): ")
        H=float(MolarH)

        while error>0.000001:
            GuessDens=NewDens
            mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
            mH=mPu*H/Pu
            Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
            error=((((Dens1-GuessDens))**2)**0.5)/(Dens1)
            NewDens=Dens1

        Dens=NewDens

    ### Molarity Concentration Input###
    if HUnitVariable=="2":
        MolarPu=input("Enter Plutonium Molarity (mol Pu/L solution): ")
        Pu=float(MolarPu)

        ConcentrationH=input("Enter H concentration (g H/L solution): ")
        CH=float(ConcentrationH)
        H=CH/(MH)

        while error>0.000001:
            GuessDens=NewDens
            mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
            mH=mPu*H/Pu
            Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
            error=(abs(Dens1-GuessDens))/(Dens1)
            NewDens=Dens1
        Dens=NewDens

    ### Molarity Molality Input###
    if HUnitVariable=="1":
        MolarPu=input("Enter Plutonium Molarity (mol Pu/L solution): ")
        Pu=float(MolarPu)

        molalH=input("Enter HNO3 Molality (mol HNO3/kg solvent): ")
        mH=float(molalH)

        while error>0.000001:
            GuessDens=NewDens
            H=(1000*NewDens-MPUNO34*Pu)*mH/(1000+MHNO3*mH)
            mPu=Pu*mH/H
            Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
            error=(abs(Dens1-GuessDens))/(Dens1)
```

```
        NewDens=Dens1
      Dens=NewDens
      H=Dens*mH/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)

### Concentration Concentration Input###
if PuUnitVariable=="2":
   if HUnitVariable=="2":
      ConcentrationPu=input("Enter Plutonium concentration (g Pu/L solution): ")
      CPu=float(ConcentrationPu)
      Pu=CPu/(MPu)

      ConcentrationH=input("Enter H concentration (g H/L solution): ")
      CH=float(ConcentrationH)
      H=CH/(MH)

      while error>0.000001:
         GuessDens=NewDens
         mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
         mH=mPu*H/Pu
         Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
         error=(abs(Dens1-GuessDens))/(Dens1)
         NewDens=Dens1

      Dens=NewDens

   ### Concentration Molarity Input###
   if HUnitVariable=="3":
      ConcentrationPu=input("Enter Plutonium concentration (g Pu/L solution): ")
      CPu=float(ConcentrationPu)
      Pu=CPu/(MPu)

      MolarH=input("Enter HNO3 Molarity (mol HNO3/L solution): ")
      H=float(MolarH)

      while error>0.000001:
         GuessDens=NewDens
         mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
         mH=mPu*H/Pu
         Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
         error=(abs(Dens1-GuessDens))/(Dens1)
         NewDens=Dens1
      Dens=NewDens

      ### Concentration Molality Input###
   if HUnitVariable=="1":
      ConcentrationPu=input("Enter Plutonium concentration (g Pu/L solution): ")
      CPu=float(ConcentrationPu)
      Pu=CPu/(MPu)

      molalH=input("Enter HNO3 Molality (mol HNO3/kg solvent): ")
      mH=float(molalH)

      while error>0.000001:
         GuessDens=NewDens
         H=(1000*NewDens-MPUNO34*Pu)*mH/(1000+MHNO3*mH)
         mPu=Pu*mH/H
         Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
         error=(abs(Dens1-GuessDens))/(Dens1)
         NewDens=Dens1
      Dens=NewDens
      H=Dens*mH/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)
if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
   print("ERROR: Data entered is out of bounds of equation range.")
else:


   Question=input("Would you like to create a MCNP Material Card? (Y/N): ")
   ##################################
   """SECTION 4: Material Card"""
```

```
    ##################################
    if Question=="N" or Question=="n":
      for i in range(0,3):
        print("")
      print("Your solution density is: ", Dens, " kg/L")
      for i in range(0,3):
        print("")
    elif Question=="Y" or Question=="y":


      ########################################################
      """SECTION 4.1: Find Plutonium weight percentages"""
      ########################################################

      print('')
      print('')
      w95241=0
      Purity=input('Enter Desired weight percent of Pu-239 (0-100): ')
      w94239=float(Purity)/100
      NPu239=(((MPu*Pu/MPu239)*w94239/Cm3toL)*Na/Barntocm2)
      wPuMC.append([94239,NPu239])

      if w94239>1:
        print("ERROR: Must enter a weight percent from 0-100")
      elif w94239<1:
        Purity2=input('Enter Desired weight percent of Pu-240 (0-100): ')
        w94240=float(Purity2)/100
        wPuTotal=w94240+w94239
        NPu240=(((MPu*Pu/MPu240)*w94240/Cm3toL)*Na/Barntocm2)
        wPuMC.append([94240,NPu240])

        if wPuTotal>1:
          print("ERROR: Weight percents add up to greater than 100")
        elif wPuTotal<1:
          Purity3=input('Enter Desired weight percent of Pu-241 (0-100): ')
          w94241=float(Purity3)/100
          wPuTotal=w94241+w94239+w94240
          NPu241=(((MPu*Pu/MPu241)*w94241/Cm3toL)*Na/Barntocm2)
          wPuMC.append([94241,NPu241])
          if wPuTotal>1:
            print("ERROR: Weight percents add up to greater than 100")
          elif wPuTotal<1:
            Purity4=input('Enter Desired weight percent of Pu-242 (0-100): ')
            w94242=float(Purity4)/100
            wPuTotal=w94242+w94239+w94240+w94241
            NPu242=(((MPu*Pu/MPu242)*w94242/Cm3toL)*Na/Barntocm2)
            wPuMC.append([94242,NPu242])
            if wPuTotal>1:
              print("ERROR: Weight percents add up to greater than 100")
            elif wPuTotal<1:
              Purity5=input('Enter Desired weight percent of Pu-238 (0-100): ')
              w94238=float(Purity5)/100
              wPuTotal=w94238+w94239+w94240+w94241+w94242
              NPu238=(((MPu*Pu/MPu238)*w94238/Cm3toL)*Na/Barntocm2)
              wPuMC.append([94238,NPu238])
              if wPuTotal==1:
                Purity7=input("Would you like to add Am-241? (Y/N):")
                if Purity7=="Y" or Purity7=="y":
                  AmCheck=input("Would you like to enter Am-241 content as a wt.% of Pu now or as an impurity concentration
later?(%/I): ")
                  if AmCheck=="%":
                    AmCont=input("Enter desired weight percent Am-241 (0-100):")
                    w95241=float(AmCont)/100
                    Am241=(MPu*Pu/MAm241)*w95241
                    NAm241=((Am241/Cm3toL)*Na/Barntocm2)
                    wPuMC.append([95241,NAm241])
              elif wPuTotal>1:
                print("ERROR: Weight percents add up to greater than 100")
              elif wPuTotal<1:
```

```
                Purity6=input("Would you like to add Am-241? (Y/N):")
                if Purity6=="Y" or Purity6=="y":
                  AmCheck2=input("Would you like to enter Am-241 content as a wt.% of Pu now or as an impurity concentration
later?(%/I): ")
                  if AmCheck2=="%":
                    AmCont2=input("Enter desired weight percent Am-241 (0-100):")
                    w95241=float(AmCont2)/100
                    Am241=(MPu*Pu/MAm241)*w95241
                    NAm241=((Am241/Cm3toL)*Na/Barntocm2)
                    wPuMC.append([95241,NAm241])


    ###################################
    """SECTION 4.2: Find Impurities"""
    ###################################

    ImpQ=input("Enter number of impurities desired:")
    NumImp=int(ImpQ)
    if NumImp>0:
      for i in range(0,NumImp):
        ImpurityN=i+1
        print(" ")
        print(' ')
        print("For impurity #",ImpurityN)
        ZAIDQ=input("Enter impurity ZAID:")
        ZAID=float(ZAIDQ)
        CQ=input("Enter impurity concentration (g/L):")
        CImp=float(CQ)
        MQ=input("Enter impurity Molar Mass (g/mol):")
        MImp=float(MQ)
        BQ=input("Enter the number of bonds the impurity has with NO3- (ex. Fe has 3):")
        BImp=float(BQ)
        Imp=CImp/MImp
        NImp=((Imp/Cm3toL)*Na/Barntocm2)
        NContribution=BImp*Imp
        NImpCont.append(NContribution)
        Implist.append([ZAID,NImp])
        implist2.append([ZAID,CImp,MImp,BImp])

    NImpContribution=sum(NImpCont)

    ###################################
    """SECTION 4.3: Form Material Card"""
    ###################################

    PuNO34=Pu
    HNO3=H
    if w95241>0:
      N=(4*PuNO34)+(HNO3)+NImpContribution+(3*Am241)
    else:
      N=(4*PuNO34)+(HNO3)+NImpContribution
    rhoPUNO34=Pu*MPUNO34/1000
    rhoHNO3=H*MHNO3/1000
    rhoSolvent=Dens-rhoPUNO34-rhoHNO3
    Solvent=rhoSolvent*1000/MSolvent

    Htotal=Solvent*2+HNO3*1
    Ototal=(Solvent*1)+N*3
    NH=((Htotal/Cm3toL)*Na/Barntocm2)
    NN=((N/Cm3toL)*Na/Barntocm2)
    NO=((Ototal/Cm3toL)*Na/Barntocm2)

    for i in range(0,len(wPuMC)):
      if (wPuMC[i][1])>0:
        MaterialCard.append([str(wPuMC[i][0]),str(wPuMC[i][1])])

    MaterialCard.append(['1001',str(NH)])
    MaterialCard.append(['7014',str(NN)])
    MaterialCard.append(['8016',str(NO)])
```

```
for i in range(0,len(Implist)):
  if (Implist[i][1])>0:
    MaterialCard.append([str(Implist[i][0]),str(Implist[i][1])])
Lm=len(MaterialCard)
for i in range(0,3):
  print("")
print("Your solution density is: ", Dens, " kg/L")


print("")

print("###  Material Card  ###")
for i in range(0,Lm):
  print(MaterialCard[i][0],MaterialCard[i][1])

for i in range(0,5):
  print("")

##################################
"""SECTION 5: Create MCNP INPUT CARD"""
##################################
Q=input("Would you like to add the solution density and material card to a MCNP input file? (Y/N): ")
if Q=="Y" or Q=="y":
  print("")
  print("Reminder: In input file add @ symbol where solution density is required.")
  print("")
  file2=input("Enter Entire MCNP File Name (file.txt): ")

  FQ=input("If you would like to specify new file name enter name now (Please include .txt). If not type 1.:")
  if FQ=="1":
    newfile="newfile.txt"
  else:
    newfile=FQ

  with open(file2) as f:
    lines=f.readlines()
  L=len(lines)

  Rd=[0]*L

  for i in range(0,L):
    strng=lines[i]
    new=strng.split()
    Rd[i]=new


  for i in range(1, L):

    if (Rd[i]==[]):
      continue
    else:
      Temp=Rd[i][0]
      if (Temp=='c'):
        continue
      else:
        Temp1=Rd[i]

        for j in range(0, len(Temp1)):
          if (Rd[i][j]=='@'):
            DensI=i
            DensJ=j
            break

  Rd[DensI][DensJ]=str(-Dens)
  Matnum=str(Rd[DensI][DensJ-1])
  MatKey='m'+Matnum
  mtnm=float(Rd[DensI][DensJ-1])
  Space='    '
  if mtnm<10:
```

```
            Space1="   "
        elif mtnm<100:
            Space1="  "
        elif mtnm<1000:
            Space1=' '

        for i in range(0,Lm):
            if i==0:
                Md[i]=[MatKey,MaterialCard[0][0],MaterialCard[0][1]]
                new1=Md[i][0]+Space1+Md[i][1]+glue+Md[i][2]+'\n'
                FMatCard[i]=new1
            else:
                new1=Space+glue.join(MaterialCard[i])+'\n'
                FMatCard.append(new1)

        for i in range(DensI, L):

            if (Rd[i]==[]):
                continue
            else:
                Temp=Rd[i][0]
                if (Temp=='c'):
                    continue
                else:
                    Temp1=Rd[i]

                    for j in range(0, len(Temp1)):
                        if (Rd[i][j]==MatKey):
                            MatI=i

        for i in range(0,L):
            if i==DensI:
                newdens=glue.join(Rd[i])+'\n'
                Flines.append(newdens)
            elif i==MatI:
                for k in range(0,Lm):
                    Flines.append(FMatCard[k])

            else:
                Flines.append(lines[i])

        with open(newfile,'w') as f:
            for i in range(0,L+Lm-1):
                strg=Flines[i]
                f.writelines(strg)
        print("")
        print("File Complete!")
        ##################################
        """SECTION 6: Repeat Loop"""
        ##################################
        print("")
    indicator=1
    while indicator==1:
        Qmulti=input("Would you like to run another case? (Y/N):")

        if Qmulti=='y' or Qmulti=="Y":
            Qinput=input("would you like to change the input units or temperature? (Y/N):")
            error=1
            NewDens=1.2
            MaterialCard=[]
            if Qinput=="y" or Qinput=="Y":
                ###Units###
                PuUnitVariable=input("For Pu: Enter 1 to input molality (mol Pu/kg solvent), enter 2 to input Concentration (g Pu/L
solution), or enter 3 to input molarity (mol Pu/L Solution): ")
                HUnitVariable=input("For H: Enter 1 to input molality (mol H/kg solvent), enter 2 to input Concentration (g H/L
solution), or enter 3 to input molarity (mol H/L Solution): ")
                    ###Temperature###
                TemperatureUnit=input("Enter Temperature of Solution in Celsius: ")
                T=float(TemperatureUnit)
```

```
if PuUnitVariable=="1":
   if HUnitVariable=="1":
      molalPu=input("Enter Plutonium Molality (mol Pu/kg solvent): ")
      mPu=float(molalPu)
      molalH=input("Enter HNO3 Molality (mol HNO3/kg solvent): ")
      mH=float(molalH)

      Dens=do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH
      Pu=Dens*mPu/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)   #molPu/Lsolution
      H=Dens*mH/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)

   ### Molality Concentration Input###
   if HUnitVariable=="2":
      molalPu=input("Enter Plutonium Molality (mol Pu/kg solvent): ")
      mPu=float(molalPu)
      ConcentrationH=input("Enter H concentration (g H/L solution): ")
      CH=float(ConcentrationH)
      H=CH/(MH)

      while error>0.000001:
         GuessDens=NewDens
         Pu=(1000*NewDens-MHNO3*H)*mPu/(1000+MPUNO34*mPu)
         mH=mPu*H/Pu
         Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
         error=(abs(Dens1-GuessDens))/(Dens1)
         NewDens=Dens1
      Dens=NewDens
      Pu=Dens*mPu/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)   #molPu/Lsolution

   ### Molality Molarity Input###
   if HUnitVariable=="3":
      molalPu=input("Enter Plutonium Molality (mol Pu/kg solvent): ")
      mPu=float(molalPu)

      MolarH=input("Enter HNO3 Molarity (mol HNO3/L solution): ")
      H=float(MolarH)

      while error>0.000001:
         GuessDens=NewDens
         Pu=(1000*NewDens-MHNO3*H)*mPu/(1000+MPUNO34*mPu)
         mH=mPu*H/Pu
         Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
         error=(abs(Dens1-GuessDens))/(Dens1)
         NewDens=Dens1
      Dens=NewDens
      Pu=Dens*mPu/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)   #molPu/Lsolution

### Molarity Molarity Input###
if PuUnitVariable=="3":
   if HUnitVariable=="3":
      MolarPu=input("Enter Plutonium Molarity (mol Pu/L solution): ")
      Pu=float(MolarPu)

      MolarH=input("Enter HNO3 Molarity (mol HNO3/L solution): ")
      H=float(MolarH)

      while error>0.000001:
         GuessDens=NewDens
         mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
         mH=mPu*H/Pu
         Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
         error=((((Dens1-GuessDens))**2)**0.5)/(Dens1)
         NewDens=Dens1

      Dens=NewDens

   ### Molarity Concentration Input###
   if HUnitVariable=="2":
```

```
      MolarPu=input("Enter Plutonium Molarity (mol Pu/L solution): ")
      Pu=float(MolarPu)

      ConcentrationH=input("Enter H concentration (g H/L solution): ")
      CH=float(ConcentrationH)
      H=CH/(MH)

      while error>0.000001:
        GuessDens=NewDens
        mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
        mH=mPu*H/Pu
        Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
        error=(abs(Dens1-GuessDens))/(Dens1)
        NewDens=Dens1
      Dens=NewDens

    ### Molarity Molality Input###
    if HUnitVariable=="1":
      MolarPu=input("Enter Plutonium Molarity (mol Pu/L solution): ")
      Pu=float(MolarPu)

      molalH=input("Enter HNO3 Molality (mol HNO3/kg solvent): ")
      mH=float(molalH)

      while error>0.000001:
        GuessDens=NewDens
        H=(1000*NewDens-MPUNO34*Pu)*mH/(1000+MHNO3*mH)
        mPu=Pu*mH/H
        Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
        error=(abs(Dens1-GuessDens))/(Dens1)
        NewDens=Dens1
      Dens=NewDens
      H=Dens*mH/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)

### Concentration Concentration Input###
if PuUnitVariable=="2":
  if HUnitVariable=="2":
      ConcentrationPu=input("Enter Plutonium concentration (g Pu/L solution): ")
      CPu=float(ConcentrationPu)
      Pu=CPu/(MPu)

      ConcentrationH=input("Enter H concentration (g H/L solution): ")
      CH=float(ConcentrationH)
      H=CH/(MH)

      while error>0.000001:
        GuessDens=NewDens
        mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
        mH=mPu*H/Pu
        Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
        error=(abs(Dens1-GuessDens))/(Dens1)
        NewDens=Dens1

      Dens=NewDens

    ### Concentration Molarity Input###
    if HUnitVariable=="3":
      ConcentrationPu=input("Enter Plutonium concentration (g Pu/L solution): ")
      CPu=float(ConcentrationPu)
      Pu=CPu/(MPu)

      MolarH=input("Enter HNO3 Molarity (mol HNO3/L solution): ")
      H=float(MolarH)

      while error>0.000001:
        GuessDens=NewDens
        mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
        mH=mPu*H/Pu
        Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
```

```
            error=(abs(Dens1-GuessDens))/(Dens1)
            NewDens=Dens1
          Dens=NewDens

          ### Concentration Molality Input###
          if HUnitVariable=="1":
            ConcentrationPu=input("Enter Plutonium concentration (g Pu/L solution): ")
            CPu=float(ConcentrationPu)
            Pu=CPu/(MPu)

            molalH=input("Enter HNO3 Molality (mol HNO3/kg solvent): ")
            mH=float(molalH)

            while error>0.000001:
              GuessDens=NewDens
              H=(1000*NewDens-MPUNO34*Pu)*mH/(1000+MHNO3*mH)
              mPu=Pu*mH/mPu
              Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
              error=(abs(Dens1-GuessDens))/(Dens1)
              NewDens=Dens1
            Dens=NewDens
            H=Dens*mH/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)
elif Qinput=="n" or Qinput=="N":
  if PuUnitVariable=="1":
    if HUnitVariable=="1":
      molalPu=input("Enter Plutonium Molality (mol Pu/kg solvent): ")
      mPu=float(molalPu)
      molalH=input("Enter HNO3 Molality (mol HNO3/kg solvent): ")
      mH=float(molalH)

      Dens=do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH
      Pu=Dens*mPu/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)   #molPu/Lsolution
      H=Dens*mH/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)

    ### Molality Concentration Input###
    if HUnitVariable=="2":
      molalPu=input("Enter Plutonium Molality (mol Pu/kg solvent): ")
      mPu=float(molalPu)
      ConcentrationH=input("Enter H concentration (g H/L solution): ")
      CH=float(ConcentrationH)
      H=CH/(MH)

      while error>0.000001:
        GuessDens=NewDens
        Pu=(1000*NewDens-MHNO3*H)*mPu/(1000+MPUNO34*mPu)
        mH=mPu*H/Pu
        Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
        error=(abs(Dens1-GuessDens))/(Dens1)
        NewDens=Dens1
      Dens=NewDens
      Pu=Dens*mPu/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)   #molPu/Lsolution

    ### Molality Molarity Input###
    if HUnitVariable=="3":
      molalPu=input("Enter Plutonium Molality (mol Pu/kg solvent): ")
      mPu=float(molalPu)

      MolarH=input("Enter HNO3 Molarity (mol HNO3/L solution): ")
      H=float(MolarH)

      while error>0.000001:
        GuessDens=NewDens
        Pu=(1000*NewDens-MHNO3*H)*mPu/(1000+MPUNO34*mPu)
        mH=mPu*H/Pu
        Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
        error=(abs(Dens1-GuessDens))/(Dens1)
        NewDens=Dens1
      Dens=NewDens
      Pu=Dens*mPu/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)   #molPu/Lsolution
```

```
### Molarity Molarity Input###
if PuUnitVariable=="3":
  if HUnitVariable=="3":
    MolarPu=input("Enter Plutonium Molarity (mol Pu/L solution): ")
    Pu=float(MolarPu)

    MolarH=input("Enter HNO3 Molarity (mol HNO3/L solution): ")
    H=float(MolarH)

    while error>0.000001:
      GuessDens=NewDens
      mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
      mH=mPu*H/Pu
      Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
      error=((((Dens1-GuessDens))**2)**0.5)/(Dens1)
      NewDens=Dens1

    Dens=NewDens

  ### Molarity Concentration Input###
  if HUnitVariable=="2":
    MolarPu=input("Enter Plutonium Molarity (mol Pu/L solution): ")
    Pu=float(MolarPu)

    ConcentrationH=input("Enter H concentration (g H/L solution): ")
    CH=float(ConcentrationH)
    H=CH/(MH)

    while error>0.000001:
      GuessDens=NewDens
      mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
      mH=mPu*H/Pu
      Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
      error=(abs(Dens1-GuessDens))/(Dens1)
      NewDens=Dens1
    Dens=NewDens

  ### Molarity Molality Input###
  if HUnitVariable=="1":
    MolarPu=input("Enter Plutonium Molarity (mol Pu/L solution): ")
    Pu=float(MolarPu)

    molalH=input("Enter HNO3 Molality (mol HNO3/kg solvent): ")
    mH=float(molalH)

    while error>0.000001:
      GuessDens=NewDens
      H=(1000*NewDens-MPUNO34*Pu)*mH/(1000+MHNO3*mH)
      mPu=Pu*mH/H
      Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
      error=(abs(Dens1-GuessDens))/(Dens1)
      NewDens=Dens1
    Dens=NewDens
    H=Dens*mH/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)

### Concentration Concentration Input###
if PuUnitVariable=="2":
  if HUnitVariable=="2":
    ConcentrationPu=input("Enter Plutonium concentration (g Pu/L solution): ")
    CPu=float(ConcentrationPu)
    Pu=CPu/(MPu)

    ConcentrationH=input("Enter H concentration (g H/L solution): ")
    CH=float(ConcentrationH)
    H=CH/(MH)

    while error>0.000001:
      GuessDens=NewDens
```

```
            mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
            mH=mPu*H/Pu
            Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
            error=(abs(Dens1-GuessDens))/(Dens1)
            NewDens=Dens1

          Dens=NewDens

        ### Concentration Molarity Input###
        if HUnitVariable=="3":
          ConcentrationPu=input("Enter Plutonium concentration (g Pu/L solution): ")
          CPu=float(ConcentrationPu)
          Pu=CPu/(MPu)

          MolarH=input("Enter HNO3 Molarity (mol HNO3/L solution): ")
          H=float(MolarH)

          while error>0.000001:
            GuessDens=NewDens
            mPu=1000*Pu/((1000*GuessDens)-(MPUNO34*Pu)-(MHNO3*H))
            mH=mPu*H/Pu
            Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
            error=(abs(Dens1-GuessDens))/(Dens1)
            NewDens=Dens1
          Dens=NewDens

          ### Concentration Molality Input###
          if HUnitVariable=="1":
            ConcentrationPu=input("Enter Plutonium concentration (g Pu/L solution): ")
            CPu=float(ConcentrationPu)
            Pu=CPu/(MPu)

            molalH=input("Enter HNO3 Molality (mol HNO3/kg solvent): ")
            mH=float(molalH)

            while error>0.000001:
              GuessDens=NewDens
              H=(1000*NewDens-MPUNO34*Pu)*mH/(1000+MHNO3*mH)
              mPu=Pu*mH/mPu
              Dens1=(do+A*mPu+B*mH+C*T+D*mPu*mH+E*mPu*T+F*mPu*mPu+G*mH*mH)
              error=(abs(Dens1-GuessDens))/(Dens1)
              NewDens=Dens1
            Dens=NewDens
            H=Dens*mH/(1+(MHNO3/1000)*mH+(MPUNO34/1000)*mPu)
if mPu>3.3 or mH<0 or mH>6.27 or T<22.5 or T>60:
  print("ERROR: Data entered is out of bounds of equation range.")
else:
  for i in range(0,3):
    print("")
  print("Your solution density is: ", Dens, " kg/L")
  for i in range(0,3):
    print("")
  wPuMC=[]
  if Question=="Y" or Question=="y":
    QwPu=input("Would you like to change the weight percents of Plutonium isotopes?(Y/N):")
    if QwPu=="y" or QwPu=="Y":
      w94238=0
      w94239=0
      w94240=0
      w94241=0
      w94242=0
      w95241=0
      Purity=input('Enter Desired weight percent of Pu-239 (0-100): ')
      w94239=float(Purity)/100
      NPu239=(((MPu*Pu/MPu239)*w94239/Cm3toL)*Na/Barntocm2)
      wPuMC.append([94239,NPu239])

      if w94239> 1:
        print("ERROR: Must enter a weight percent from 0-100")
```

```
                elif w94239< 1:
                    Purity2=input('Enter Desired weight percent of Pu-240 (0-100): ')
                    w94240=float(Purity2)/100
                    wPuTotal=w94240+w94239
                    NPu240=(((MPu*Pu/MPu240)*w94240/Cm3toL)*Na/Barntocm2)
                    wPuMC.append([94240,NPu240])

                    if wPuTotal> 1:
                        print("ERROR: Weight percents add up to greater than 100")
                    elif wPuTotal< 1:
                        Purity3=input('Enter Desired weight percent of Pu-241 (0-100): ')
                        w94241=float(Purity3)/100
                        wPuTotal=w94241+w94239+w94240
                        NPu241=(((MPu*Pu/MPu241)*w94241/Cm3toL)*Na/Barntocm2)
                        wPuMC.append([94241,NPu241])
                        if wPuTotal> 1:
                            print("ERROR: Weight percents add up to greater than 100")
                        elif wPuTotal< 1:
                            Purity4=input('Enter Desired weight percent of Pu-242 (0-100): ')
                            w94242=float(Purity4)/100
                            wPuTotal=w94242+w94239+w94240+w94241
                            NPu242=(((MPu*Pu/MPu242)*w94242/Cm3toL)*Na/Barntocm2)
                            wPuMC.append([94242,NPu242])
                            if wPuTotal> 1:
                                print("ERROR: Weight percents add up to greater than 100")
                            elif wPuTotal< 1:
                                Purity5=input('Enter Desired weight percent of Pu-238 (0-100): ')
                                w94238=float(Purity5)/100
                                wPuTotal=w94238+w94239+w94240+w94241+w94242
                                NPu238=(((MPu*Pu/MPu238)*w94238/Cm3toL)*Na/Barntocm2)
                                wPuMC.append([94238,NPu238])
                                if wPuTotal== 1:
                                    Purity7=input("Would you like to add Am-241? (Y/N):")
                                    if Purity7=="Y" or Purity7=="y":
                                        AmCheck=input("Would you like to enter Am-241 content as a wt.% of Pu now or as an impurity
concentration later?(%/I): ")
                                        if AmCheck=="%":
                                            AmCont=input("Enter desired weight percent Am-241 (0-100):")
                                            w95241=float(AmCont)/100
                                            Am241=(MPu*Pu/MAm241)*w95241
                                            NAm241=((Am241/Cm3toL)*Na/Barntocm2)
                                            wPuMC.append([95241,NAm241])
                                elif wPuTotal> 1:
                                    print("ERROR: Weight percents add up to greater than 100")
                                elif wPuTotal< 1:
                                    Purity6=input("Would you like to add Am-241? (Y/N):")
                                    if Purity6=="Y" or Purity6=="y":
                                        AmCheck2=input("Would you like to enter Am-241 content as a wt.% of Pu now or as an impurity
concentration later?(%/I): ")
                                        if AmCheck2=="%":
                                            AmCont2=input("Enter desired weight percent Am-241 (0-100):")
                                            w95241=float(AmCont2)/100
                                            Am241=(MPu*Pu/MAm241)*w95241
                                            NAm241=((Am241/Cm3toL)*Na/Barntocm2)
                                            wPuMC.append([95241,NAm241])
            elif QwPu=="N" or QwPu=="n":
                NPu238=(((MPu*Pu/MPu238)*w94238/Cm3toL)*Na/Barntocm2)
                wPuMC.append([94238,NPu238])
                NPu239=(((MPu*Pu/MPu239)*w94239/Cm3toL)*Na/Barntocm2)
                wPuMC.append([94239,NPu239])
                NPu240=(((MPu*Pu/MPu240)*w94240/Cm3toL)*Na/Barntocm2)
                wPuMC.append([94240,NPu240])
                NPu241=(((MPu*Pu/MPu241)*w94241/Cm3toL)*Na/Barntocm2)
                wPuMC.append([94241,NPu241])
                NPu242=(((MPu*Pu/MPu242)*w94242/Cm3toL)*Na/Barntocm2)
                wPuMC.append([94242,NPu242])
                NAm241=(((MPu*Pu/MAm241)*w95241/Cm3toL)*Na/Barntocm2)
                wPuMC.append([95241,NAm241])
            print("Would you like to:")
```

```
print("1. Re-enter the entire impurity list.")
print("2. Re-enter the impurity concentrations only.")
print("3. Leave the impurity list exactly as is.")
Qimps=input("(1/2/3):")
if Qimps=="1":
   ImpQ=input("Enter number of impurities desired:")
   NumImp=int(ImpQ)
   Implist=[]
   if NumImp>0:
      for i in range(0,NumImp):
         ImpurityN=i+1
         print(" ")
         print(' ')
         print("For impurity #",ImpurityN)
         ZAIDQ=input("Enter impurity ZAID:")
         ZAID=float(ZAIDQ)
         CQ=input("Enter impurity concentration (g/L):")
         CImp=float(CQ)
         MQ=input("Enter impurity Molar Mass (g/mol):")
         MImp=float(MQ)
         BQ=input("Enter the number of bonds the impurity has with NO3- (ex. Fe has 3):")
         BImp=float(BQ)
         Imp=CImp/MImp
         NImp=((Imp/Cm3toL)*Na/Barntocm2)
         NContribution=BImp*Imp
         NImpCont.append(NContribution)
         Implist.append([ZAID,NImp])

   NImpContribution=sum(NImpCont)
elif Qimps=="2":

   for i in range(0,len(implist2)):
      print("")
      print('')
      print("For impurity with ZAID#",implist2[i][0],":")
      newCImp=input("Please enter the concentration in (g/L):")
      CImp=float(newCImp)
      MImp=implist2[i][2]
      BImp=implist2[i][3]
      Imp=CImp/MImp
      NImp=((Imp/Cm3toL)*Na/Barntocm2)
      NContribution=BImp*Imp
      NImpCont[i]=NContribution
      Implist[i]=[implist2[i][0],NImp]
   NImpContribution=sum(NImpCont)
PuNO34=Pu
HNO3=H
if w95241>0:
   N=(4*PuNO34)+(HNO3)+NImpContribution+(3*Am241)
else:
   N=(4*PuNO34)+(HNO3)+NImpContribution
rhoPUNO34=Pu*MPUNO34/1000
rhoHNO3=H*MHNO3/1000
rhoSolvent=Dens-rhoPUNO34-rhoHNO3
Solvent=rhoSolvent*1000/MSolvent

Htotal=Solvent*2+HNO3*1
Ototal=(Solvent*1)+N*3
NH=((Htotal/Cm3toL)*Na/Barntocm2)
NN=((N/Cm3toL)*Na/Barntocm2)
NO=((Ototal/Cm3toL)*Na/Barntocm2)

for i in range(0,len(wPuMC)):
   if (wPuMC[i][1])>0:
      MaterialCard.append([str(wPuMC[i][0]),str(wPuMC[i][1])])

MaterialCard.append(['1001',str(NH)])
MaterialCard.append(['7014',str(NN)])
MaterialCard.append(['8016',str(NO)])
```

```
for i in range(0,len(Implist)):
   if (Implist[i][1])>0:
      MaterialCard.append([str(Implist[i][0]),str(Implist[i][1])])

Lm=len(MaterialCard)
for i in range(0,5):
   print("")

print("###  Material Card  ###")
for i in range(0,Lm):
   print(MaterialCard[i][0],MaterialCard[i][1])

for i in range(0,5):
   print("")

if Q=="Y" or Q=="y":
   print("")
   print("Reminder: In input file add @ symbol where solution density is required.")
   print("")
   file2=input("Enter Entire MCNP File Name (file.txt): ")

   FQ=input("If you would like to specify new file name enter name now (Please include .txt). If not type 1.:")
   if FQ=="1":
      newfile="newfile.txt"
   else:
      newfile=FQ

   with open(file2) as f:
      lines=f.readlines()
   L=len(lines)
   Rd=[0]*L

   for i in range(0,L):
      strng=lines[i]
      new=strng.split()
      Rd[i]=new


   for i in range(0, L):

      if (Rd[i]==[]):
         continue
      else:
         Temp=Rd[i][0]
         if (Temp=='c'):
            continue
         else:
            Temp1=Rd[i]

            for j in range(0, len(Temp1)):
               if (Rd[i][j]=='@'):
                  DensI=i
                  DensJ=j
                  break

   Rd[DensI][DensJ]=str(-Dens)
   Matnum=str(Rd[DensI][DensJ-1])
   MatKey='m'+Matnum
   mtnm=float(Rd[DensI][DensJ-1])
   Space='    '
   if mtnm<10:
      Space1="   "
   elif mtnm<100:
      Space1="  "
   elif mtnm<1000:
      Space1=' '

   for i in range(0,Lm):
      if i==0:
```

```
                    Md[i]=[MatKey,MaterialCard[0][0],MaterialCard[0][1]]
                    new1=Md[i][0]+Space1+Md[i][1]+glue+Md[i][2]+'\n'
                    FMatCard[i]=new1
                else:
                    new1=Space+glue.join(MaterialCard[i])+'\n'
                    FMatCard.append(new1)

            for i in range(DensI, L):

                if (Rd[i]==[]):
                    continue
                else:
                    Temp=Rd[i][0]
                    if (Temp=='c'):
                        continue
                    else:
                        Temp1=Rd[i]

                        for j in range(0, len(Temp1)):
                            if (Rd[i][j]==MatKey):
                                MatI=i

            for i in range(0,L):
                if i==DensI:
                    newdens=glue.join(Rd[i])+'\n'
                    Flines.append(newdens)
                elif i==MatI:
                    for k in range(0,Lm):
                        Flines.append(FMatCard[k])

                else:
                    Flines.append(lines[i])

            with open(newfile,'w') as f:
                for i in range(0,L+Lm-1):
                    strg=Flines[i]
                    f.writelines(strg)
            print("")
            print("File Complete!")
        elif Qmulti=='n' or Qmulti=="N":
            indicator=0
        else:
            print("ERROR: please enter only Y or N")

else:
    print("")
    print("")
    print("ERROR: Please only enter 1,2 or 3.")
```

# REFERNECES

[1] American Nuclear Society. (2014). Nuclear Criticality Safety in Operations with Fissionable Materials Outside Reactors. (ANSI/ANS-8.1-2014(R2018))

[2] Charrin, N., Moisy, P., & Blanc, P. (2000, 01). Determination of Fictive Binary Data for Plutonium(IV) Nitrate. *Radiochimica Acta, 88*. doi:10.1524/ract.2000.88.1.025

[3] Söhnel,, O., & Novotný, P. (1985). Densities of Aqueous Solutions of Inorganic Substances. Amsterdam: Elsevier.

[4] Hofstetter, K. J., Bowers, D. L., & Kemmerlin, R. P. (1973). *Measurement of Acidity and Density of Plutonium Solutions.* doi:10.2172/6635724

[5] Leclaire, N. P., Anno, J. A., Courtois, G., Dannus, P., Poullot, G., & Rouyer, V. (2003). Criticality Calculations Using the Isopiestic Density Law of Actinide Nitrates. *Nuclear Technology, 144*(3), 303-323. doi:10.13182/NT03-A3447

[6] Weber, C. F., & Hopper, C. M. (2006). Application of the Pitzer Method for Modeling Densities of Actinide Solutions in the SCALE Code System. *Nuclear Technology, 153*(1), 1-8. doi:10.13182/NT06-A3684

[7] B. T. Rearden and M. A. Jessee, Eds., SCALE Code System, ORNL/TM-2005/39, Version 6.2.3, Oak Ridge National Laboratory, Oak Ridge, Tennessee (2018). Available from Radiation Safety Information Computational Center as CCC-834