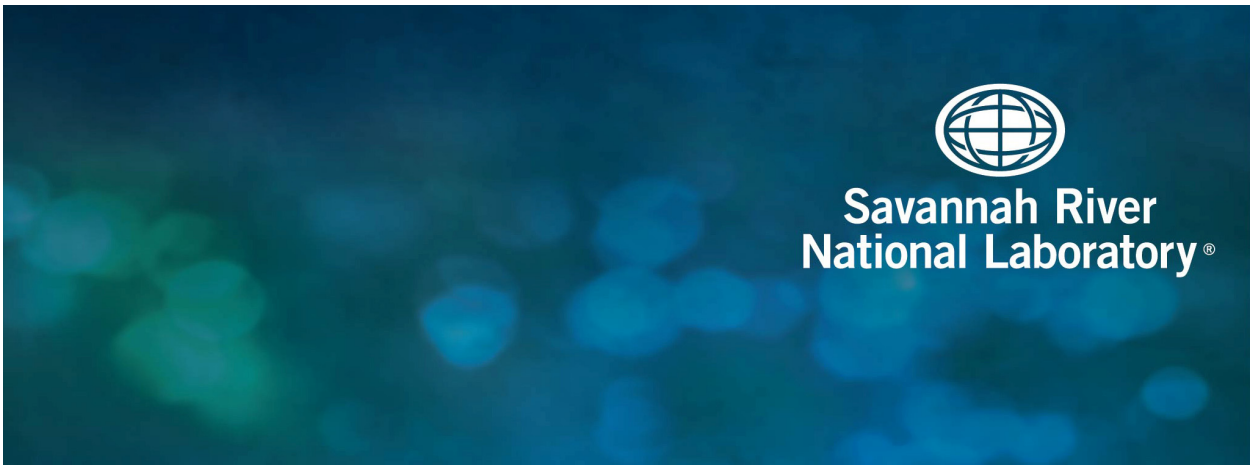
Contract No:

This document was prepared in conjunction with work accomplished under Contract No. 89303321CEM000080 with the U.S. Department of Energy (DOE) Office of Environmental Management (EM).

Disclaimer:



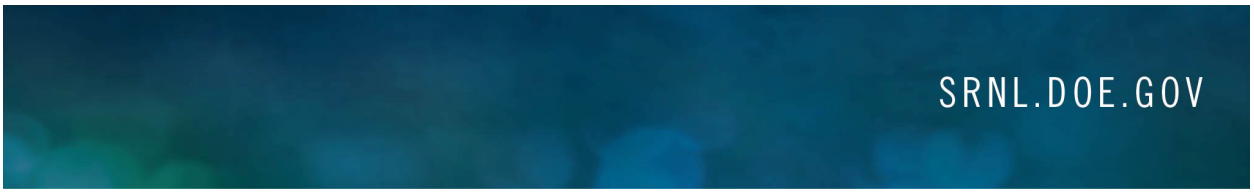
A U.S. DEPARTMENT OF ENERGY NATIONAL LAB • SAVANNAH RIVER SITE • AIKEN, SC • USA

E-Area Low Level Waste Generator Inventory Uncertainty Estimation

Stephanie B. Taylor
Tad S. Whiteside

January 2022

SRNL-STI-2021-00276, Revision 0



DISCLAIMER

Printed in the United States of America
Prepared for
U.S. Department of Energy

Keywords: E-Area
SWE
Inventory
Uncertainty

Retention: *permanent*

E-Area Low Level Waste Generator Inventory Uncertainty Estimation

Stephanie B. Taylor
Tad S. Whiteside

January 2022

Savannah River National Laboratory is operated by Battelle Savannah River Alliance for the U.S. Department of Energy under Contract No. 89303321CEM000080.

Reviews and Approvals

Authors:

S. B. Taylor, Contractor Assurance — Date

T. S. Whiteside, Nuclear Measurements — Date

Technical Review:

D. Li, Earth and Biological Systems — Date

Approval:

J. J. Mayer, R&D Manager — Date
Earth and Biological Systems

Executive Summary

This report describes the method used to estimate the uncertainty and bias in the low-level waste inventory, as reported by Waste Generators, for waste disposed in the E-Area Low-Level Waste Facility at SRS. The waste cut uncertainty and bias is summed to calculate the current total uncertainty and bias in each disposal unit in the E-Area Low-Level Waste Facility. This information is used to estimate the future uncertainty and bias when all disposal units are closed.

For most waste cuts, there are two areas of inventory uncertainty and bias: in the isotopic characterization of the waste stream and in the measurement of total waste cut activity. Waste stream characterization is accomplished through a combination of process knowledge and analytical measurements. This characterization is documented by a technical baseline that describes the relative quantity of each isotope present in the waste stream. The total activity of each individual waste cut is usually measured directly or is calculated through a ratio of activity per weight or activity per volume. The waste cut total activity is multiplied by the waste stream isotopic fraction to calculate the total isotopic activity present in the waste cut. The waste cut uncertainty and bias is calculated by analyzing the uncertainties and biases in the waste stream characterization and waste cut measurements and adding the uncertainties in quadrature and multiplying the reported isotopic activities by the bias factors.

The current isotopic inventory, uncertainty, and bias of a disposal unit is the sum of the currently disposed individual waste cut isotopic activities, uncertainties, and biases. The future uncertainty and bias is estimated based on the average "worst-case" (largest) uncertainties and biases of currently closed and operational disposal units.

This report details the method, data sources, and calculation techniques used to calculate these values. It also presents some of the difficulties in working with the reported data and gives examples of the calculations and analysis. The software used to carryout the analysis is included and the location of the complete individual analyses is given.

After discussion with waste generators, examination of multiple waste characterization documents, and a review of SRS Manual 1S, the reported ELLWF radionuclide activities are clearly conservative and likely represent reasonable upper bounds for the true activities. However, quantifying the bias and uncertainty in each waste cut requires detailed analysis of the documentation for each waste cut measurement and waste stream characterization.

Contents

List of Abbreviations **viii**

1 Introduction **1**

2 Information Sources **3**

3 Waste Stream Characterization Method **3**
3.1 Description . . . 3
3.2 Baseline Assumptions . . . 4

4 Waste Cut Measurement Methods **4**
4.1 CHAR BY PACK . . . 4
4.1.1 Description . . . 4
4.1.2 Baseline Assumptions . . . 5
4.2 Dose-to-Curie . . . 5
4.2.1 Description . . . 5
4.2.2 Baseline Assumptions . . . 5
4.3 Smear-to-Curie . . . 5
4.3.1 Description . . . 5
4.3.2 Baseline Assumptions . . . 6
4.4 RAD Weight or Volume . . . 6
4.4.1 Description . . . 6
4.4.2 Baseline Assumptions . . . 6

5 Calculating Waste Cut Activity, Uncertainty, and Bias **7**

6 Calculating Disposal Unit Activity, Uncertainty, and Bias **8**

7 Simple Example Calculations **9**
7.1 A single waste cut with a single isotope in a DU . . . 9
7.2 A single waste cut with multiple isotopes in a DU . . . 9
7.3 Multiple waste cuts (same stream) in a DU . . . 10
7.4 Multiple waste cuts (different streams) in a DU . . . 10

8 Usage of Uncertainty and Bias Calculations **11**

9 Estimating Future Inventory, Uncertainty, and Bias **13**

10 Method Implementation **16**
10.1 Code . . . 16
10.2 Prioritizing Waste Stream Analysis . . . 17
10.3 Waste Stream Analysis . . . 20

10.4 Technical Baseline Analysis 24
10.4.1 Overview 24
10.4.2 Uncertainty Analysis 24
10.4.3 Example Technical Baseline Analysis 26

11 Other Conservatisms **33**

12 Recommendations **33**

13 Conclusions **34**

14 References **35**

Appendix A Commands to execute method **37**

Appendix B Code **38**

Appendix C Prioritized Waste Streams **150**

List of Abbreviations

CBP Characterization By Package

CDF Cumulative Distribution Function

CIG Cemented In Grout

CWTS Consolidated Waste Tracking System

DOE Department of Energy

dpm disintegrations per minute

DTC Dose-to-Curie

DU Disposal Unit

ELLWF E-Area Low Level Waste Facility

LLD Lower-Limit of Detection

LFRG Low-Level Waste Disposal Facility Federal Review Group

PA Performance Assessment

PDF Probability distribution function

RAD Rad Volume/Weight

SOF Sum Of Fractions

SRNL Savannah River National Laboratory

SRS Savannah River Site

STC Smear-to-Curie

SWF Special Waste Form

WITS Waste Information Tracking System

WSC Waste Stream Characterization

WSCF Waste Stream Characterization Form

1 Introduction

The Savannah River Site's (SRS) E-Area Low Level Waste Facility (ELLWF) is the disposal facility for low-level nuclear waste generated by nuclear processes at SRS and external facilities which have agreements with U.S. DOE to dispose of the waste at SRS. The ELLWF consists of 7 closed disposal units (DU), 11 open disposal units, and 14 future disposal units (Figure 1.1). As part of the environmental compliance requirements, performance assessments (PA) must be conducted for this facility to ensure the disposed waste will not impact human health or the environment (DOE, 2017). One of the comments received from the LFRG after the 2008 PA was to evaluate the impact of uncertainties on the expected environmental performance of the ELLWF after closure (DiSanza et al., 2008). This work describes the method used to estimate the uncertainty and bias in the Waste Generator-reported isotopic activities. The estimated uncertainty and bias are then used in additional modeling efforts to address those comments.

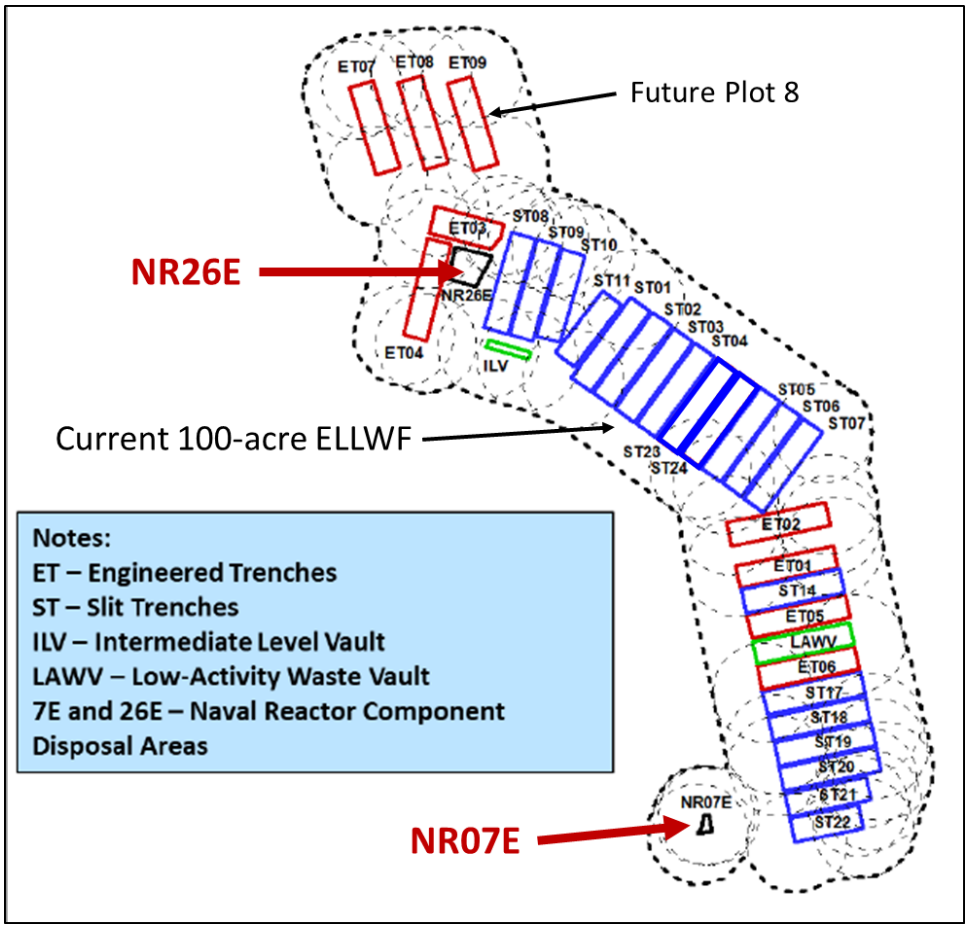


Figure 1.1: The E-Area Low-Level Waste Facility showing Current and Future Disposal Units.

The generation and management of waste to be disposed in the ELLWF must follow the requirements

of the *Savannah River Site Radioactive Waste Requirements Manual 1S* (SRS Manual 1S) (SRS, 2021). Waste that comes from the same location, job, task, etc. and has been characterized using the same method is part of a *waste stream*. Individual bags of waste, items, or other components, such as soil or rubble, from this waste stream are called *waste cuts*. One or more waste cuts are placed in a *waste container*. The waste cuts in a waste container may be from one or more waste streams, and the waste containers may be nested. An example would be a 5-gallon metal drum placed in a steel box placed in a Sealand container. In general, the isotopic composition of a waste stream is determined through a combination of analytical measurements and process knowledge and then the individual waste cut or entire waste container is characterized by one of four different methods which determines the activity of each isotope in the waste cut.

The radiological characterization process is divided into two phases:

1. The first phase determines the radionuclides, and their fraction of total activity, present in a waste stream, with an emphasis on reporting PA radionuclides (SRS, 2021). This is called the Waste Stream Characterization (WSC). The technical baseline used in the WSC document should contain enough information to determine how the WSC was completed.

2. The second phase determines the total activity within a waste cut. The method used to determine this activity is described within the WSC document. To fully understand the uncertainty in the activity measurement, the documentation of the measurement of each waste cut or container is also needed. Documentation of these results is not referenced in the Waste Information Tracking system. For some waste cuts, this information is in the technical baseline documents; however, for others, this documentation is unavailable.

After a waste cut is characterized and is part of a container ready to be shipped to the ELLWF, the waste stream, waste cut, and waste container information is entered into the Waste Information Tracking System (WITS).

This work uses the information found in WITS and the technical baseline documents as the inputs to a method which estimates, prioritizes, and refines the uncertainty and bias in the reported activity of each current and future isotope found in each current and future DU. This method first determines the uncertainty and bias in the isotopic composition (per waste stream) and then in the total measured activity of the individual waste cuts. This information is combined to calculate the activity, uncertainty, and bias of each isotope in each waste cut. Next, the waste cut isotopic activities, uncertainties, and biases are used to determine the total isotopic activity, uncertainty, and bias in each closed and operational DU. Finally, these results are used to determine the expected isotopic composition of future DUs.

After discussion with waste generators, examination of multiple waste characterization documents, and a review of SRS Manual 1S, the reported ELLWF radionuclide activities are clearly conservative and likely represent reasonable upper bounds for the true activities. However, quantifying the bias and uncertainty in each waste cut requires analysis of the documentation for both the waste cut measurement and the waste stream characterization.

2 Information Sources

Three sources of information are needed to completely perform the analysis. These are WITS, the technical baseline documents, and the individual waste cut measurements.

The waste cut isotopic activity, waste container, and waste stream information are obtained from the Waste Information Tracking System (WITS)/Consolidated Waste Tracking System (CWTS). WITS is the original database used at the ELLWF and its data has been merged with CWTS in 2021 (SRNS, 2021). This work will refer to the combined database system as WITS.

The waste stream information in WITS references the waste stream characterization form (WSCF). The WSCF summarizes the isotopic composition, expected distribution, waste cut measurement method, and the technical baseline used to describe the waste coming to ELLWF from a specific project or process. For example, the rubble from the demolition of a building will be characterized with the expected isotopic composition of the rubble. This characterization will typically be done through a combination of process knowledge (what was in the building and what is expected to be in the rubble) and analytical measurements (samples of rubble sent to a lab for isotopic analysis). This information is synthesized into a technical baseline document that describes the relative quantity of isotopes in the waste stream. Usually this document also describes how the individual waste cuts from the project/process will be measured.

The total activity of each waste cut is in WITS, however the documentation that contains the results of those individual waste cut measurements is not referenced within WITS nor is it on the WSCF. Some of the technical baseline documents contain the measured activities of all the waste cuts in a waste stream, however usually this information is incomplete or not present. Therefore, the majority of waste cut measurements use the baseline uncertainties described in Section 4.

3 Waste Stream Characterization Method

3.1 Description

A waste stream is the waste coming from a nuclear process. Before waste is disposed, the isotopic composition of the waste and fraction of total activity (relative isotopic quantity) in the waste stream is determined through a combination of analytical measurements and process knowledge. This determination is made in the technical baseline documents (Section 10.4) and is subsequently captured on the WSCF and in WITS.

Most of the WSC documents are found in the EAV Low-Level Waste Stream Characterizations (LLWSC) Lotus Notes database (Dunbar, 2009). However, some are not present, and tracking down that information requires working with the Solid Waste customer. Solid Waste (Nat Roddy) appears to have access to the paper/offline electronic copies of this information, however where these documents are located is not clear. Note: email sent to Nat Roddy asking this question on 10/13/2021. He said prior to the Notes database they were filed in EDWS. However, when I (Tad Whiteside) have searched EDWS for these waste stream documents, I have not located them.

3.2 Baseline Assumptions

Initially we assume the relative isotopic composition is unbiased (bias factor of 1) and has a default characterization uncertainty of 10%. The exception is for the waste cuts characterized by package – those have a waste stream characterization uncertainty of 0%, since each waste cut is individually characterized.

When a conservatism is identified, the bias is quantified, if possible. Otherwise, when identified, but not quantified, we assume the conservatism causes the true value to be 90% of the reported value (bias factor of 0.9).

4 Waste Cut Measurement Methods

There are four methods authorized by SRS Manual 1S to measure waste cut activity. These methods are labeled in WITS as CHAR BY PACK (CBP), Dose-To-Curie (DTC), Smear-To-Curie (STC), and RAD weight or volume (RAD). As shown in Table 4.1, a 2017 survey of WITS indicated that the RAD and DTC measurement methods are used on the majority of waste cuts disposed in the ELLWF, (Whiteside, 2017).

Table 4.1: Methods Used to Measure Waste Activity

Calculation Method	Num Waste Cuts	% Waste Cuts
CHAR BY PACK	10,589	5%
Dose-to-Curie	63,132	28%
RAD weight/volume	149,109	65%
Smear-to-Curie	3,753	2%

The following sections describe each measurement technique and the baseline assumptions made about uncertainty and bias.

4.1 CHAR BY PACK

4.1.1 Description

Waste cuts using the CBP method do not use a relative isotopic waste stream composition. Each package may have a unique distribution of radionuclide contents determined based on process knowledge, analytical assay, or a combination of both. Various techniques may be employed to estimate the inventory, including but not limited to, pedigree of radioactive sources and radioactive decay (i.e., calculations based on time of manufacture to disposal), direct assay (i.e., non-destructive gamma spectroscopy method used in determining activity levels in TRU waste that is screened below 100 nCi/g to be disposed of as LLW), and materials accountability calculations (i.e., mass

balance from the influent and effluent of materials through a process where waste is generated) (SRS, 2021).

4.1.2 Baseline Assumptions

Packages with the CBP designation have had their radionuclides individually characterized by analytical analysis or otherwise carefully derived. Because each package is individually characterized, it is assumed these values are unbiased (bias factor is 1) and have a low uncertainty (5%). This uncertainty value is the lowest that would typically be reported by laboratory measurements (DiPrete, 2021).

4.2 Dose-to-Curie

4.2.1 Description

Waste cuts with a total activity calculated using DTC rely on measuring the maximum measured gamma photon energy which is converted to a dose equivalent rate in mrem/hour. A conversion factor is developed using software called SRS-DTC (SRS, 2014). The SRS-DTC software accounts for the geometry of the package, the density of the waste matrix, and the fractional proportion of gamma emitting radionuclides to the total radionuclide distribution. This method requires the use of a predefined radionuclide distribution (from the waste stream technical baseline) so that a ratio of the major gamma emitting radionuclide (e.g., Co-60 or Cs-137) to the total radionuclide distribution can be established. As an example, for a rectangular package such as a B-25 box, the generator obtains measurements on each of the four vertical sides. The maximum dose equivalent rate is then used to convert to radionuclide activity based on the factor from SRS-DTC for the given package configuration. This total activity is then distributed according to the predefined isotopic distribution as described in the waste stream characterization.

4.2.2 Baseline Assumptions

Chapter 3, Section 5.10.4 in the SRS 1S Manual (SRS, 2021) specifies using the Bicron Micro Rem LE or equivalent for all DTC measurements. This instrument has a reported accuracy of within 10% for Cs-137 between 20% and 100% of full scale on any range (Fisher-Scientific, 2007). A baseline measurement uncertainty of 10% is used for DTC measurements.

There is a 100% systematic bias in the DTC measurement technique for reporting isotopic activities (Cook et al., 2011; INEL, 1995; Nevada Test Site, 2006; Verst, 2021). Thus, the bias factor is set to 0.5 for waste cuts measured with this technique.

4.3 Smear-to-Curie

4.3.1 Description

The STC method establishes a package's activity by applying the maximum contamination levels in units of disintegrations per minute (dpm) to the distribution of radionuclides described in the waste

stream technical baseline analysis. The contamination may be reported based on maximum dpm for a process area, which is typically the case for non-equipment waste. For equipment waste, smears are typically taken directly from the waste (i.e., equipment), with the highest observed dpm rate used for calculation of activity. The maximum alpha, beta/gamma, neutron, etc. values, along with the weight of the waste, conversion factors for mass, surface area, and fractional portion that the measured contamination represents (i.e., alpha or beta/gamma) of the total radionuclide distribution, are used to produce the total activity for a waste package (SRS, 2021). This method typically employs multiple smears analyzed using the "best reasonably available measurement method."

4.3.2 Baseline Assumptions

Because each container is individually characterized, it is assumed these values are unbiased (bias factor of 1) and have a low uncertainty (10%). This assumption is conservative, as values are based on maximum dpm for a location or smear.

Near the end of this method implementation, we discovered on page 22 of the document WSRC-RP-95-00964, Rev. 0 a description of how smear data typically has a 50% uncertainty at 95% confidence (2-sigma) due to a 20 second count time (Johnson, 1995). If this count time is typical of STC measurements, in future analysis, the baseline assumption should be revised to 25% uncertainty (1-sigma).

4.4 RAD Weight or Volume

4.4.1 Description

The RAD calculation method, like the DTC and STC methods, uses the relative isotopic distribution as described in the waste stream technical baseline analysis. To determine the total activity, generators first use either the DTC or STC technique to determine the activity and isotopic distribution for a single waste measurement (usually this is of a waste cut, but it does not have to be). Then, that measurement is used to create a per mass (or per volume) conversion factor for the waste stream. They next either weigh each waste cut container or estimate the volume by some prescribed method (e.g., visual estimate) and apply the appropriate activity mass or volume conversion to arrive at total activity. The predefined distribution is then used to scale the total activity to each of the radionuclides in the waste cut (SRS, 2021).

4.4.2 Baseline Assumptions

Since the measurement method for the RAD method is based on either DTC or STC, the bias factor is either be 0.5 (DTC) or 1 (STC). To determine which method (DTC/STC) is used in waste cuts characterized using this method, each waste stream's technical baseline document must be analyzed.

As a baseline assumption and using the information on Table 4.1, it is assumed the same proportion of DTC (63,132/(63,132+3,753) = 94%) to STC (3,753/(63,132+3,753) = 6%) measurements make

up the RAD measurements. Therefore, a random choice of bias factor is assigned to the RAD waste cuts in the same proportion as DTC to STC: 94% have a bias factor of 0.5 and 6% have a bias factor of 1.

Since the technique used (DTC/STC) is not specified and each cut is individually characterized, it is assumed these measurements have an uncertainty of 10%.

5 Calculating Waste Cut Activity, Uncertainty, and Bias

The activity (A) of each isotope in each waste cut is usually determined as the product of the measured activity of a waste cut (M) and the fraction of activity associated with that isotope in the waste stream (C):

$$A = M \cdot C \tag{5.1}$$

The DTC, STC, and RAD methods use both factors. In the CBP method, only M is used, and the quantity C equals 1 with 0 uncertainty.

Since the total activity of an isotope is the product of M and C, the total relative uncertainty (U_A) in the reported activity of an isotope is calculated by:

$$U_A = \sqrt{{U_M}^2 + {U_C}^2} \tag{5.2}$$

where U_M is the relative uncertainty of the measurement of the waste cut's total activity, and U_C is the relative uncertainty associated with this isotope's fraction of the total waste stream. The uncertainties are added in quadrature because we assume they are independent and the errors are random (Taylor, 1997). The absolute uncertainty of the reported value is simply the product of A and U_A.

Depending on the methods and techniques used, both M and C may contain systematic biases. Generally, these biases are conservative – thereby making the reported activity larger than the actual or "true" activity. The bias factors (of M and C) are recorded as fractions of the true value. For example, a bias factor of 1 means the reported value is unbiased, a bias factor of 0.9 means the reported value is biased-high and the true value is 90% of the reported value, and a bias factor of 1.1 means the reported value is biased-low and the true value is 110% of the reported value. The "true" activity, A_T, is calculated by multiplying M and C by their bias factors.

The full expression of true activity of a waste cut (Ci) is expressed as:

$$A_T \pm U_{A_T} = A_W \cdot \delta_{A_W} \pm U_{A_W} = (M_W \cdot \delta_M \pm U_M) \cdot (C_W \cdot \delta_C \pm U_C) \tag{5.3}$$

where:

U_{A_T} is the relative uncertainty in the true activity of the item, equal to U_W

A_W is the WITS reported activity of the item (Ci)

δ_{A_W} is the product of the measurement and characterization biases (dimensionless)

U_{A_W} is the relative uncertainty of the activity of the item (dimensionless), calculated by 5.2

M_W is the total activity (measurement) of the item (Ci)

C_W is the fraction of this isotope in the waste stream (characterization) of the item (dimensionless)

δ_M is the measurement bias (dimensionless)

δ_C is the characterization bias (dimensionless)

U_M is the relative uncertainty in the measurement value of the item (dimensionless)

U_C is the relative uncertainty in the characterization value of the item (dimensionless)

6 Calculating Disposal Unit Activity, Uncertainty, and Bias

The total current inventory of a radionuclide in a DU, as reported by WITS, is defined as I_W (Ci), and is simply the sum of the WITS-reported activities of each waste cut, $A_W \pm U_{A_W}$, in that DU:

$$I_W \pm unc_{I_W} = \sum_{a}^{i} (A_W \pm U_{A_W}) \tag{6.1}$$

where the absolute uncertainty in this total inventory value is calculated as:

$$unc_{I_W} = \sqrt{{unc_{W_a}}^2 + {unc_{W_b}}^2 + \cdots + {unc_{W_i}}^2} \tag{6.2}$$

where unc_{W_a} through unc_{W_i} refer to the absolute uncertainty in Ci ($A_W \cdot U_{A_W}$) for each waste cut.

Similarly, the total true isotopic activity in a DU, I_T (Ci), is expressed as the sum of the true activities of each waste cut ($A_T \pm U_{A_T}$) in that DU:

$$I_T \pm unc_{I_T} = \sum_{a}^{i} (A_T \pm U_{A_T}) \tag{6.3}$$

where the absolute uncertainty in the total inventory value is calculated as:

$$unc_{I_T} = \sqrt{{unc_{T_a}}^2 + {unc_{T_b}}^2 + \cdots + {unc_{T_i}}^2} \tag{6.4}$$

where unc_{T_a} through unc_{T_i} refer to the absolute uncertainty in Ci ($A_T \cdot U_{A_T}$) for each waste cut.

These values are related through the equation:

$$I_T \pm unc_{I_T} = I_W \cdot \delta_B \pm unc_{I_W} \tag{6.5}$$

where δ_B is the total inventory bias in the DU (dimensionless), calculated from (I_T/I_W) . The relative uncertainty in a DU, U_{DU} , is $\left(\frac{unc_{I_T}}{I_T}\right)$ or $\left(\frac{unc_{I_W}}{I_W}\right)$.

7 Simple Example Calculations

As described in Section 6, to determine an isotope's activity, uncertainty, and bias in a DU, first the activity, uncertainty, and bias corrected "true" activity of each isotope in each waste cut are calculated. Then the DU activity is calculated by summing the waste cut activities and the uncertainty is calculated by adding the absolute uncertainties in quadrature. The following examples, which use the baseline assumptions, unless otherwise specified, show various permutations of these calculations for a DU. The full method is an extension of these examples to multiple waste cuts and waste streams.

7.1 A single waste cut with a single isotope in a DU

Assume in a DU there is a single waste cut which is part of a waste stream containing a single isotope. This waste cut has a WITS reported activity of 1 Ci as determined by the CBP method. The 1-sigma measurement uncertainty is 5% and is unbiased ($\delta_M = 1$).

As there is no waste stream characterization, the true activity and uncertainty of this waste cut is 1 ± 0.05 Ci.

Because there is only one waste cut in this DU, the DU activity and uncertainty equal the waste cut activity and uncertainty (1 ± 0.05 Ci).

7.2 A single waste cut with multiple isotopes in a DU

Assume in a DU there is a single waste cut which is part of a waste stream containing multiple isotopes. This waste cut has a total reported activity of 1 Ci (M) as measured using the STC method, which is noted in the technical baseline document as being conservative ($\delta_M = 0.9$) and having a 10% uncertainty (U_M). The isotope in this waste stream distribution is characterized as having 50% of the total activity (C) with an uncertainty of 10% (U_C) and is unbiased ($\delta_C = 1$).

Solving Equations (5.1) and (5.2), the WITS isotope activity in this waste cut is 0.5 Ci and the 1-sigma uncertainty is:

$$rel_{unc} = \sqrt{\left({U_M}^2 + {U_C}^2\right)} = \sqrt{\left(0.1^2 + 0.1^2\right)} = 14\% \tag{7.1}$$

The true activity of this waste cut Equation (5.3) is:

$$A_T = (1Ci \cdot 0.9 \pm 10\%) \cdot (0.5 \cdot 1 \pm 10\%) = 0.45Ci \pm 14\% = 0.45 \pm 0.063Ci \tag{7.2}$$

Because there is only a single waste cut in this DU, the DU activity and uncertainty equal the waste cut activity and uncertainty (0.45 ± 0.063 Ci).

7.3 Multiple waste cuts (same stream) in a DU

Assume in a DU there are two waste cuts, each from the same waste stream containing multiple isotopes. Each waste cut has a total reported activity of 1 Ci (M) as measured using the STC method, which is noted in the technical baseline document as being conservative ($\delta_M = 0.9$) and having a 10% uncertainty (U_M). The isotope in this waste stream distribution is characterized as having 50% of the total activity (C) with an uncertainty of 10% (U_C) and is unbiased ($\delta_C = 1$).

Solving Equations (5.1) and (5.2), the WITS reported isotope activity in each waste cut is 0.5 Ci and the 1-sigma uncertainty is:

$$rel_{unc} = \sqrt{\left({U_M}^2 + {U_C}^2\right)} = \sqrt{\left(0.1^2 + 0.1^2\right)} = 14\% \tag{7.3}$$

and the true activity of each waste cut, Equation (5.3), is 0.45 ± 0.063 Ci.

The WITS inventory for this isotope in this DU is 1 Ci and solving Equation (6.3), the true inventory is:

$$I_T \pm U_{I_T} = (0.45 + 0.45) \pm \sqrt{\left(0.063^2 + 0.063^2\right)} = 0.9 \pm 0.089\ Ci \tag{7.4}$$

This DU's bias (δ_B) is $I_T/I_W = 0.9/1 = 0.9$ and it's relative uncertainty is ($U_{I_T}/I_T = 0.089/0.9 = 9.9\%$).

7.4 Multiple waste cuts (different streams) in a DU

Assume in a DU there are two waste cuts, each from a different waste stream and each waste stream contains multiple isotopes. Each waste cut has a total reported activity of 1 Ci ($M_1 = M_2 = 1$ Ci) as measured using the STC method, one of which is noted in the technical baseline document as being conservative ($\delta_{M_1} = 0.9$) and the other is unbiased ($\delta_{M_2} = 1$). Each of these measurements have different measurement uncertainties ($U_{M_1} = 15\%$ and $U_{M_2} = 5\%$). The isotope in each waste stream distribution is characterized as having 50% of the total activity ($C_1 = C_2 = 50\%$). In one waste stream the characterization uncertainty is 5% (U_{C_1}) and is unbiased ($\delta_{C_1} = 1$), in the other, the uncertainty is 20% (U_{C_2}) and is biased ($\delta_{C_2} = 0.9$).

Solving Equations (5.1) and (5.2), the isotope activity in each waste cut is 1 Ci $\cdot$ 50% = 0.5 Ci and the 1-sigma uncertainty of one waste cut is:

$$rel_{unc} = \sqrt{\left(0.15^2 + 0.05^2\right)} = 15.8\% \tag{7.5}$$

The 1-sigma uncertainty of the other waste cut is:

$$rel_{unc} = \sqrt{\left(0.05^2 + 0.20^2\right)} = 20.6\% \tag{7.6}$$

Solving Equation (5.3), the true activity of one waste cut is:

$$A_{T_1} = (0.5 \cdot 0.9 \cdot 1) \pm 15.8\% = 0.45 \pm 0.071 Ci \tag{7.7}$$

and the true activity of the other waste cut is:

$$A_{T_2} = (0.5 \cdot 1 \cdot 0.9) \pm 20.6\% = 0.45 \pm 0.093 Ci \tag{7.8}$$

The WITS inventory for this isotope in this DU is 1 Ci and solving Equation (6.3), the true inventory is:

$$I_T \pm U_{I_T} = (0.45 + 0.45) \pm \sqrt{\left(0.071^2 + 0.093^2\right)} = 0.90 \pm 0.117\ Ci \tag{7.9}$$

This DU's bias (δ_B) is $I_T/I_W = 0.9/1 = 0.9$ and it's relative uncertainty is ($U_{I_T}/I_T = 0.117/0.9 = 13\%$).

8 Usage of Uncertainty and Bias Calculations

The results of the DU activity, uncertainty, and bias calculations can be presented in the form of a normal probability density function (PDF) and cumulative distribution function (CDF) by normalizing I_W to 1 and using that as the mean (μ) and using the relative uncertainty, U_W, as the standard deviation (σ) in the following equations. The standard form for the PDF is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \tag{8.1}$$

and CDF is

$$f(x) = \frac{1}{2}\left[1 + erf\left\{\frac{x-\mu}{\sigma\sqrt{2}}\right\}\right] \tag{8.2}$$

Because these functions describe the uncertainty with respect to the normalized activity, x cannot be less than zero, so when utilizing the CDF equation and $x \leq 0$, the function is set to equal zero. Figures 8.1 and 8.2 show examples of the PDF and CDF for Np-237 in the ILV DU and the average Slit Trench DU.

Performance Assessments (PAs) of the ELLWF are regularly conducted as part of legal compliance for disposing of nuclear waste. The PAs are modeling studies that describe how the location and waste will change over time (up to 10,000 years in the future). These studies ensure the entombment practices and waste movement through the environment will not adversely impact human or environmental health over this time-period. The DU isotopic activity, uncertainty, and bias are used in the closure analysis (the condition of the DUs at the end of waste disposal) portion of the performance assessment. The closure analysis is a Monte-Carlo process, where each run estimates a final inventory of each isotope in a DU from other model parameters and uses a random value (between 0 and 1) combined with the CDF to select a normalized isotopic inventory value. The final inventory values are multiplied by the bias factor (δ_B) to get the first estimate of the true final DU inventory (E_T). Then, each of these estimated true closure inventory values are multiplied by the normalized inventory to create the final true inventory for this isotope for this run.

Using Np-237 in the ILV (Figure 8.1) as an example, if a random value of 0.5 is selected from the CDF, the normalized inventory of 1 is returned (orange line). If a random value of 0.8 is selected, a normalized inventory value of approximately 1.06 is returned. This normalized value is multiplied by E_T to get the final inventory in a DU, for this Monte-Carlo run. For isotopes with small uncertainty, all of the normalized inventory values will be close to 1.0. For an isotope with a large uncertainty, such as the amount of Np-237 in the average Slit Trench (Figure 8.2), this value will vary by a much larger amount (usually between 0 and 3). Here, using a random value of 0.8 returns a normalized value of approximately 2.0.

To speed up the calculations, the CDF_{min} is pre-calculated at $x = 0$. Then, during the Monte-Carlo analysis, any CDF value (y-axis) less than or equal to CDF_{min} returns 0. The CDF_{min} value is approximately 0.16 in Figure 8.2.

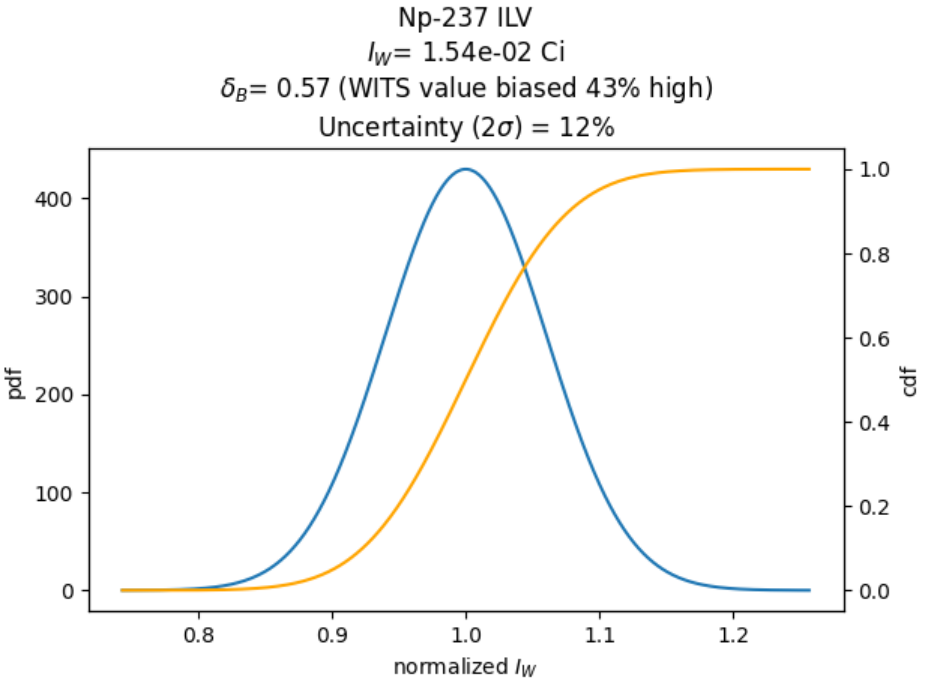


Figure 8.1: PDF (blue) and CDF (orange) of the uncertainty in the WITS reported activity of NP-237 in the ILV (I_W) as of May 1, 2021. δ_B is the bias in that inventory.

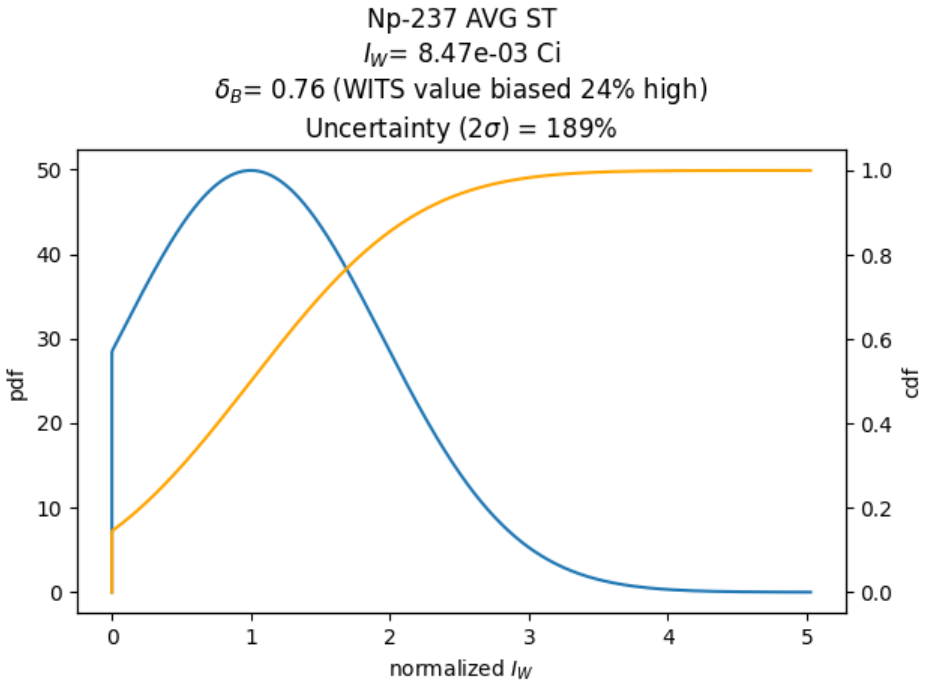


Figure 8.2: PDF (blue) and CDF (orange) of uncertainty in the activity of NP-237 in the average Slit Trench (I_W) as of May 1, 2021. δ_B is the bias in that inventory.

9 Estimating Future Inventory, Uncertainty, and Bias

As waste is added to a DU, the inventory changes over time. Assuming no decay, the inventory should increase until the DU is closed, as shown in Figure 9.1. At each point in time, the inventory will have an uncertainty distribution associated with it. When the DU is first created there is no inventory ($I = I_{open} = 0$) so there is no distribution. Currently ($I = I_{now}$) there is some inventory and an associated uncertainty distribution. At closure ($I = I_{close}$) the uncertainty distribution may be larger, smaller, or the same as current uncertainty distribution.

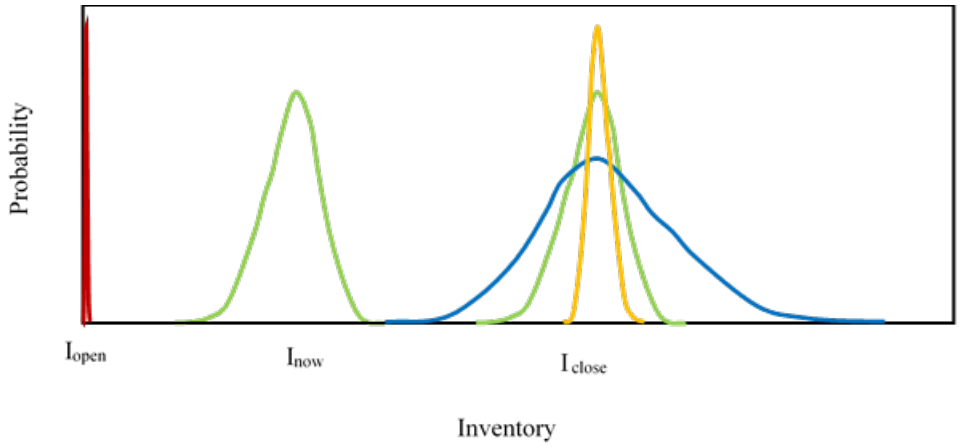


Figure 9.1: Uncertainty in reported inventory in a DU at different times. When DU is opened, there is no inventory so there is no uncertainty (red). As of now, the reported inventory has some uncertainty distribution (green). At closure, the inventory uncertainty may either be: same as now (green), less uncertain than now (yellow), or more uncertain than now (blue).

The isotopic inventory uncertainty distribution and bias values are created as outlined in Section 6. This is sufficient for closed DUs, however for future DUs or DUs that are still open, additional isotopic values may be needed for closure analysis. Additionally, there are special waste forms (SWF) used during the closure analysis that are not currently labeled as such within WITS. To ensure there are uncertainty and bias values for each isotope and SWF in each DU, present and future, the following procedures are used.

First, all DUs that currently contain waste (closed and open DUs, as shown in Table 9.1) are examined. In each of these DUs, the existence of uncertainty and bias files for the isotopes expected to be found in each of the four exposure pathways (GroundWater, Inadvertent Human Intruder, Air, and Radon) is verified (SRNL, 2021). If these isotope files are not found, because they do not exist in WITS in this DU, a placeholder file is created for them. These placeholder files containing the parameters for the bias and CDF are created using the following values:

I_W 1×10^{-12} Ci (1 pCi)

U_W 10%

δ_B 1 (unbiased)

The I_W value is approximately 1 pCi (it is actually a random value centered around 1×10^{-12} with a 10% standard deviation). This approximation is so that if these placeholder files are the only ones available in the current DUs, the average standard deviation between the inventories will not be zero (as calculated for future DUs). This approach is used for all closed and open disposal units, except for ST23. For ST23, all inventory currently disposed (as of May 1, 2021), is considered the SWF of Components-in-Grout (CIG), as denoted by an "A" appended to the isotope symbol.

WITS has not been updated with this information; therefore, this SWF (and other SWFs: such as sealed Naval Casks and Tall Boxes) are addressed during the analysis. Future inventory in ST23 will be treated as using the average trench inventory, as discussed for all future DUs.

Table 9.1: Disposal Unit and Waste Acceptance Status

DU	Status
LAWV	Open
ILV	Open
NR0	Closed
NR1	Open
ET01	Closed
ET02, ET03	Open
ET04-09	Future
ST01-ST05	Closed
ST06-ST09, ST14	Open
ST10, ST11, ST17-ST22, ST24	Future
ST23[a]	Open

[a] See text for ST23 discussion

To estimate future DU values, first the average DU values of currently closed and open DUs are calculated for each type of DU: Slit Trench (ST) and Engineered Trench (ET). These values are calculated in the following manner. For each isotope in a current DU type (Slit Trench (ST) or Engineered Trench (ET)), the WITS or placeholder activity, the estimated true activity, and the absolute uncertainty (of the WITS activity) are multiplied by an averaging factor and divided by the current composition vectors Sum-of-Fraction's (SOF) total, Table 9.2. The SOF total is found in additional modeling inputs to the PA (SRNL, 2021). The arithmetic average of these values is calculated and saved.

Table 9.2: Current DU parameters

DU	Avg Factor	SOF Total
ET01, ST01-ST05	1.00	1.000
ET02	0.95	0.751
ET03	0.95	0.352
ST06	0.95	0.143
ST07	0.95	0.456
ST08	0.95	0.370
ST09	0.95	0.936
ST14	0.95	0.943

Next, for isotopes in future Slit Trenches (except ST10, which uses ST09 values) and ST23 and Engineered Trenches, the average inventory values are used. To estimate the uncertainty and bias, the worst-case values are used. The worst-case is the maximum uncertainty and the minimum bias factor for each isotope in the current DUs of the type. So, for Slit Trenches, the maximum uncertainty and minimum bias factor in ST01-09 and ST14 is selected.

As an example, the results of this analysis for future Engineered Trench DUs for the six prioritized radionuclides are presented in Table 9.3.

Table 9.3: Expected activity, bias, and uncertainty in future Engineered Trench DUs

Isotope	I_{now} (Ci)	δ_B	I_{true} (Ci)	I_{now} Bias	2σ unc.
H-3	1.02	0.55	0.56	45% high	4%
C-14	0.07	0.92	0.06	8% high	5%
Sr-90	52.00	0.57	29.60	43% high	19%
Tc-99	0.05	0.58	0.03	42% high	12%
I-129	6.56E-05	0.58	3.80E-05	42% high	4%
Np-237	0.02	0.58	0.01	42% high	3%

10 Method Implementation

10.1 Code

The calculations and estimations described above have been implemented in two pieces of code: `zmake_dump.py` and `zanalysis.py`. A copy of this code and it's supporting modules is found in Appendix B. This code runs via 32-bit Python v3.8 because access to WITS is through Microsoft Access, which is 32-bit on SRS-machines. We will outline the operation of the code and highlight key functions in this section. The detailed operations of the code can best be understood by reading and running the code found in the appendix. Instructions for how to do this are shown in Appendix A. The only "gotcha" is that there is a date set in the `main(args)` function of `zmake_dump.py` (`xfuture=October 20, 2021`) which prevents the code from running. This was done so future analysts must check the code before using the results. To run the code, simply change this date to some date in the future (for example: October 20, 3021) and then run it.

The `zmake_dump.py` code extracts data for each of the ELLWF DUs from WITS, parses it into various intermediate JSON files, and assigns each isotope in each waste cut either 1) the baseline

uncertainty and bias factor as described in Sections 3 and 4 or 2) the uncertainty and bias factors as estimated from our analysis of the technical baseline documents, Section 10.4. The most interesting portions of this code are found in the `get_isod`, `get_bias`, `get_bias_cuts`, and `get_unc` functions. The `get_isod` function creates the isotope dictionary for each isotope in each waste cut. This function checks for poorly defined packages in WITS (there are currently 20, details of our understanding of their problems are outlined in that function), gets containers with special waste forms of isotopes, and updates the isotope to this form, calls the `get_bias` function, and finally checks that the sum of each isotope's activity in a waste cut in a container equals the total of each isotope in the container. The `get_bias` function is recursive and calls `get_bias_cuts` for each outer container and each inner container and waste cut in that outer container. The `get_bias_cuts` function is the code that steps through each waste cut in each container and calls the `get_unc` function. The `get_unc` function gets the measurement and characterization uncertainties and biases, either the baseline values or the analyzed values.

The `zanalysis.py` code processes the intermediate JSON files created by `zmake_dump.py` and does the summing of measurements and uncertainties and biases as well as creation of files containing the parameters for each isotope. The bulk of this work is done in the `make_hist_pdf_cdf` function. This function calls the recursive `get_act3` function, that calls the `get_cut_info` which sums the cut activities in each container and gets the fractional activity and absolute uncertainties. These fractions are summed and uncertainties are added in quadrature for the total uncertainty in each DU. The `make_dist` function creates a normal distribution based on the total WITS activity and absolute uncertainty and 1000 PDF and CDF values are sampled from this distribution. The bias factor in the DU is estimated from the total "true" inventory and total WITS activity. Files are created of the PDF, CDF, statistics, and a figure for each isotope in each DU. This code also creates, using the `make_avg_out_aux` function, the average values for isotopes in Slit and Engineered Trenches. These average values are used as the values for isotopes in future DUs. In addition to the analysis, this code performs some statistical analysis, creates figures for use in the PA documentation, reformats the files for intake by other software, and saves a compressed copy of the results.

The creation of this software was an iterative process: initial versions developed the computational pipeline to get a set of initial analyses using the baseline estimates. Later revisions focused on using the completed technical baseline analyses of prioritized waste streams. The final revisions incorporated information on how to predict the composition, uncertainties, and biases in future DUs and formatting the results for intake by other software.

10.2 Prioritizing Waste Stream Analysis

As of May 31, 2021, there were 2417 waste streams in WITS. From the analysis conducted herein, the reported ELLWF radionuclide activities are clearly conservative and likely represent reasonable upper bounds for the true activities. However, quantifying the extent of this conservatism and determining the systematic biases and uncertainty within each reported waste cut is a resource-intensive task requiring expert analysis of the waste stream characterization (WSC) and supporting documentation for each waste cut disposed in the ELLWF. Each examination to extract this infor-

mation requires somewhere between one day and one week and, in some cases, the desired values may be undeterminable.

To most efficiently utilize the time available to complete this task, the analysis was prioritized by isotope per DU by using the ranked GroundWater (GW) Sum of Fractions (SRNL, 2021). The `zppp.py` code in Appendix B implements this prioritization by finding the activity of each isotope in each waste stream in each DU and ordering those waste streams from highest to lowest activity. Various options show more of the prioritized isotopes and waste streams. We have used these options to limit the number of waste streams per isotope to those with a Sum of Fraction value greater than 1% and within 1 order-of-magnitude of the maximum value. This has limited the analysis to a median of 10 waste streams per isotope per DU. The result from this prioritization is shown in Appendix C.

There are 282 prioritized waste streams in the 18 ELLWF DUs. Of these, 193 are unique. Some of these waste streams are priorities in multiple DUs. Therefore, analyzing one waste stream gives a rough 1.5-for-1 impact per isotope per analysis. Further, multiple waste streams use the same technical baseline. In addition, multiple isotopes are typically found in a waste stream, so one waste stream analysis, focused on a single isotope in a single DU, improves the estimates of the uncertainty and bias of these other isotopes as well.

An analysis of the activity of waste streams in a DU shows the activities are distributed exponentially. An example of this analysis is shown in Figure 10.1. This figure highlights the fact that two waste streams contain the bulk of total H-3 activity in ET01 (right to left).

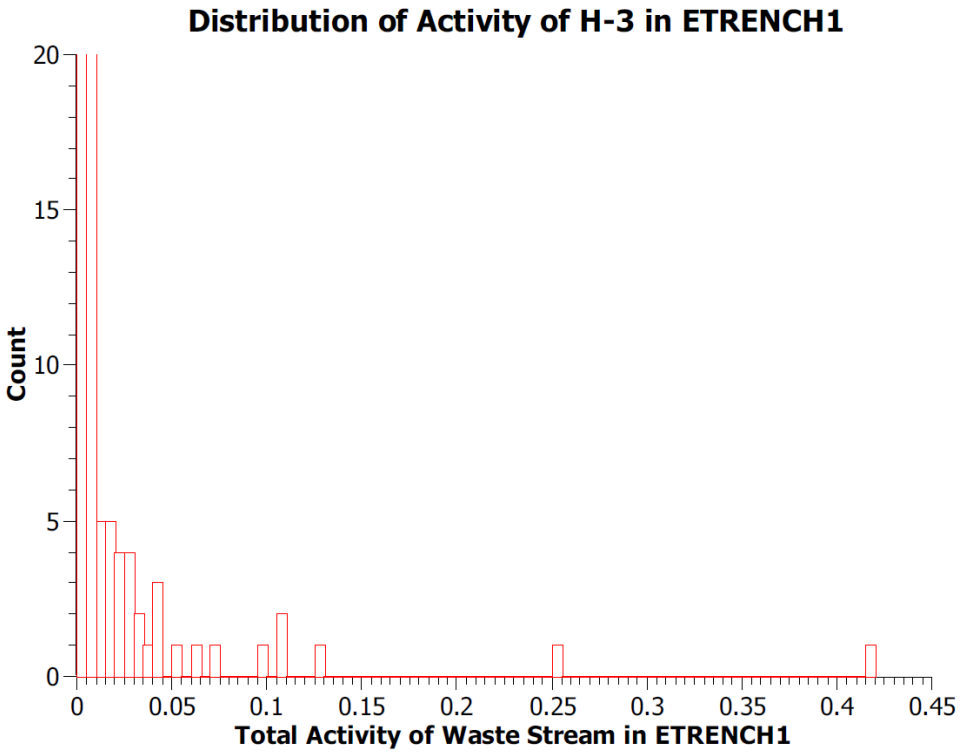


Figure 10.1: Distribution of activity of H-3 per waste stream in ET01

To calculate the DU uncertainty, the absolute uncertainties of each waste cut are added in quadrature, Equation (6.2). Since the waste cut uncertainty is calculated from the waste stream characterization and waste cut measurement, determining the waste stream uncertainty for those waste streams with the largest activity in a DU should have the greatest impact on the overall DU uncertainty. Figure 10.2 shows how this works for eight waste cuts, each from a different waste stream. Each waste cut is assigned an uncertainty (top) and then the total DU activity and uncertainty is calculated (bottom). The left-most column shows the DU calculation with a baseline uncertainty of 10%. Each of the following columns shows the impact of determining the uncertainty in an increasing number of waste streams (assume 5% in each). This demonstrates that determining the uncertainty in the waste streams with the largest activity is the most efficient method to impact the DU uncertainty calculation.

Ci	rel unc	abs unc		Ci	rel unc	abs unc		Ci	rel unc	abs unc		Ci	rel unc	abs unc
10	0.1	1		10	0.05	0.5		10	0.05	0.5		10	0.05	0.5
1	0.1	0.1		1	0.1	0.1		1	0.05	0.05		1	0.05	0.05
0.1	0.1	0.01		0.1	0.1	0.01		0.1	0.1	0.01		0.1	0.05	0.005
0.01	0.1	0.001		0.01	0.1	0.001		0.01	0.1	0.001		0.01	0.1	0.001
0.01	0.1	0.001		0.01	0.1	0.001		0.01	0.1	0.001		0.01	0.1	0.001
0.01	0.1	0.001		0.01	0.1	0.001		0.01	0.1	0.001		0.01	0.1	0.001
0.01	0.1	0.001		0.01	0.1	0.001		0.01	0.1	0.001		0.01	0.1	0.001
0.01	0.1	0.001		0.01	0.1	0.001		0.01	0.1	0.001		0.01	0.1	0.001
Total Ci	abs unc	rel unc		Total Ci	abs unc	rel unc		Total Ci	abs unc	rel unc		Total Ci	abs unc	rel unc
11.15	1.00504	9.0%		11.15	0.510005	4.6%		11.15	0.502598	4.5%		11.15	0.502524	4.5%

Figure 10.2: Example of determining waste stream relative uncertainty on total DU uncertainty. The baseline case is the left-most column.

10.3 Waste Stream Analysis

Of the 2417 waste streams, 168 were analyzed beyond the default baseline values, Table 10.1. The majority of these were the prioritized waste streams identified in Section 10.2 and shown in Appendix C. However, some were analyzed before the prioritization method was implemented. Those are the waste streams that contain the highest activity of one or more of the six PA radioisotopes (H-3, C-14, Sr-90, Tc-99, I-129, and Np-237) in each DU. All of these waste streams had their technical baseline documentation analyzed and waste stream characterization uncertainties and biases were calculated from this information. Seventy-eight of these waste streams had their waste cut measurement uncertainties and biases calculated. This was possible because those values were derived from information in the technical baseline documentation.

WITS stores the waste stream number, the waste stream characterization form (WSCF) number, and the waste cut characterization type. It does not store the technical baseline document number. Therefore, each WSCF must be located and examined by hand to determine the technical baseline document number, and then that document must be located. An additional issue is that the name of the attached document in Lotus notes is not always named the technical baseline document number, but some other, arbitrary, name. For example, in SWE-WSCF-2005-0045 Rev0 there are two technical baselines, one named "122R Characterization Cover" and the other "122-R Waste Stream". The correct technical baseline document number is N-CLC-R-00004 Rev0. Further, in some of the older WSCF documents the technical baseline documents are not able to be found.

The Waste Cut Measurement Method is one of those described in Section 4, and the uncertainty in that characterization is determined either from the defaults or has been determined by analyzing the technical baseline. Typically, the uncertainty in the waste cut measurements is small (10%) compared to the uncertainty in the waste stream composition (100%), and because these relative uncertainties are added in quadrature, Equation (5.2), the large uncertainty dominates. Therefore, when prioritizing the analysis, the focus was on determining the waste stream composition uncertainties.

Table 10.1: The analyzed waste streams.

Waste Stream Number	WSC Form Number	Technical Baseline	Waste Cut Meas
00460-LLW_v.0	SWE-WSCF-1995-0141	750-3403_REV4	['RAD']*
00475-LLW_v.0	SWE-WSCF-1996-0011	WSRC-RP-95-00964_REV0	['RAD']
00564-LLW_v.0	SWE-WSCF-1996-0060	750-3403_REV4	['RAD']*
00687-LLW_v.0	SWE-WSCF-1997-0045	00687-LLW_V0	['RAD']
00687-LLW_v.1	SWEWSCF19970045_rev1	00687-LLW_V1	['RAD']
00689-LLW_v.1	SWEWSCF19970047_rev1	C-ESR-F-00012_REV1	['RAD']
00689-LLW_v.2	SWEWSCF19970047_rev2	C-ESR-F-00012_REV2	['RAD']
100TRITIUM-LLW_v.0	SWEWSCF20050045_rev0	N-CLC-R-00004_REV0	['RAD']*
105L2004-LLW_v.0	SWEWSCF20040067_rev0	Q-CLC-L-00064_REV0	['DTC', 'RAD']
105L2004-LLW_v.3	SWEWSCF20040067_rev5	Q-CLC-L-00160_REV1	['DTC', 'RAD']
105L6034-LLW_v.0	SWEWSCF20050048_rev0	Q-CLC-L-00073_REV0	['RAD']*
105L9005-LLW_v.0	SWEWSCF20070021_rev0	Q-CLC-G-00073_REV0	['RAD']
105PDISAREA-LLW_v.1	SWEWSCF20080004_rev1	Q-CLC-P-00013_REV3	['RAD']
105PDISAREA-LLW_v.2	SWEWSCF20080004_rev2	Q-CLC-P-00013_REV3	['RAD']
105PDISARECERCLA-LLW_v.0	SWEWSCF20100033_rev0	Q-CLC-P-00013_REV3	['RAD']
105PDISPCBBULK-LLW_v.0	SWEWSCF20100053_rev0	Q-CLC-P-00013_REV3	['RAD']
105PPROCESS-LLW_v.1	SWEWSCF20070023_rev1	Q-CLC-P-00013_REV0	['RAD']
105PPROCESS-LLW_v.2	SWEWSCF20070023_rev2	Q-CLC-P-00013_REV3	['RAD']
105PPROPCBBULK-LLW_v.1	SWEWSCF20080008_rev1	Q-CLC-P-00013_REV0	['RAD']
105PPROPCBBULK-LLW_v.2	SWEWSCF20080008_rev2	Q-CLC-P-00013_REV3	['RAD']
105RDISAREA-LLW_v.0	SWEWSCF20090015_rev0	Q-CLC-P-00013_REV3	['RAD']
105RDISAREA-LLW_v.1	SWEWSCF20090015_rev1	Q-CLC-P-00013_REV3	['RAD']
105RDISARECERCLA-LLW_v.0	SWEWSCF20100037_rev0	Q-CLC-P-00013_REV3	['RAD']
105RDISPCBBULK-LLW_v.0	SWEWSCF20100036_rev0	Q-CLC-P-00013_REV3	['RAD']
105RPROCESS-LLW_v.1	SWEWSCF20070026_rev1	Q-CLC-P-00013_REV0	['RAD']
105RPROCESS-LLW_v.2	SWEWSCF20070026_rev2	Q-CLC-P-00013_REV3	['RAD']
105RPROPCBBULK-LLW_v.1	SWEWSCF20090014_rev1	Q-CLC-P-00013_REV0	['RAD']
105RPROPCBBULK-LLW_v.2	[]	Q-CLC-P-00013_REV3	[]
105X1004-LLW_v.4	SWEWSCF20000053_rev4	Q-CLC-K-00143_REV0	['DTC', 'RAD']
105X1004LAR-LLW_v.1	None	Q-CLC-K-00006_REV1	['DTC', 'RAD']
105X2004-LLW_v.4	SWEWSCF20000054_rev4	Q-CLC-K-00143_REV0	['DTC', 'RAD']
105X2004LAR-LLW_v.1	None	Q-CLC-K-00006_REV1	['DTC', 'RAD']
105X3001-LLW_v.0	SWEWSCF19990086_rev0	Q-CLC-C-00029_REV0	['RAD']*
105X6025-LLW_v.0	SWEWSCF20030052_rev0	Q-CLC-K-00097_REV0	['RAD']*
105X6026-LLW_v.0	SWEWSCF20030064_rev1	Q-CLC-L-00047_REV1	['RAD']*
105X6029-LLW_v.0	SWEWSCF20030065_rev1	Q-CLC-L-00052_REV0	['RAD']*
211F813OSF-LLW_v.1	SWEWSCF20060061_rev1	SDD-2006-00484_REV1	['RAD']
211F813SLUDGE-LLW_v.0	SWEWSCF20070001_rev1	SDD-2006-00590_REV1	['RAD']*
211FCERCLA-LLW_v.0	SWEWSCF20070014_rev0	G-CLC-M-00002_REV2	['RAD', 'STC']
211FPIPEOSF-LLW_v.0	SWEWSCF20060064_rev1	SDD-2006-00484_REV3	['RAD']
211FPIPRMCERCLA-LLW_v.0	SWEWSCF20080028_rev0	Q-CLC-F-00181_REV0	['RAD']
211FTANKOSF-LLW_v.0	SWEWSCF20060059_rev0	SDD-2006-00484_REV0	['RAD']
211FTANKOSF-LLW_v.1	SWEWSCF20060059_rev1	SDD-2006-00484_REV1	['RAD']
221F0006-LLW_v.4	SWEWSCF19950097_rev6	C-ESR-F-00012_REV4	['DTC', 'RAD']
221F0006SOIL-LLW_v.1	SWEWSCF19980088_rev1	C-ESR-F-00012_REV1	['RAD']
221FB25HOTSIDE-LLW_v.0	SWEWSCF19990134_rev0	C-ESR-F-00012_REV1	['DTC', 'RAD']
221FHWGCOMB-LLW_v.1	SWEWSCF20020061_rev1	C-ESR-F-00012_REV3	['DTC', 'RAD']
221FHWGCOMB-LLW_v.2	SWEWSCF20020061_rev2	C-ESR-F-00012_REV4	['DTC', 'RAD']
221FOXIDE-LLW_v.0	SWEWSCF20040032_rev0		['CHAR BY PACK']*
221H0021-LLW_v.5	SWEWSCF19970052_rev6	Q-CLC-H-00262_REV3	['RAD', 'STC']
221H0021-LLW_v.6	SWEWSCF19970052_rev7	Q-CLC-H-00262_REV6	['RAD', 'STC']
221H0021-LLW_v.7	SWEWSCF19970052_rev8	Q-CLC-H-00262_REV6	['RAD', 'STC']
221H0021-LLW_v.8	SWEWSCF19970052_rev9	Q-CLC-H-00262_REV6	['RAD', 'STC']
221H0034-LLW_v.1	SWEWSCF20030058_rev1	Q-CLC-H-00189_REV0	['RAD']
221H0036-LLW_v.1	SWEWSCF20030060_rev1	Q-CLC-H-00189_REV0	['RAD']

* Waste cut measurement in technical baseline
** Technical baseline is another document, see unc.xlsx

Continued on next page

Table 10.1 – *Continued from previous page*

Waste Stream Number	WSC Form Number	Technical Baseline	Waste Cut Meas
221H0036-LLW_v.5	SWEWSCF20030060_rev5	Q-CLC-H-00262_REV6	['RAD', 'STC']
221S00017-LLW_v.0	SWEWSCF20030076_rev0	Q-CLC-S-00023_REV1	['DTC', 'RAD']
221S00019-LLW_v.0	SWEWSCF20060063_rev0	Q-CLC-S-00036_REV1	['DTC', 'RAD']
221S00020-LLW_v.0	SWEWSCF20090004_rev0	Q-CLC-S-00055_REV0	['DTC', 'RAD']
221S00021-LLW_v.0	SWEWSCF20090012_rev0	Q-CLC-S-00061_REV0	['DTC', 'RAD']
221S00022-LLW_v.0	SWEWSCF20100050_rev0	Q-CLC-S-00073_REV0	['DTC', 'RAD']
221S00024-LLW_v.0	SWEWSCF20120005_rev0	Q-CLC-S-00091_REV0	['DTC', 'RAD']
221S00024-LLW_v.1	SWEWSCF20120005_rev1	UNC**	['DTC', 'RAD']
230HBGI-LLW_v.0	SWEWSCF20060039_rev0	SDD-2006-00180_REV3	['RAD']*
244H2004-LLW_v.1	SWEWSCF20000051_rev2	Q-CLC-H-00118_REV1	['DTC', 'RAD']
4202DTRITCERCLA-LLW_v.0	SWEWSCF20110001_rev0	Q-CLC-D-00020_REV0	['RAD']*
420DTRITIUM-LLW_v.0	SWEWSCF20050055_rev0	N-CLC-D-00005_REV0	['RAD', 'STC']*
511S01701-LLW_v.0	SWEWSCF20170004_rev0	Q-CLC-S-00119_REV0	['DTC', 'RAD']
511S09001-LLW_v.0	SWEWSCF20090008_rev0	Q-CLC-S-00057_REV0	['DTC', 'RAD']
511S09002-LLW_v.0	SWEWSCF20090016_rev0	Q-CLC-S-00057_REV1	['DTC', 'RAD']
511S1001-LLW_v.0	SWEWSCF20100026_rev0	Q-CLC-S-00071_REV0	['DTC', 'RAD']
5PDPCBBULKCERCLA-LLW_v.0	SWEWSCF20100038_rev1	Q-CLC-P-00013_REV4	['RAD']
5RDPCBBULKCERCLA-LLW_v.0	SWEWSCF20100034_rev1	Q-CLC-P-00013_REV4	['RAD']
745NHXPHS2-LLW_v.0	SWEWSCF20120013_rev0		['CHAR BY PACK']*
770URXCERCLA-LLW_v.0	SWEWSCF20100047_rev0	Q-CLC-U-00005_REV0	['RAD']*
772F022B-LLW_v.4	SWEWSCF19980062_rev6	G-CLC-F-00226_REV1	['DTC', 'RAD']
772F022B-LLW_v.6	SWEWSCF19980062_rev9	G-CLC-F-00226_REV7	['DTC', 'RAD']
772FHLGEN-LLW_v.0	SWEWSCF20140001_rev0	G-CLC-F-00328_REV0	['DTC', 'RAD']
772FHLGEN-LLW_v.1	SWEWSCF20140001_rev1	G-CLC-F-00328_REV1	['DTC', 'RAD']
772FHLGEN-LLW_v.2	SWE-WSCF-2014-0001_rev2	G-CLC-F-00328_REV1	['DTC', 'RAD']
77710A0001-LLW_v.0	SWEWSCF20050001_rev1	G-CLC-A-00132_REV1	['RAD', 'STC']*
BAL001A-LLW_v.0	SWEWSCF20050007_rev0	G-CLC-B-00004_REV0	['STC']*
BOX332005-LLW_v.0	SWEWSCF20050050_rev0	OBU-TRU-2005-00110	['RAD']*
ETF-001-125-LLW_v.0	SWEWSCF19990026_rev1	SWD-ETF-99-003_REV0	['DTC', 'RAD']
ETF-001-125-LLW_v.6	SWEWSCF19990026_rev8	Q-CLC-H-00202_REV3	['RAD']*
ETF-001-125-LLW_v.7	SWEWSCF19990026_rev9	Q-CLC-H-00202_REV5	['RAD']
ETF0032-LLW_v.1	SWEWSCF20090003_rev2	Q-CLC-H-00249_REV1	['RAD']*
FAGW-0001-LLW_v.3	SWEWSCF19970030_rev5	N-CLC-G-00090_REV0	['RAD']*
FCANBLBX06-LLW_v.0	SWEWSCF20060055_rev0		['CHAR BY PACK']*
FCXBOX157-LLW_v.0	SWEWSCF20110040_rev0	Q-CLC-F-00254_REV0	['RAD']*
FDDTRITIUM-LLW_v.0	SWEWSCF20020030_rev0	G-CLC-A-00083_REV0	['RAD']
FHLABSTC-LLW_v.0	SWEWSCF20080003_rev0	G-CLC-F-00267_REV1	['RAD', 'STC']
FHLABSTC-LLW_v.1	SWEWSCF20080003_rev1	G-CLC-F-00267_REV2	['RAD', 'STC']
FHLABSTC-LLW_v.2	SWEWSCF20080003_rev2	G-CLC-F-00267_REV4	['RAD', 'STC']
FHLABSTC-LLW_v.3	SWEWSCF20080003_rev3	G-CLC-F-00267_REV5	['RAD', 'STC']
FHW00001FTFGROU-LLW_v.2	SWEWSCF20140006_rev2	UNC**	['DTC', 'RAD']
FHW00001FTFJCW-LLW_v.1	SWEWSCF20010039_rev1	UNC**	['DTC', 'RAD']
FHW00001FTFJCW-LLW_v.2	SWEWSCF20010039_rev2	UNC**	['DTC', 'RAD']
FHW00001FTFJCW-LLW_v.3	SWEWSCF20010039_rev3	UNC**	['DTC', 'RAD']
FHW00001FTFJCW-LLW_v.6	SWEWSCF20120036_rev2	UNC**	['DTC', 'RAD']
FHW00001FTFSOIL-LLW_v.1	SWEWSCF20010039_rev1	UNC**	['DTC', 'RAD']
FHW00001FTFSOIL-LLW_v.2	SWEWSCF20010039_rev2	UNC**	['DTC', 'RAD']
FHW00001FTFSOIL-LLW_v.3	SWEWSCF20010039_rev3	UNC**	['DTC', 'RAD']
FHW00001FTFSOIL-LLW_v.6	SWEWSCF20120037_rev2	UNC**	['DTC', 'RAD']
FHW00001HTFGROU-LLW_v.2	SWEWSCF20140004_rev3	Q-CLC-H-00525_REV0	['DTC', 'RAD']
FHW00001HTFJCW-LLW_v.1	SWEWSCF20010039_rev1	WSRC-TR-94-00290_REV4	['DTC', 'RAD']
FHW00001HTFJCW-LLW_v.2	SWEWSCF20010039_rev2	Q-CLC-H-00204_REV1	['DTC', 'RAD']
FHW00001HTFJCW-LLW_v.3	SWEWSCF20010039_rev3	Q-CLC-H-00204_REV3	['DTC', 'RAD']
FHW00001HTFJCW-LLW_v.6	SWEWSCF20120030_rev2	Q-CLC-H-00525_REV0	['DTC', 'RAD']
FHW00001HTFSOIL-LLW_v.1	SWEWSCF20010039_rev1	UNC**	['DTC', 'RAD']
FHW00001HTFSOIL-LLW_v.2	SWEWSCF20010039_rev2	UNC**	['DTC', 'RAD']
FHW00001HTFSOIL-LLW_v.3	SWEWSCF20010039_rev3	UNC**	['DTC', 'RAD']

* Waste cut measurement in technical baseline

** Technical baseline is another document, see unc.xlsx

Continued on next page

Table 10.1 – *Continued from previous page*

Waste Stream Number	WSC Form Number	Technical Baseline	Waste Cut Meas
FHW00001HTFSOIL-LLW_v.6	SWEWSCF20120031_rev2	UNC**	['DTC', 'RAD']
FHW00001PCBBULK-LLW_v.2	SWEWSCF20050060_rev2	UNC**	['RAD']
FHW00001PCBBULK-LLW_v.5	SWEWSCF20050060_rev6	UNC**	['DTC', 'RAD']
FHW00003FTF-LLW_v.0	SWEWSCF20030035_rev0	WSRC-TR-2003-00130_REV0	['RAD']
FHW00003FTF-LLW_v.2	SWEWSCF20030035_rev2	Q-CLC-H-00231_REV1	['RAD']
FHW00003HTF-LLW_v.0	SWEWSCF20030035_rev0	UNC**	['RAD']
FHW00003HTF-LLW_v.2	SWEWSCF20030035_rev2	UNC**	['RAD']
FHW00003SALT-LLW_v.0	SWEWSCF20030035_rev0	UNC**	['RAD']
FHW00004PRECIP-LLW_v.1	SWEWSCF20040063_rev1	UNC**	['DTC', 'RAD']
FHW00004PRECIP-LLW_v.4	SWEWSCF20040063_rev4	UNC**	['DTC', 'RAD']
HAGW-0005-LLW_v.1	SWEWSCF19990034_rev1	WM-CTF-00152_REV0	['RAD']*
HBLBLACKBOX-LLW_v.0	SWEWSCF20030073_rev0	WSRC-TR-94-00371_REV0	['RAD']
HBLTRUTOLLW01-LLW_v.0	SWEWSCF20070030_rev1	N-CLC-H-00451_REV1	['RAD']
HIGHLEVELCEL06-LLW_v.1	SWEWSCF20060020_rev2		['CHAR BY PACK']*
HTK00002-LLW_v.1	SWEWSCF20030014_rev1	WSRC-TR-2000-00249_REV3	['RAD']
HTK00002-LLW_v.3	SWEWSCF20030014_rev3	Q-CLC-H-00191_REV1	['RAD']
HTK00002-LLW_v.6	SWEWSCF20030014_rev6	Q-CLC-H-00402_REV3	['RAD']*
HTK000022H42-LLW_v.0	SWEWSCF20010050_rev0	WSRC-TR-2001-00132_REV0	['RAD']
ISO2040FTFJCWFE-LLW_v.2	SWEWSCF20140016_rev2	UNC**	['DTC', 'RAD']
ISO2040HTFJCWFE-LLW_v.2	SWEWSCF20140020_rev2	UNC**	['DTC', 'RAD']
MCU00003-LLW_v.0	SWEWSCF20100052_rev0	Q-CLC-H-00307_REV0	['DTC', 'RAD']
MCU00004-LLW_v.0	SWEWSCF20110045_rev0	Q-CLC-H-00405	['DTC', 'RAD']
NMMPCBBULK1-LLW_v.2	SWEWSCF20050015_rev2	Q-CLC-K-00143_REV0	['DTC', 'RAD']
NMMPCBBULK4-LLW_v.2	SWEWSCF20050016_rev2	Q-CLC-K-00143_REV0	['DTC', 'RAD']
NMMPCBLEACH1-LLW_v.3	SWEWSCF20040023_rev3	Q-CLC-K-00143_REV0	['DTC', 'RAD']
NMMPCBLEACH2-LLW_v.3	SWEWSCF20040024_rev3	Q-CLC-K-00143_REV0	['DTC', 'RAD']
P007OUTFALCERCLA-LLW_v.0	SWEWSCF20100023_rev0	Q-CLC-P-00026_REV0	['RAD']*
PCBBULKSRNL-LLW_v.4	SWEWSCF20090010_rev4	G-CLC-A-00239_REV3	['RAD']*
PCBLABSRNL-LLW_v.1	SWEWSCF20140017_rev1	G-CLC-A-00239_REV3	['RAD']*
PCBLEACHSRNL-LLW_v.3	SWEWSCF20110016_rev3	G-CLC-A-00239_REV3	['RAD']*
SDDSTCROOM-LLW_v.0	SWEWSCF20060043_rev0	G-CLC-M-00002_REV2	['RAD', 'STC']
SEGSLUDGE3-LLW_v.0	SWEWSCF19990032_rev0	G-CLC-C-00004_REV0	['RAD']*
SEGSLUDGE3-LLW_v.1	SWEWSCF19990032_rev1		['CHAR BY PACK']*
SEGSLUDGE5-LLW_v.0	SWEWSCF19990009_rev0	G-CLC-C-00005_REV0	['RAD']*
SFPPCBBULK1-LLW_v.3	SWEWSCF20060048_rev3	Q-CLC-L-00160_REV1	['DTC', 'RAD']
SFPPCBLEACH1-LLW_v.3	SWEWSCF20040022_rev3	Q-CLC-L-00160_REV1	['DTC', 'RAD']
SOL99001CERCLA-LLW_v.0	SWEWSCF20010031_rev0	Q-CLC-E-00022_REV0	['RAD']
SRNLJCW-LLW_v.0	SWEWSCF20060036_rev0	G-CLC-A-00152_REV0	['RAD']
SRNLJCW-LLW_v.1	SWEWSCF20060036_rev1	G-CLC-A-00239_REV0	['RAD']
SRNLJCW-LLW_v.4	SWEWSCF20060036_rev4	G-CLC-A-00239_REV3	['RAD']*
TRITIUMBE-LLW_v.0	SWEWSCF20080027_rev0	Q-CLC-H-00233_REV0	['RAD']
TRITNHPE-LLW_v.0	SWEWSCF20110019_rev0	UNC**	['RAD']*
TRITNHPE-LLW_v.1	SWEWSCF20110019_rev1	UNC**	['RAD']*
TRITNHPE-LLW_v.2	SWEWSCF20110019_rev2	Q-CLC-H-00325_REV0	['RAD']*
TRITNHPE-LLW_v.3	SWEWSCF20110019_rev3	UNC**	['RAD']*
WPT00001LE-LLW_v.0	SWEWSCF20000022_rev0	HLW-STE-2000-00179	['RAD']
WPT00001LE-LLW_v.1	SWEWSCF20000022_rev1	HLW-STE-2000-00179	['RAD']
WPT0000251-LLW_v.0	SWEWSCF19990123_rev3	WSRC-TR-2000-00115_REV0	['RAD']
WPT0000251-LLW_v.1	SWEWSCF19990123_rev4	UNC**	['RAD']
ZSS0043-LLW_v.0	SWEWSCF20040031_rev0	WSP-SSF-2004-00025_REV1	['RAD']

* Waste cut measurement in technical baseline

** Technical baseline is another document, see unc.xlsx

10.4 Technical Baseline Analysis

10.4.1 Overview

The analysis of the technical baseline documents was an iterative process that rapidly converged to a standard method. A general description of the error analysis and uncertainty propagation process is presented herein. Also included are an example calculation and notes on where the uncertainty estimation methods deviated from the general process. However, please note that since each technical baseline document describes a unique waste stream, each analysis was customized to accommodate that particular waste stream.

To follow the analysis of each technical baseline, it is best to examine the waste stream and technical baseline Excel files directly. These are located on the SRNL HPC storage at:

`//godzilla-01/hpc/project/projwork82/PA2022/Personnel/Whiteside/earea_unc/db/uncert_analysis_docs/`

To follow the analysis, first choose the waste stream directory and then examine the unc.xlsx file to determine which technical baseline document and technical baseline table applies to this waste stream. For example, the waste stream characterization "SOL99001CERCLA-LLW_v.0" is from Table 5 of the technical baseline "Q-CLC-E-00022_rev0". Next, open the technical baseline PDF and Excel file. Finally, work backwards using the "Trace Precedents" Formula Auditing tool. Using this tool, it is possible to trace through the Excel file and compare the numbers to those in the technical baseline document and ensure the data has been input correctly and the uncertainty propagation techniques described in Section 10.4.2 have been followed correctly.

For all waste streams, the PA radionuclides (H-3, C-14, Sr-90, Tc-99, I-129, U-234, U-235, and Np-237) are specifically reported unless able to be excluded as outlined in SRS Manual 1S Section 5.9.2 (SRS, 2021). When a PA radionuclide cannot be excluded, those that are less than the lower limit of detection (LLD) are assigned a value of 0.1 times the LLD, excepting I-129. I-129 is either reported as the Cs-137 activity times 9.00E-7 (two times the I-129/Cs-137 ratio - fission product ratio) or 0.1 times the LLD of I-129.

We have attempted to be consistent in our estimation of uncertainty by determining how the above rules were applied during the technical baseline analysis. If an isotope is reported at 0.1 times the LLD, we have used an uncertainty of 500% with a bias factor of 0.9. This uncertainty is likely conservative and should probably be 1000%: e.g. $1000\% \times 0.1$ LLD = 1 LLD. Reporting a value for something that is not measured is conservative, so a bias factor of 0.9 is a reasonable baseline factor. If the I-129 is reported as a factor of Cs-137 activity, we use the uncertainty of Cs-137 and a bias factor of 0.5 (because the factor is double the fission product ratio).

10.4.2 Uncertainty Analysis

Analyzing the technical baseline to reproduce the waste stream characterization of relative isotopic distribution allows the determination of the uncertainty and bias. This is done by following the procedures for uncertainty propagation set out in (Taylor, 1997).

This work uses three methods to determine $q \pm \delta q$ by propagating uncertainty through these calculations. For sums:

$$q = x + \cdots + z - (u + \cdots + w) \tag{10.1}$$

$$\delta q = \sqrt{(\delta x)^2 + \cdots + (\delta z)^2 + (\delta u)^2 + \cdots + (\delta w)^2} \tag{10.2}$$

For products:

$$q = \frac{x \times \cdots \times z}{u \times \cdots \times w} \tag{10.3}$$

$$\delta q = \sqrt{\left(\frac{\delta x}{x}\right)^2 + \cdots + \left(\frac{\delta z}{z}\right)^2 + \left(\frac{\delta u}{u}\right)^2 + \cdots + \left(\frac{\delta w}{w}\right)^2} \tag{10.4}$$

Finally, the generic case, where q is a function of several variables $x, \cdots, z$ then uncertainty in q is:

$$\delta q = \sqrt{\left(\frac{\partial q}{\partial x}\delta x\right)^2 + \cdots + \left(\frac{\partial q}{\partial z}\delta z\right)^2} \tag{10.5}$$

The uncertainty analysis is carried out by reproducing the results reported in the technical baseline document. This is typically done by creating an Excel file with each table in the document as it's own worksheet. Usually there are multiple tables in a technical baseline document that start with the results of analytical measurements and then follow a series of calculations based on the requirements in SRS Manual 1S to eliminate or report the waste stream composition. The most efficient way to complete the waste stream analysis is to enter the analytical measurement data and reported uncertainty and any bias and then propagate the results to the next table, following the calculation method outlined in the technical baseline document. When this is done correctly, the final waste stream characterization will match the values reported in the technical baseline document along with having the final uncertainty and bias of that characterization.

An additional complication is that many technical baseline documents contain results from other technical baseline documents, SRS reports, or other sources . To complete the uncertainty analysis in the top-level technical baseline, the uncertainty and bias must first be calculated in these supporting documents and propagated into the technical baseline document.

Occasionally errors are found in the calculation of the values in these documents or the source documents are unavailable. In these cases, best estimates of the values reported are made. Usually these are made using the baseline assumptions outlined in Sections 3 and 4.

Because each technical baseline document is unique, creating each technical baseline analysis Excel file is a one-off processes that must be repeated for each waste stream.

The waste stream characterization reports the relative isotopic quantity in each waste stream. We refer to that value as the Curie Fraction and it is determined through a combination of the three uncertainty propagation methods outlined above.

For example, two sets of analytical measurements, reporting multiple isotopes in Ci/g with relative uncertainties, are added together. In this case the relative uncertainties must be converted to

absolute uncertainties before being added in quadrature, Equation (10.2). Then the final Curie Fraction is found by normalizing the individual isotopes concentrations to fraction of the whole. To determine the relative uncertainty in this Curie Fraction, the uncertainty of each isotope is calculated using Equation (10.5).

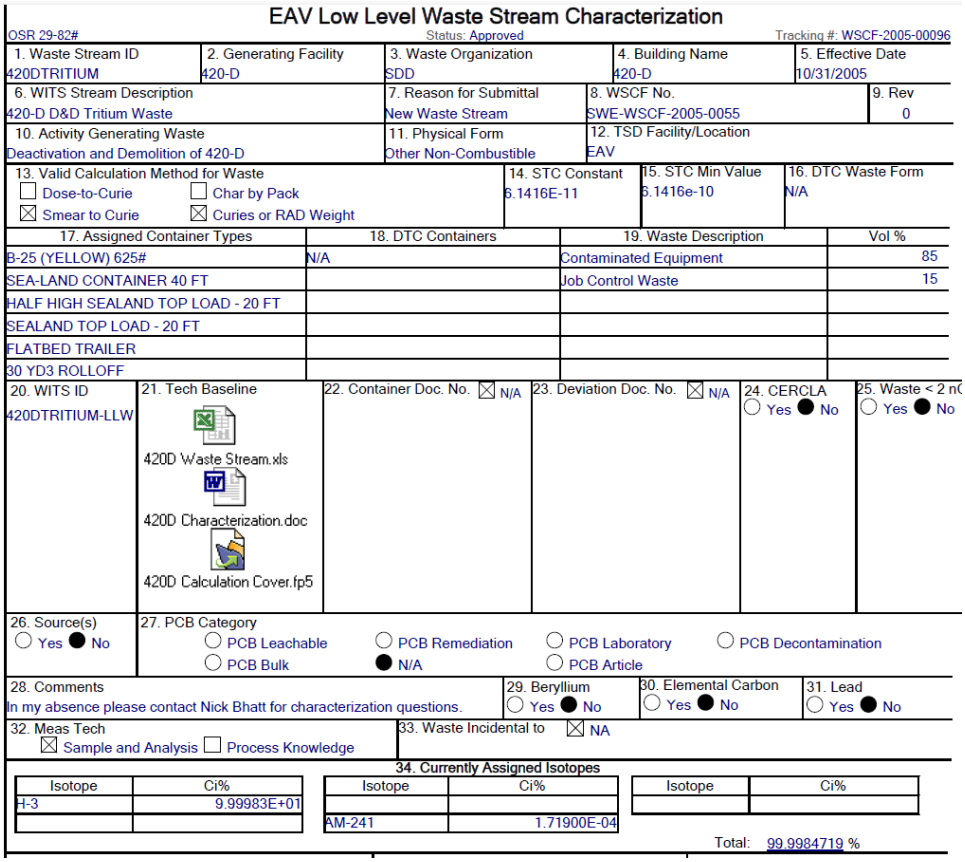
To help with this calculation, we developed a tool, `zpartials.py` in Appendix B, to calculate the uncertainty in the Curie Fraction.

The bias in the waste stream characterization is a straight-forward process that is determined by propagating the bias using the same operations as the isotopic activities.

10.4.3 Example Technical Baseline Analysis

The following analysis of a waste stream and the application of that analysis to determining the uncertainty and bias in a specific waste cut shows the process and some of the challenges encountered.

Waste Stream Characterization Analysis The analyzed waste stream is 420DTRITIUM-LLW v.0. From WITS, the WSCF number is SWEWSCF20050055, Revision 0. Examining the WSCF, Figure 10.3, the technical baseline document is called "420D Characterization.doc", and examining the "420D Calculation Cover.fp5", identifies the true document number as N-CLC-D-00005 Rev. 0 (SRS, 2006a). The WSCF also shows this waste stream has two isotopes and their relative distribution.



EAV Low Level Waste Stream Characterization

OSR 29-82# | Status: Approved | Tracking #: WSCF-2005-00096

1. Waste Stream ID	2. Generating Facility	3. Waste Organization	4. Building Name	5. Effective Date
420DTRITIUM	420-D	SDD	420-D	10/31/2005

6. WITS Stream Description	7. Reason for Submittal	8. WSCF No.	9. Rev
420-D D&D Tritium Waste	New Waste Stream	SWE-WSCF-2005-0055	0

10. Activity Generating Waste	11. Physical Form	12. TSD Facility/Location
Deactivation and Demolition of 420-D	Other Non-Combustible	EAV

13. Valid Calculation Method for Waste	14. STC Constant	15. STC Min Value	16. DTC Waste Form
☐ Dose-to-Curie ☐ Char by Pack ☒ Smear to Curie ☒ Curies or RAD Weight	6.1416E-11	6.1416e-10	N/A

17. Assigned Container Types	18. DTC Containers	19. Waste Description	Vol %
B-25 (YELLOW) 625#	N/A	Contaminated Equipment	85
SEA-LAND CONTAINER 40 FT		Job Control Waste	15
HALF HIGH SEALAND TOP LOAD - 20 FT			
SEALAND TOP LOAD - 20 FT			
FLATBED TRAILER			
30 YD3 ROLLOFF			

20. WITS ID	21. Tech Baseline	22. Container Doc. No. ☒ N/A	23. Deviation Doc. No. ☒ N/A	24. CERCLA	25. Waste < 2 n0
420DTRITIUM-LLW	420D Waste Stream.xls 420D Characterization.doc 420D Calculation Cover.fp5			○ Yes ● No	○ Yes ● No

26. Source(s)	27. PCB Category
○ Yes ● No	○ PCB Leachable ○ PCB Remediation ○ PCB Laboratory ○ PCB Decontamination ○ PCB Bulk ● N/A ○ PCB Article

28. Comments	29. Beryllium	30. Elemental Carbon	31. Lead
In my absence please contact Nick Bhatt for characterization questions.	○ Yes ● No	○ Yes ● No	○ Yes ● No

32. Meas Tech	33. Waste Incidental to
☒ Sample and Analysis ☐ Process Knowledge	☒ NA

34. Currently Assigned Isotopes

Isotope	Ci%	Isotope	Ci%	Isotope	Ci%
H-3	9.99983E+01				
		AM-241	1.71900E-04		

Total: 99.9984719 %

Figure 10.3: Box 34 shows the distribution of isotopic activity into H-3 and Am-241 for waste stream 420DTRITUM-LLW_v.0.

Attachment 1 of the technical baseline document, Figure 10.4, contains the results of the sample analysis. The final column (AVG) contains the arithmetic average concentration (uCi/g for H-3 and dpm/g for the remaining isotopes). Attachment 2 of the technical baseline document, Figure 10.5, shows the distribution of activity, reported in Attachment 1, to various isotopes following the 1S procedure (SRS, 2021) and then the elimination of many of those isotopes due to process knowledge, resulting in the final "Normalized Activity (Ci)" isotopic distribution. This distribution contains only H-3 (μ/Ci) and Am-241 (dpm/g) with their activities converted to Ci (assuming 1 g of sample).

	98B	98A	EP-97	18B	EP-57	18A	34B	34A	
Sample #	300216892	300216893	300216886	300216890	300216887	300216891	300216888	300216889	AVG
Alpha (dpm/g)	<1.35E+02	<1.18E+02	5.53E+00	2.58E+01	6.08E+01	1.20E+01	<5.76E-01	8.15E-01	2.10E+01
Beta (dpm/g)	<1.86E+02	<1.59E+02	<2.59E+01	1.62E+02	5.79E+02	3.40E+01	<9.05E+00	<1.46E+01	2.58E+02
Tritium (uCi/g)	6.90E-03	2.81E-02	1.41E-03	2.07E+01	4.33E+02	1.31E+01	3.07E+00	2.84E-04	5.87E+01
Gamma PHA (dpm/g)									
Ra-226			4.93E+01						4.93E+01
Co-60				5.51E+02	7.82E+02	3.20E+01	4.61E+00	8.02E+00	2.76E+02
Cs-137				1.35E+02	7.35E+01				1.04E+02
Am-241				1.01E+02	3.47E+02				2.24E+02
Eu-152					2.71E+01				2.71E+01
Eu-154					2.98E+02				2.98E+02

Figure 10.4: Attachment 1 of N-CLC-D-00005 Rev. 0

	A	B	C	D	E	F	G	H	I	J	K
1	Radionuclide	Radionuclide Designation	Result (dpm/g)	Result (uCi/g)	Reportable Result (dpm/g)	Activity (uCi)/gm	MALLD uCi/g	>2 Orders of magnitude below MALLD?	Percent Activity	Normalized Activity (Ci)	Normalized Percent Activity
2	Tritium	PA	NA	5.870E+01	1.303E+08	5.870E+01	N/A	N/A	1.000E+02	5.870E-05	9.999983E+01
3	Cs-137		1.04E+02		1.040E+02	4.685E-05	1.000E-05	NO	7.981E-05		
4	Ba-137m	DAUGHTER(Cs-137)	9.84E+01		9.838E+01	4.432E-05	1.000E-05	NO	7.550E-05		
5	U-238		4.93E+01		4.930E+01	2.221E-05	1.000E-06	NO	3.783E-05		
6	Am-241	TRU	2.24E+02		2.240E+02	1.009E-04	1.000E-05	NO	1.719E-04	1.009E-10	1.7189E-04
7	Co-60		2.76E+02		2.760E+02	1.243E-04	1.000E-04	NO	2.118E-04		
8	Ra-226	DAUGHTER(U-238)	4.93E+01		4.930E+01	2.221E-05	1.000E-06	NO	3.783E-05		
9	Th-234	DAUGHTER(U-238)	4.93E+01		4.930E+01	2.221E-05	1.000E-06	NO	3.783E-05		
10	Pa-234-m	DAUGHTER(U-238)	4.93E+01		4.930E+01	2.221E-05	1.000E-06	NO	3.783E-05		
11	Pa-234	DAUGHTER(U-238)	4.93E+01		4.930E+01	2.221E-05	1.000E-06	NO	3.783E-05		
12	U-234	DAUGHTER(U-238)	4.93E+01		4.930E+01	2.221E-05	1.000E-06	NO	3.783E-05		
13	Th-230	DAUGHTER(U-238)	4.93E+01		4.930E+01	2.221E-05	1.000E-06	NO	3.783E-05		
14	Rn-222	DAUGHTER(U-238)	4.93E+01		4.930E+01	2.221E-05	1.000E-06	NO	3.783E-05		
15	Po-218	DAUGHTER(U-238)	4.93E+01		4.930E+01	2.221E-05	1.000E-06	NO	3.783E-05		
16	Pb-214	DAUGHTER(U-238)	4.93E+01		4.930E+01	2.221E-05	1.000E-06	NO	3.783E-05		
17	Pb-210	DAUGHTER(U-238)	4.93E+01		4.930E+01	2.221E-05	1.000E-06	NO	3.783E-05		
18	Bi-210	DAUGHTER(U-238)	4.93E+01		4.930E+01	2.221E-05	1.000E-06	NO	3.783E-05		
19	Po-210	DAUGHTER(U-238)	4.93E+01		4.930E+01	2.221E-05	1.000E-06	NO	3.783E-05		
20	Eu-152		2.71E+01		2.710E+01	1.221E-05			2.080E-05		
21	Eu-154		2.98E+02		2.980E+02	1.342E-04			2.287E-04		
22	Total					5.870E+01			1.000E+02	5.870E-05	1.000E+02

Figure 10.5: Attachment 2 of N-CLC-D-00005 Rev. 0

The uncertainty in H-3 and Am-241 is the sample standard deviation (Excel function stdev.s()) of the samples analyzed and converted into Ci. This distribution is summarized in Table 10.2.

Table 10.2: Average isotope activity and uncertainty normalized to Ci

Iso	Act	Abs Unc	Norm Act (Ci)	Norm Unc (Ci)
H-3 (μCi/g)	5.87E+01	1.51E+02	5.87E-05	1.51E-04
Am-241 (dpm/g)	2.24E+02	1.74E+02	1.01E-10	7.88E-11

The Curie Fraction (relative isotopic distribution) is calculated by dividing the Normalized Activity of each isotope by the total Normalized Activity (and then multiplying by 100 to get the "Normalized Percent Activity"). The distribution is made up of two factors (H-3 and Am-241), so the uncertainty in each factor is calculated using Equation (10.5) as shown in Figure 10.6. H-3 makes up such a large fraction of the total distribution, the uncertainty in this portion of the distribution is small (4.625E-04%) and Am-241 is such as small fraction of the total distribution, the uncertainty is large

(269.53%).

$$Q_{^3H} = \frac{^3H}{^3H + ^{241}Am} = 0.999998$$

$$\delta Q_{^3H} = \sqrt{\left(\frac{\partial Q_{^3H}}{\partial\, ^3H}\delta\, ^3H\right)^2 + \left(\frac{\partial Q_{^3H}}{\partial^{241}Am}\delta^{241}Am\right)^2}$$

$$\delta Q_{^3H} = \sqrt{\left(\frac{^{241}Am}{(^3H + ^{241}Am)^2}\delta\, ^3H\right)^2 + \left(\frac{-\,^3H}{(^3H + ^{241}Am)^2}\delta^{241}Am\right)^2}$$

$$\delta Q_{^3H} = \sqrt{\left(\frac{1.01\text{E-}10}{(5.87\text{E-}5 + 1.01\text{E-}10)^2}1.51\text{E-}4\right)^2 + \left(\frac{-5.87\text{E-}5}{(5.87\text{E-}5 + 1.01\text{E-}10)^2}7.88\text{E-}11\right)^2}$$

$$\delta Q_{^3H} = \sqrt{4.44\text{E-}06^2 + \text{-}1.34\text{E-}06^2} = 4.625\text{E-}06$$

$$\frac{\delta Q_{^3H}}{Q_{^3H}} = \frac{4.625\text{E-}06}{0.999998} = 4.625\text{E-}06 = 4.625\text{E-}04\% \text{ of H-3 distribution}$$

$$Q_{^{241}\text{Am}} = \frac{^{241}\text{Am}}{^3H + ^{241}Am} = 1.72\text{E-}06$$

$$\delta Q_{^{241}\text{Am}} = \sqrt{\left(\frac{\partial Q_{^{241}\text{Am}}}{\partial\, ^3H}\delta\, ^3H\right)^2 + \left(\frac{\partial Q_{^{241}\text{Am}}}{\partial^{241}Am}\delta^{241}Am\right)^2}$$

$$\delta Q_{^{241}\text{Am}} = \sqrt{\left(\frac{-^{241}Am}{(^3H + ^{241}Am)^2}\delta\, ^3H\right)^2 + \left(\frac{^3H}{(^3H + ^{241}Am)^2}\delta^{241}Am\right)^2}$$

$$\delta Q_{^{241}\text{Am}} = \sqrt{\left(\frac{\text{-}1.01\text{E-}10}{(5.87\text{E-}05 + 1.01\text{x}10^{-10})^2}1.51\text{E-}4\right)^2 + \left(\frac{5.87\text{E-}05}{(5.87\text{E-}05 + 1.01\text{E-}10)^2}7.88\text{E-}11\right)^2}$$

$$\delta Q_{^{241}\text{Am}} = \sqrt{1.34\text{E-}06^2 + \text{-}4.44\text{E-}06^2} = 4.625\text{E-}06$$

$$\frac{\delta Q_{^{241}\text{Am}}}{Q_{^{241}\text{Am}}} = \frac{4.625\text{E-}06}{1.721\text{E-}06} = 2.6953 = 269.53\% \text{ of Am-241 distribution}$$

Figure 10.6: Calculation of Distribution Uncertainty in H-3 and Am-241

This information, summarized in Table 10.3, is all that is needed for the C and U_C factors in Equations (5.1) and (5.2). For this example we are setting the bias factor equal to 1. However, in the technical baseline, they use too few significant figures for the average H-3 concentration (use 58.7 versus the correct value of 58.74). This slightly biases the distribution of H-3 too low and Am-241 too high.

Table 10.3: Relative isotope distribution and uncertainty

Iso	C	U_C
H-3	9.9999E-01	4.625E-04%
Am-241	1.7178E-06	269.53%

Waste Cut Measurement Analysis

Once the waste stream analysis is complete, the next step is to determine the activity of each waste cut. How these measurements are to be made should always be described in the technical baseline documents. The results of these measurements may not always be in the technical baseline documents. In this example, the technical baseline documentation describes that these waste cuts should be characterized by the RAD and STC methods.

The STC method uses the smear values converted to Curies. This is usually done for an instrument or other object that can easily be smeared. For rubble, soil or other loose objects, the RAD method is used. In this method, a series of smears are obtained and an STC per pound conversion factor is created following (SRS, 2021) Waste Acceptance Criteria 2.02. This (RAD method) is the technique used in this waste stream.

The activity of a waste cut using this method is calculated by the equation:

$$A = S \times W \times STCC \tag{10.6}$$

where A is the activity in Ci, S is the smear value in dpm / 100 cm^2, W is the waste cut weight in lbs, and $STCC$ is the STC conversion factor, defined in Equation (10.7).

The STC conversion factor in this waste stream is expressed as:

$$STCC = \frac{T \times P \times K}{D_a \times F \times E} \tag{10.7}$$

where the following values came from (SRS, 2006a):

T stands for one-sided contamination ($T = 1$)

P fraction of smear data present on the waste ($P = 1$)

K 2.0472E-09 mg-Ci-100cm2/lb-dpm-cm2 (SRS, 2021)

D_a areal density of waste, where $D_a = 500$ mg/cm2 (SRS, 2006b)

F fraction of activity distribution present on waste, where $F = 1$, conservatively for tritium

E smear efficiency, where $E = 0.1$ for surveying

This results in a STC conversion factor of 4.0944E-11 Ci 100 cm^2 / lb dpm.

The STC conversion factor has uncertainty associated with all these assumptions and measurements. Without the documentation to determine how these values are set, estimating the uncertainty in this constant is nearly impossible. A reasonable uncertainty assumption is 20% based on the number of significant figures (one) reported for D_a.

The weight uncertainty is estimated to be approximately 1% (Gray, 2012; NIST, 2019).

When measuring a waste cut using this method, multiple smears are typically taken of the object. The uncertainty is the standard deviation of the multiple smear values. In this waste stream, multiple smears were made of several B-25s. The uncertainty in these values varies between 0% and 282% and has an average uncertainty of 118%, as shown in Figure 10.7.

	A	B	C	D	E	F	G	H
	B-25#	SD00004142	SD00004156	SD00004150	SD00004162	SD00004161	SD00004160	SD00004155
	Survey #	194&195	194&195	97&94	Logsheet	98	103&99	Waste
	05-400D-							Datasheet
	TRITIUM	150	150	146053	1575	150	150	710
	DPM/100CM2	150	150	9381		150	25651	194677
		150	150	84133		150	74656	716618
		150	150	60488		19852	2189	6445
		150	150	66551		30486	4189	690
		150	150	150		35079	15000	486961
		150	150	1000		31505	6182	234
		150	150	1000		28296	4007	103009
				1000		36121	3639	7181
				1000		8965	1322	150
				1000		9482	229	6640
				1000		5982	150	246
				1000			150	1559
				1000			150	437564
				3400			150	387765
				1000			150	
				1000			150	
				1000			150	
							150	
							1000	
							1526	
							6247	
							150	
							150	
							4543	
							150	
							150	
							205427	
							150	
							78864	
							39186	
							150	
							150	
							150	
							150	
	Average	150	150	21175	1575	17185	13616	156697
		0	0	40971.40649		14504.8551	38378.19191	284952.8858
		0%	0%	193%		84%	282%	150%

Figure 10.7: B-25 Smear Uncertainty: Sample Standard Deviation of Multiple Smears Divided by the Average Activity

For individual waste cuts where the smear value uncertainty is known, that value should be used. For waste cuts where the smear value and uncertainty are unknown, the 118% is a reasonable choice.

Calculating the waste cut measurement uncertainty is completed by combining these uncertainties in quadrature, per Equation (10.4), and results in:

$$U_M = \sqrt{1.18^2 + .01^2 + .02^2} \tag{10.8}$$

for a measurement uncertainty U_M of 120%.

Waste Cut Activity, Uncertainty, and Bias Calculation

When a waste cut is created for this waste stream, assigning the appropriate amount of activity to each isotope uses the total measured activity of the waste cut and the isotopic fraction of H-3 and Am-241.

For example, Waste Package SD00003950 is a top-loaded 20-foot SeaLand container which contains one waste cut (Id 1). It has a total activity of 737.990 Ci. The smear values for this package are not available, so the 120% measurement uncertainty value will be used. Since these factors are unbiased, the isotopic distribution is calculated using Equations (5.1) and (5.2) and the results are shown in Table 10.4

Table 10.4: Isotope activity, uncertainty

Iso	M (Ci)	U_M	C (Ci frxn)	U_C	A (Ci) $\pm U_A$
H-3	737.990	120%	9.9999E-01	4.625E-04%	737.988 ±120%
Am-241	737.990	120%	1.7178E-06	269.53%	1.269E-3 ±295%

Additional Challenges

Each technical baseline document is written by individual waste generators, and while they all follow the same format outlined in (SRS, 2021), each has it's own style and presentation of the information. In addition, the information may be incorrectly transcribed from the technical baseline into the WSCF and into WITS. For example, in this waste stream the STC Constant is calculated in the technical baseline as 4.0944E-11 and the minimum STC value is 6.1416E-11 (this assumes a 150 dpm / 100 cm^2 smear and a 1 lb waste weight) however, on the WSCF Figure 10.3 the STC minimum value is used as the STC Constant, and the STC Min Value is set equal to 10 times this value. If the larger STC Constant is used, this could bias the reported activity high by a factor of 1.5 (bias factor of 0.66).

Another challenge is interpreting how the activity of the waste cuts is assigned. For example, the technical baseline states that all 30 yd^3 rolloff pans will use a waste weight of 30,000 lbs and a smear value of 10,000 dpm / 100 cm^2. How this information was derived is not clear. It also describes how waste in Sealand containers and individual pieces of large equipment are characterized

using analytical measurements and a "per volume" measurement. These values are challenging to interpret, so using the 120% measurement uncertainty derived above seems reasonable.

These are some of the challenges in calculating uncertainties and biases for individual waste cuts and waste streams.

11 Other Conservatisms

While this study has used the quantified bias in the DTC measurements, as shown by (Cook et al., 2011; INEL, 1995; Nevada Test Site, 2006; Verst, 2021), it is known there are additional conservatisms in the waste characterization and reporting methods, but these are more difficult to consistently quantify.

A general description of common conservatisms employed by waste generators and performance assessment modelers is as follows:

- Currently the inventory is not decay-corrected until after DU closure. In future work, this bias can be reduced by decay-correcting the inventory, specifically for isotopes with no unstable daughters (H-3, C-14, I-129), based on the disposal date or waste cut measurement date.
- Generators may assume higher material density than what is present to account for metal present in the waste in the DTC method.
- For large, contaminated equipment, the waste volume is the estimated envelope volume as described in Section 5.27 of (SRS, 2021).
- If a PA radionuclide cannot be excluded from a waste stream, it is assigned a value of 10% of the lower limit of detection. An exception is I-129, which is assigned a value of either 10% of the lower limit of detection or two times the I-129/Cs-137 ratio, as described in Section 5.9.3 of (SRS, 2021).
- The factors used in the STC calculations, Sections 5.10.6 and 5.10.7 of (SRS, 2021).

Understanding that these conservatisms are present provides further justification that the inventory in WITS is conservative and the approximated uncertainty covers the true estimate with high confidence.

12 Recommendations

The following are recommendations for improving the estimation of uncertainty and bias values in the ELLWF inventory.

- Additional analysis of the highest priority waste streams for the isotopes of most-concern in each DU, should be continued. This analysis should continue until the uncertainty and bias of each isotope in a DU does not effectively change upon analysis of additional waste streams.

- Future waste streams should have the technical baseline document and revision number included in WITS.
- WITS should be updated to include all the special-waste forms (SWF) used within the 2022 PA.
- WITS should be updated to include the uncertainty and bias analysis conducted in this work.
- The properties of multiple special-waste forms should be defined. For example, in ST23, there is currently both the SWF of U-235-D (depleted U-235) and the generic waste form U-235. Both of these are currently found within CIG and will become U-235-A. What this means for future Performance Assessments should be defined.
- The low-level waste disposal procedure should be modified so waste generators include the uncertainty and bias in future and revalidated waste-streams and ensure these values are captured within WITS and the technical baseline documentation.
- The code that operates (create, remove, update, and delete information) with the WITS database should have a complete test-suite, it should be audited by an independent team to ensure inputs are sanitized and outputs are as expected. A few examples where improvements may be necessary:
  - Some of the waste-cut IDs appear to be truncated or non-sanitized. Such as in Waste Container CBALX3638, the cut id is "1,5,7,8,9,". This appears to be truncated. Perhaps the waste-cut IDs are limited to 10 characters or the character after the last comma was an escape character, such as % or *, causing the insert code to abort.
  - It appears that RD00010012 (among others) uses the wrong waste_stream_version to get PCT_DIST_CI_NORM used to calculate the activity. U-238 is not in the waste stream version reportedly used, so probably it is using either Version 0 or Version 1. NOTE: This particular issue was resolved on 9/14/2021, which was after this document was originally drafted (9/10/2021).
- Identify and correct any discrepancies between entries in WITS and the analysis code (this work) that extracts those entries from WITS.

13 Conclusions

Retroactively determining the uncertainty in generator-reported radionuclide activities is a challenging task that requires a great deal of expert analytical time. Using the process outlined in this report, the uncertainty and bias have been calculated for some of the prioritized waste cuts and waste streams and estimated for the remaining waste cuts and waste streams. The values produced by this method can be further refined by analyzing additional, prioritized containers and waste streams. These results have been summed to estimate the uncertainty and bias for each isotope in each closed, open, and future DU in the ELLWF. These estimates are used by other model codes to predict the performance of the ELLWF at closure and up to 10,000 years in the future.

One advantage of having a comprehensive framework is that the law-of-large-numbers comes into play: each waste cut is not particularly important. If the results of this approach show an adverse impact on the performance analysis, refinement and further inspection of the technical baseline assumptions can be done.

14 References

Cook, J. R., McDowell-Boyer, L., and Roddy, N. S. (2011). Methods for estimating inventory uncertainty in SRS Performance Assessments. Technical Report SRNL-STI-2010-00666, Rev. 0, Savannah River Site, Aiken, SC.

DiPrete, D. P. (2021). Personal communication on uncertainty in waste package activity measurements to T. S. Whiteside. Technical report, Savannah River National Laboratory.

DiSanza, E. F., O'Dell, M., Willcox, M. V., Bergeron, M., Crowe, B., Krenzien, S., Shott, G., Smith, D., Sully, M., Wood, M., Yucel, V., Pope, H., and Goldston, S. (2008). Review Team report for the E-Area Low-Level Waste Facility DOE 435.1 Performance Assessment at the Savannah River Site. Technical Report SRS Review Team Report, DOE, Washington, DC.

DOE (2017). *DISPOSAL AUTHORIZATION STATEMENT AND TANK CLOSURE DOCUMENTATION*. Washington, DC.

Dunbar, P. (2009). LLW SC Low Level Waste Stream Characterizations - EAV / 2673. Technical Report B-SWCD-E-00002, Savannah River Site.

Fisher-Scientific, T. (2007). Micro REM/Micro Sievert product specification, `https://assts.thermofisher.com/TFS-Assets/LSG/Specification-Sheets/D10492~.pdf`. Technical Report D10492, Thermo Fisher Scientific, Franklin, MA.

Gray, D. (2012). Truck scale accuracy: A weighty issue revealed, `https://fifthwheelst.com/commercial-truck-scales-vs-portable-scales-for-weighing-rvs.html`. Technical report, Fifth Wheel Street.

INEL (1995). A comprehensive inventory of radiological and nonradiological contaminants in waste buried or projected to be buried in the subsurface disposal area of the INEL RWMC during the years 1984–2003, Vol. 1. Technical Report INEL/95-0135, Idaho National Engineering Laboratory, Idaho Falls, ID.

Johnson, C. R. (1995). Defense Programs Division Low Level / Mixed Waste Characterization Plan. Technical Report WSRC-RP-95-00964, Rev. 0, Savannah River Site, Aiken, SC.

Nevada Test Site (2006). Addendum 2 to the Performance Assessment for the Area 5 Radioactive Waste Management Site at the Nevada Test Site, Nye County, Nevada, Update of Perforamnce Asessment methods and results. Technical Report DOE/NV11718–176-ADD2, Bechtel Nevada and Neptune and Company, Las Vegas, NV.

NIST (2019). Specifications, tolerances, and other technical requirements for weighing and measuring devices as adopted by the 104th National conference on Weights and Measures 2019. Technical Report NIST Handbook 44, 2020 Edition, National Institute of Standards and Technology, Washington, DC.

SRNL (2021). Performance Assessment of the E-Area Low-Level Waste Disposal Facility at the Savannah River Site. Technical Report SRNL-STI-2021-00388, Rev. 0, SRNL, Aiken, SC.

SRNS (2021). *Requirements Specification for Software for the Consolidated Waste Tracking System CWTS*. Aiken, SC.

SRS (2006a). 420-D low level waste stream characterization. Technical Report N-CLC-D-00005, Rev. 1, Savannah River Site, Aiken, SC.

SRS (2006b). RBA and CA waste characterization for SDD facilities. Technical Report G-CLC-M-0002, Savannah River Site, Aiken, SC.

SRS (2014). SRS-DTC Software Quality Assurance Plan. Technical Report B-SQP-G-00030, Rev. 1., Savannah River Site, Aiken, SC.

SRS (2021). *SRS Radioactive Waste Requirements Manual 1S*. Aiken, SC.

Taylor, J. R. (1997). *An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements 2nd ed.* University Science Books, Sausalito, CA.

Verst, C. (2021). Effect of modeled source uniformity in a B25 waste box on calculated dose rates and estimated activity. Technical Report SRNL-STI-2021-00291, Rev. 0, Savannah River National Laboratory, Aiken, SC.

Whiteside, T. S. (2017). Characterizing the uncertainty in generator inventory estimates, Part 1. Technical Report SRNL-STI-2017-00740, Rev. 0, Savannah River National Laboratory, Savannah River Site, Aiken SC.

Appendix A Commands to execute method

To get the most recent values from WITS and do the analysis:

```
zmake_dump base all
zmake_dump best all
zanalysis best all all --avg worst --stats --zip --ap2 --ap2ppp
```

After doing additional analysis of prioritized waste streams the following commands may be used:

```
zmake_dump best all --nosql
zanalysis best all all --avg worst --stats --zip --ap2 --ap2ppp
```

Appendix B Code

- zmake_dump.py Listing 1
- zanalysis.py Listing 2
- db_info.py Listing 3
- unc_aux.py Listing 4
- zapp2.py Listing 5
- zisochk.py Listing 6
- zstats.py Listing 7
- zppp.py Listing 8
- zpartials.py Listing 9

Listing 1: zmake_dump

```
#Use Python 32-bit to access the WITS database (because on SRS machines, ACCESS is
    32-bit)

#python file to get the data from the database and then split into parts
import argparse
import csv
import datetime
#import gc
import glob
import itertools
import json
import math
import multiprocessing
import os
import pandas
import random
import re
import shutil
import sys
import subprocess
import time

import db_info
import unc_aux


#-----------------------------------------------------------------------------
#show how long something took
#-----------------------------------------------------------------------------
def print_time(time_start, time_stop):
    time_diff = time_stop-time_start
```

```
    print("This took: ", time_diff, "secs")
    print("           ",time_diff/60,"mins")
#enddef

#-----------------------------------------------------------------------------
#Get the outermost containers in a location (and act and unc) by isotope
#This is used for base case
#-----------------------------------------------------------------------------
def get_containers2(conn, loc, nosql, ddir, unctype):

    rlst  = []
    xfile = os.path.join(ddir,'dump_tmp_'+loc+'.txt')  #./db/dump_base/dump_tmp_...
    if nosql:
        if os.path.exists(xfile):
            with open(xfile,'r') as dfileo:
                rlst = json.load(dfileo)
            #endwith
        else:
            print(xfile,"doesn't exist")
            print("run code without the --nosql option")
            conn.close()
            sys.exit()
        #endif
    else:
        #order the results by activity, so can better focus our efforts on finding
            uncertainty
        SQL = """
        SELECT c.CNTNR_NUM, c.OVERPACK_LVL, c.PARENT_CNTNR_NUM,
                a.ISOTOPIC_ACTIVITY_CI,
                e.ELEMENT_ABBR_ID, e.ATOMIC_MASS_NUM, e.METASTABLE, e.
                    ELEMENT_DERIVATIVE
        FROM ((WITS_TBL_CONTAINER AS c
                INNER JOIN WITS_TBL_CNTNR_ELEMENT_DIST AS a ON c.CNTNR_NUM = a.
                    CNTNR_NUM)
                INNER JOIN WITS_TBL_ELEMENT AS e ON a.ELEMENT_CODE = e.ELEMENT_CODE)
        WHERE c.PARENT_CNTNR_NUM IS NULL AND c.LOCATION_CODE = ?
        ORDER BY a.ISOTOPIC_ACTIVITY_CI DESC;
        """

        cursor = conn.cursor()
        params = (loc)
        print("executing SQL...(this is slow)")
        print("ETRENCH1 is 43 sec, ETRENCH2 is 19 sec, ETRENCH3 is 17 sec, rest is
            <10 sec each")
        time_start = time.time()
        cursor.execute(SQL,params)
        rows = cursor.fetchall()
        print("done")

        time_stop = time.time()
```

```
    print_time(time_start, time_stop)

    for row in rows:
        rlst.append([row.CNTNR_NUM, int(row.OVERPACK_LVL), row.PARENT_CNTNR_NUM,
            float(row.ISOTOPIC_ACTIVITY_CI), row.ELEMENT_ABBR_ID, int(row.
           ATOMIC_MASS_NUM), row.METASTABLE, row.ELEMENT_DERIVATIVE])
    #endfor

    with open(xfile,'w') as dfileo:
        json.dump(rlst, dfileo)
    #endwith
#endif

rowl     = []          #row list
isos     = set()
bias     = 1.00        #use 100% as the baseline bias_factor (unbiased)
meas_unc = 0.05        #use   5% measurement uncertainty (ie uncertainty in the
   total activity measurement of this container)
meth_unc = 0.10        #use  10% method uncertainty (ie how the isotopic
   distribution of this waste stream was determined)

print("getting the uncertainty")
total_unc = math.sqrt(meas_unc**2 + meth_unc**2)

for row in rlst: #each row is the isotope, activity, container
                 #          0, 1,    2,                  3,    4,   5,    6,
                        7
                  #["CSSOURCE", 0, null, 2011.432451958891, "CS", 137, null,
                      null],
    iso = make_iso(row[4], row[5], row[6], row[7])

    isos.add(iso)

    act  = row[3]

    db_info.set_seed() #everytime this is called the seed is incremented by 1
    sig = 0.01
    if unctype == 'uniform':
        rand = random.uniform(-1*sig, sig)
        #rand = 0
    elif unctype == 'gauss':
        rand = random.gauss(0,sig)
    elif unctype == 'triangular':
        rand = random.gauss(-1*sig,sig)
    #endif

    punc = total_unc + rand #percent uncertainty
    #print(punc, total_unc, rand)
    unc  = act * punc       #'actual' uncertainty

    #get the bias
```

```
    #from the priorty package list, there are 83 DTC, 29 CBP, 106 RAD, 0(1) STC
        packages
    #setting the fraction weights=[0.381,0.133,0.482,0.005]
    #this returns a bias for the package, the real method will need to maybe
        return a list of
    #l_calc_method = random.choices(['DTC','CBP','RAD','STC'],weights
        =[0.381,0.133,0.482,0.005],k=1) #tad original
    l_calc_method = random.choices(['DTC','CBP','RAD','STC'],weights
        =[0.28,0.05,0.65,0.02],k=1) #using weights from Tad Memo1
    calc_method = l_calc_method[0]

    if calc_method == 'DTC':
        per_bias = 0.50
    elif calc_method == 'CBP':
        per_bias = 1.00
    elif calc_method == 'RAD':
        #per_bias = random.uniform(0.00,0.50)
        l_calc_method = random.choices([1.00,0.50],weights=[0.02,0.28],k=1) #
            use weights from Tad memo1, RAD is either STC or DTC
        per_bias = l_calc_method[0]
    elif calc_method == 'STC':
        per_bias = 1.00
    #endif


    #returning actual as opposed to relative (percent) bias
    act_bias = per_bias*act

    ocnum     = row[0] #outermost container
    opack_lvl = row[1]

    rowl.append([iso,ocnum,opack_lvl,act,unc,act_bias])
#endfor

print("making the iso dict")
isod ={k:[] for k in isos}

for row in rowl:
    iso      = row[0]
    cnum     = row[1]
    olvl     = row[2]
    act      = float('{:0.4e}'.format(row[3]))
    unc      = float('{:0.4e}'.format(row[4]))
    bias     = row[5]
    for k in isod.keys():
        if k == iso:
            isod[k].append([cnum,olvl,act,unc,bias])
        #endif
    #endfor
#endfor
```

```
    print("done")

    return isod
#enddef

#-------------------------------------------------------------------------------
#turn column keys into isotope symbol
#-------------------------------------------------------------------------------
def make_iso(e_abbr, atomic_mass, meta, e_deriv):
    if meta == None:
        meta = ""
    #endif

    iso = e_abbr + '-' + str(atomic_mass) + str(meta)

    if e_deriv == None:
        pass
    else:
        iso = iso + '-' + str(e_deriv)
    #endif
    return iso
#enddef

#-------------------------------------------------------------------------------
#see if file exists (used in conjuction with nosql option)
#-------------------------------------------------------------------------------
def tmp_file_check(xfile):
    if os.path.exists(xfile):
        return
    else:
        print(xfile,"is missing!!!")
        print("run code without the '--nosql' option")
        sys.exit()
    #endif
#endif

#-------------------------------------------------------------------------------
#make list of isotopes in WITS
#-------------------------------------------------------------------------------
def make_isotopes(nosql, type, ddir):
    xfile = os.path.join(ddir,'dump_tmp_iso.txt')
    if nosql:
        tmp_file_check(xfile)
    else:
        conn = db_info.get_conn(type)

        SQL = """
        SELECT e.ELEMENT_ABBR_ID, e.ATOMIC_MASS_NUM, e.METASTABLE, e.
            ELEMENT_DERIVATIVE
        FROM WITS_TBL_ELEMENT AS e
        ORDER BY e.ATOMIC_MASS_NUM ASC, e.ELEMENT_ABBR_ID, e.METASTABLE DESC, e.
```

```
            ELEMENT_DERIVATIVE DESC;
        """

        cursor = conn.cursor()

        params = ()
        cursor.execute(SQL,params)
        rlst = []
        for row in cursor.fetchall():
            iso = make_iso(row.ELEMENT_ABBR_ID, row.ATOMIC_MASS_NUM, row.METASTABLE,
                row.ELEMENT_DERIVATIVE)

            rlst.append(iso)
        #endfor

        with open(xfile,'w') as dfileo:
            json.dump(rlst, dfileo)
        #endwith
        conn.close()
    #endif
#enddef

#-------------------------------------------------------------------------------
#make dict of ws-wsversion of dict of isotopes and fract in WITS
#-------------------------------------------------------------------------------
def make_wastestreams(nosql, type, ddir):
    xfile = os.path.join(ddir,'dump_tmp_ws.txt')
    if nosql:
        tmp_file_check(xfile)
    else:
        conn = db_info.get_conn(type)

        SQL = """
        SELECT STRM_ID_CODE, STRM_VERSION, WCF_FORM_DOC_NUM, WCF_FORM_REV_NUM
        FROM WITS_TBL_WASTE_STREAM
        ORDER BY STRM_ID_CODE, STRM_VERSION
        """

        cursor = conn.cursor()

        params = ()
        cursor.execute(SQL,params)
        wsdict = {}
        print('not very slow')
        for row in cursor.fetchall():
            docnum = row.WCF_FORM_DOC_NUM
            revnum = row.WCF_FORM_REV_NUM
            if revnum is None:
                wsdict[row.STRM_ID_CODE+"_v."+str(row.STRM_VERSION)] = [docnum,{}]
            else:
                if docnum is None:
```

```
            wsdict[row.STRM_ID_CODE+"_v."+str(row.STRM_VERSION)] = ["None
                ",{}]
        else:
            wsdict[row.STRM_ID_CODE+"_v."+str(row.STRM_VERSION)] = [docnum+"
                _rev."+str(revnum),{}]
        #endif
    #endif
#endfor
print('done')

SQL = """
SELECT a.STRM_ID_CODE, a.STRM_VERSION, a.SUB_STRM_ID_CODE, a.ELEMENT_CODE, a
    .PCT_DIST_CI_NORM, e.ELEMENT_ABBR_ID, e.ATOMIC_MASS_NUM, e.METASTABLE, e
    .ELEMENT_DERIVATIVE
FROM (WITS_TBL_WS_ELEMENT AS a
      INNER JOIN WITS_TBL_ELEMENT AS e ON a.ELEMENT_CODE = e.ELEMENT_CODE);
"""

params = ()
cursor.execute(SQL,params)

rlst = []
print('slow? 3 min')
time_start = time.time()
for row in cursor.fetchall():
    iso = make_iso(row.ELEMENT_ABBR_ID, row.ATOMIC_MASS_NUM, row.METASTABLE,
        row.ELEMENT_DERIVATIVE)
    rlst.append([row.STRM_ID_CODE+"_v."+str(row.STRM_VERSION), int(row.
        SUB_STRM_ID_CODE), iso, float(row.PCT_DIST_CI_NORM)/100])
#endfor
print("done")
time_stop = time.time()
print_time(time_start, time_stop)

# r0                                                    , r1                    ,
    r2 , r3
#[row.STRM_ID_CODE+"_v."+str(row.STRM_VERSION), int(row.SUB_STRM_ID_CODE),
    iso, float(row.PCT_DIST_CI_NORM)/100])
for r in rlst:
    #print(r)
    cutdict = wsdict[r[0]][1]  #get the cutdict
    #print(cutdict)

    cutdict[r[1]]   = {}  #make empty isodict
    wsdict[r[0]][1] = cutdict
#endfor

for r in rlst:
    cutdict = wsdict[r[0]][1]  #get the cutdict
    isodict = cutdict[r[1]]
```

```
            isodict[r[2]] = r[3]
            cutdict[r[1]] = isodict

            wsdict[r[0]][1] = cutdict
        #endfor

        #{strmversion:[WCFFORM,{cut:{iso1:pct_act,iso2:pct_act,...},cut2:{iso...},
            cut3:{iso...}}]}

        with open(xfile,'w') as dfileo:
            json.dump(wsdict, dfileo)
        #endwith

        conn.close()
    #endif
#enddef

#-------------------------------------------------------------------------------
# get all containers, isotope, and activity
#-------------------------------------------------------------------------------
def make_cntnrisoact(nosql, type, ddir):
    xfile = os.path.join(ddir,'dump_tmp_cntnr_act.txt')
    if nosql:
        tmp_file_check(xfile)
    else:
        conn = db_info.get_conn(type)

        SQL = """
        SELECT c.CNTNR_NUM, a.ISOTOPIC_ACTIVITY_CI, e.ELEMENT_ABBR_ID, e.
            ATOMIC_MASS_NUM, e.METASTABLE, e.ELEMENT_DERIVATIVE
        FROM ((WITS_TBL_CONTAINER AS c
                INNER JOIN WITS_TBL_CNTNR_ELEMENT_DIST AS a ON c.CNTNR_NUM = a.
                    CNTNR_NUM)
                INNER JOIN WITS_TBL_ELEMENT AS e ON a.ELEMENT_CODE = e.ELEMENT_CODE)
        WHERE c.FACILITY_CODE ='EAV'
        """

        cursor = conn.cursor()

        params = ()
        print("slow..making all the container information...6 min")

        time_start = time.time()
        cursor.execute(SQL,params)

        #rows = cursor.fetchall() #this failed b/c too big, trying fetchmany below

        dciso = {}
        rlst  = []
        while True:
            rows = cursor.fetchmany(100)
```

```
            if not rows:
                break
            else:
                for r in rows:
                    iso = make_iso(r.ELEMENT_ABBR_ID, r.ATOMIC_MASS_NUM, r.
                        METASTABLE, r.ELEMENT_DERIVATIVE)
                    #print(iso)

                    dciso[r.CNTNR_NUM] = {}

                    rlst.append((r.CNTNR_NUM, iso, r.ISOTOPIC_ACTIVITY_CI))
                #endfor
            #endif
        #endwhile
        print("done")
        time_stop = time.time()
        print_time(time_start, time_stop)

        for (cntr,iso,act) in rlst:
            isodict      = dciso[cntr]  #get isodict, may be empty
            isodict[iso] = act          #assign container[iso] activity
            dciso[cntr]  = isodict
        #endfor

        #{cntnr1:{iso1:act,iso2:act,...},cntnr2:{...}}

        with open(xfile,'w') as dfileo:
            json.dump(dciso, dfileo)
        #endwith
        conn.close()
    #endif
#enddef

#-------------------------------------------------------------------------------
#make container dump file
#-------------------------------------------------------------------------------
def make_inner_cntnrs(nosql, type, ddir):
    xfile = os.path.join(ddir,'dump_tmp_inner.txt')
    if nosql:
        tmp_file_check(xfile)
    else:
        conn = db_info.get_conn(type)

        SQLinner = """
        SELECT c.CNTNR_NUM, c.OVERPACK_LVL, c.PARENT_CNTNR_NUM, c.ACTIVITY_CI
        FROM WITS_TBL_CONTAINER AS c
        WHERE c.FACILITY_CODE = 'EAV' AND c.PARENT_CNTNR_NUM IS NOT NULL
        ORDER BY CNTNR_NUM ASC;
        """

        cursor = conn.cursor()
```

```
        params = ()

        print("not too slow...~13sec")
        time_start = time.time()
        cursor.execute(SQLinner,params)
        rlst = []
        for r in cursor.fetchall():
            rlst.append((r.CNTNR_NUM,int(r.OVERPACK_LVL),r.PARENT_CNTNR_NUM, float(r
                .ACTIVITY_CI)))
        #endfor
        print('done')
        time_stop = time.time()
        print_time(time_start, time_stop)

        with open(xfile,'w') as dfileo:
            json.dump(rlst, dfileo)
        #endwith
        conn.close()
    #endif
#enddef

#-------------------------------------------------------------------------------
#make dict of cntnrs and waste cuts
#-------------------------------------------------------------------------------
def make_cuts(nosql, type, ddir):
    xfile = os.path.join(ddir,'dump_tmp_cuts.txt')
    if nosql:
        tmp_file_check(xfile)
    else:
        msg =       "=============================================================\n"
        msg = msg + "WARNING THIS MAY MESS UP THE db/containers/dir      \n"
        msg = msg + "need to do something like make the dir name be the \n"
        msg = msg + "  hash of the cnum and cutid (md5sum or sha1 crc32)\n"
        msg = msg + "=============================================================\n"
        msg = msg + "Hit [ENTER] to continue."
        #input(msg)

        conn = db_info.get_conn(type)
        SQL = """
        SELECT c.CNTNR_NUM, c.WASTE_CUT_ID, c.STRM_ID_CODE, c.STRM_VERSION, c.
            ELEM_SUB_STRM_ID_CODE, c.CALC_METHOD, c.ACTIVITY_CI
        FROM WITS_TBL_WASTE_CUT AS c
        ORDER BY CNTNR_NUM ASC;
        """

        cursor = conn.cursor()

        params = ()
        cursor.execute(SQL,params)

        rlst = []
```

```
        print("slow part...~1 min")
        time_start = time.time()
        for r in cursor.fetchall():

            ract = r.ACTIVITY_CI
            if ract is None:
                pass
            else:
                ract = float(ract)
            #endif
            rlst.append((r.CNTNR_NUM, r.WASTE_CUT_ID, r.STRM_ID_CODE, int(r.
               STRM_VERSION), int(r.ELEM_SUB_STRM_ID_CODE), r.CALC_METHOD, ract))
        #endfor
        print("rlst done")
        time_stop = time.time()
        print_time(time_start, time_stop)

        #get the keys
        cuts = {}
        for r in rlst:
            cuts[r[0]] = []
        #endfor
        print("keys done")

        for r in rlst:
            cuts[r[0]].append((r[1], r[2], r[3], r[4], r[5], r[6]))
        #endfor
        print("dict done")

        with open(xfile,'w') as dfileo:
            json.dump(cuts, dfileo)
        #endwith
        conn.close()
    #endif
#enddef

#-----------------------------------------------------------------------------
#Get the children containers of parent container
#-----------------------------------------------------------------------------
def get_children(pcnum, inner_cntnrs):
    children = []
    for ic in inner_cntnrs:
        if pcnum == ic[2]: #we have a winner!!
            #cnum  = ic[0]
            #opack = ic[1]
            #act   = ic[3]
            children.append((ic[0],ic[1],ic[3]))
        #endif
    #endfor
    return children
#enddef
```

```
#-------------------------------------------------------------------------------
#Recursive code to put containers in containers
#-------------------------------------------------------------------------------
def arrange_cntnrs(cnum, opack, act, inner_cntnrs):
    arrangement = []
    cinfo       = act #stand in for container_info
    if opack == 0 or opack == 1:
        arrangement.append([cnum, opack, cinfo, []])
    else:
        children = get_children(cnum, inner_cntnrs)
        a = []
        for child in children:
            #                                    cnum,    opack,       act,
            c_arrangement = arrange_cntnrs(child[0], child[1], child[2],
                inner_cntnrs)
            a.extend(c_arrangement)
        #endfor
        arrangement.append([cnum, opack, cinfo, a])
    #endif
    return arrangement
#enddef


#-------------------------------------------------------------------------------
#make DU dict outer cntnrs
#-------------------------------------------------------------------------------
def make_du_outer_cntnrs(loc, nosql, type, ddir):
    xfile = os.path.join(ddir,'dump_tmp_outer_'+loc.lower()+'.txt')
    if nosql:
        tmp_file_check(xfile)
    else:
        inner_cntnrs = db_info.read_dump(ddir,'dump_tmp_inner.txt')

        conn = db_info.get_conn(type)
        #                 r0,              r1
        SQL = """
        SELECT c.CNTNR_NUM, c.OVERPACK_LVL, c.ACTIVITY_CI
        FROM WITS_TBL_CONTAINER AS c
        WHERE c.PARENT_CNTNR_NUM IS NULL AND c.LOCATION_CODE = ?
        ORDER BY c.CNTNR_NUM ASC;
        """

        cursor = conn.cursor()
        params = (loc)

        time_start = time.time()

        cursor.execute(SQL,params)
        du_outer_cntnrs = []
        print("the slow part ~3 sec")
```

```
        for r in cursor.fetchall():
            du_outer_cntnrs.extend(arrange_cntnrs(r.CNTNR_NUM, int(r.OVERPACK_LVL),
                float(r.ACTIVITY_CI), inner_cntnrs)) #here's where it all gets built
        #endfor
        print("done")
        time_stop = time.time()
        print_time(time_start, time_stop)

        #get the keys
        cntrs = {}
        for r in du_outer_cntnrs: #cnum: [opack, act, ic]
            cntrs[r[0]] = [r[1],r[2],r[3]]
        #endfor
        print("dict done")

        with open(xfile,'w') as dfileo:
            json.dump(cntrs, dfileo)
        #endwith
        conn.close()
    #endif
#enddef


#-------------------------------------------------------
#read the 'unc' sheet of the excel file
#has the waste stream uncertainties
#see ./db/unc_analysis_docs/FHW00001HTFJCW-LLW_v.3/unc.xlsx
#-------------------------------------------------------
def get_pandas_df(ufile,sheet):
    df = pandas.read_excel(ufile,sheet,header=0)
    return df

#-------------------------------------------------------------
#read in the uncertainty for this isotope from the
#pandas dataframe
#what happens if unc not found? (should return default)
#unc is expressed as a fraction (100% = 1, 50% = 0.50)
#-------------------------------------------------------------
def get_pdist_val(iso, ufile, col, default_val):
    udf = get_pandas_df(ufile, col) #read the sheet - sheet and column use same
        value (unc or bias)
    try:
        val = udf.loc[udf['isotope']==iso][col].values[0]
    except Exception as e:
        print(e)
        print(iso,'not found in ',ufile,' - is this possible? come check data for
            different characters HYPHEN vs HYPHEN-MINUS etc')
        #input('Hit [ENTER] to continue') #this blows up multiprocessing job
        val = default_val
    #endif
    #if iso == 'AM-242M':
```

```
    #    print(iso, default_val, val, ufile)
    return val

#-------------------------------------------------------
#get the file
#-------------------------------------------------------
def get_uncfile_aux(cdir, ufile="unc.xlsx"):
    if not os.path.exists(cdir):
        #print(cdir, 'not found - come create it')
        #input('Hit [ENTER] to continue')
        ufile = "uncnotdone"
    else:
        ufile = os.path.join(cdir, ufile)
        #print(ufile)
        if not os.path.isfile(ufile):
            ufile = "uncnotdone"
        #endif
    #endif
    return ufile

#-------------------------------------------------------
#gets the excel file with the waste stream uncertainties
#-------------------------------------------------------
def get_ws_uncfile(ws, ws_v):
    waste_stream = ws+"_v."+str(ws_v)

    ddir  = db_info.get_db_subdir()
    wsdir = os.path.join(os.path.join(ddir,'uncert_analysis_docs'),waste_stream)
    #print(wsdir)
    ufile = get_uncfile_aux(wsdir)
    return ufile
#enddef

#-------------------------------------------------------
#cnum and cutid are characters and may not have been
#sanitiy checked.
#This code forces values to be in the set a-zA-Z0-9 and -
#if not in set, change the char to hex-string
#-------------------------------------------------------
def sanitize_inp(input):

    output = ""
    if re.match('^[a-zA-Z0-9-]*$',input):
        output = input
    else:
        for c in input:
            if re.match('^[a-zA-Z0-9-]*$',c):
                newc = c
            else:
                newc = hex(ord(c))
            #endif
```

```
            output = output + newc
        #endfor
    #endif
    return output
#enddef

#-------------------------------------------------------
#gets the excel file with the waste cut uncertainties
#-------------------------------------------------------
def get_meas_uncfile(ws, ws_v, cnum, cutid):
    waste_stream = ws+"_v."+str(ws_v)

    ddir   = db_info.get_db_subdir()
    wsdir  = os.path.join(os.path.join(ddir,'uncert_analysis_docs'),waste_stream)

    wccdir = os.path.join(wsdir, 'waste_cntrs_cuts')

    scnum  = sanitize_inp(cnum)
    if cutid is None:
        cdir = os.path.join(wccdir, scnum)
    else:
        scutid = sanitize_inp(cutid)
        cdir = os.path.join(wccdir, scnum+'-'+scutid)
    #endif
    #print(cdir, cnum, cutid, scnum, scutid)
    ufile = get_uncfile_aux(cdir)

    if ufile == 'uncnotdone':
        #see if there is a generic meas_unc.xlsx file
        ufile = get_uncfile_aux(wccdir, 'meas_unc.xlsx')

    return ufile
#endif

#-------------------------------------------------------------------------
#calculate the uncertainty
#waste_stream = ws+"_v."+ws_v
#dpath        = cnum-cutid OR cnum (if calc_method = DTC)
#there can be nonalphanumeric numbers, in cutid and probably cnum
#(sanitizing here, but might need to be done up in make dump)
#-------------------------------------------------------------------------
def get_unc(iso, iso_act, calc_method, ws, ws_v, dpath, meas_bias):

    db_info.set_seed()

    #---------
    #add a little random uncertainty (+/-1%) when using the default values
    #---------
    sig = 0.01
    if iso in ['ISO-0','X-0']: #this is a test case so cut out the randomness
        sig = 0.00
```

```
#endif

#----------
#Measurement uncertainty (ie tools used to make the measurment)
#----------
#check here to see if meas_unc has been created/updated (ie read the excel file)
ufile = get_meas_uncfile(ws, ws_v, dpath[0], dpath[1])

if calc_method == 'DTC':              #per mfg datasheet
    meas_unc = 0.10
elif calc_method == 'CHAR BY PACK':  #analytical measurement (DiPrete personal
   communication)
    meas_unc = 0.05
elif calc_method == 'RAD':            #either DTC/STC or weight - improve this
   later
    meas_unc = 0.10
elif calc_method == 'STC':            #analytical
    meas_unc = 0.10
#endif
#meas_unc = 10.0 #1000% playing around with unc values to see what happens
#print(meas_bias, ufile)
if ufile == 'uncnotdone':
    pass
else:
    meas_unc  = get_pdist_val(iso, ufile, 'unc', meas_unc)
    meas_bias = get_pdist_val(iso, ufile, 'bias', meas_bias)
    sig       = 0.00 #no need for random uncertainty now
#endif
#print(ufile, meas_unc, meas_bias)

#----------
#Characterization uncertainty
#----------
#using a default 10% characterization uncertainty (ie waste stream
   characterization)
char_unc = 0.10
#char_unc = 10.0 #1000% playing around with unc values to see what happens
if calc_method == 'CHAR BY PACK': #because all uncertainty is in the measurement
    of the waste cut
    char_unc = 0.0
#endif

char_bias= 1.00

#check here to see if char_unc has been updated (ie read the excel file)
ufile = get_ws_uncfile(ws, ws_v)
if ufile == 'uncnotdone':
    pass
else:
    if calc_method == 'CHAR BY PACK':
        print(ufile,"\n","WASTE_CUT IS CBP, this should NOT be here!!!")
```

```
            #print('software is now hung when running in multiprocessing mode!!')
            #sys.exit()
        #endif

        char_unc  = get_pdist_val(iso, ufile, 'unc', char_unc) #get char_unc

        #char_bias, something like: characterization method assumes that <LLD = LLD
            (biased high) or iso_value=0.10*LLD (not sure - biased low or high)
        char_bias = 1.00 #unbiased
        #char_bias < 1 #conservative
        #char_bias > 1 #aggressive
        char_bias = get_pdist_val(iso, ufile, 'bias', char_bias)
        sig       = 0.00 #no need for random uncertainty now
    #endif

    total_unc = math.sqrt(meas_unc**2 + char_unc**2)+random.uniform(-sig,sig)

    punc     = total_unc
    act_unc  = iso_act * punc

    act_bias = iso_act * meas_bias * char_bias

    """
    msg = "-------\n"
    msg = msg + str(meas_unc) + " " + str(meas_bias) + "\n"
    msg = msg + str(char_unc) + " " + str(char_bias) + "\n"
    msg = msg + str(iso_act) + "\n"
    msg = msg + str(act_bias) + "\n"
    msg = msg + "-------\n"
    print(msg)
    """

    """
    if ws == '420DTRITIUM-LLW' and ws_v == 0:
        print(iso, meas_unc, char_unc, calc_method, dpath)
    #endif

    if iso == 'AM-242M':
        print(iso, meas_unc, char_unc, ufile, sig, iso_act, punc, act_unc, act_bias)
    """
    return act_unc, act_bias
#enddef

#-----------------------------------------------------------------------
#call to get the total actual bias, activity, and unc in each container from the
    waste cuts
#bias is done at the measurement technique level (either CUT or CONTAINER)
#see container SWDSL00005 for a good example of this
#              SC00000301
#-----------------------------------------------------------------------
def get_bias_cuts(cnum, cntr_act, dcuts, iso, dws):
```

```
db_info.set_seed()

total_act_true = 0
l_act_unc      = []
dtc_flag       = False
dtc_cut        = ""
tact           = 0  #this is here to check if works

if cntr_act is None: #can be None b/c a container might not have isotope in it (
   esp. inner container in overpack)
    #print(iso,'not in this container', cnum)
    cntr_act = 0
else: #check the cuts in the container
    cuts = dcuts.get(cnum)
    if cuts is None:
        #print(cnum, 'no cuts in this container')
        pass
    else:

        """
        if iso =='SN-121M':#cnum in ['AL34750','AL34751','AL34752','AL07022035
            ']: #== 'KAL65304':
            print("---- beg ---")
            print(cnum)
            print(iso)
            print(cuts)
            print(cntr_act)
            print(total_act_true, l_act_unc, tact)
        """

        swf_a = get_swf_a()
        swf_b = get_swf_b()
        swf_s = get_swf_s()
        swf_h = get_swf_h()
        swf   = []
        swf.extend(swf_a)
        swf.extend(swf_b)
        swf.extend(swf_s)
        swf.extend(swf_h)

        for c in cuts:
            cid         = c[0]
            ws          = c[1]
            ws_v        = c[2]
            elem_sub    = c[3]
            calc_method = c[4]
            cut_act     = c[5] #total activity of the cut

            waste_stream = ws+"_v."+str(ws_v)
            swe_form     = dws[waste_stream][0] #SWE form is 1st item in list
```

```
cutd = dws[waste_stream][1]          #cut_dictionary is 2nd item in
    list
#print(c, cutd, dtc_flag)
"""
if waste_stream in ('TRITIUMBE-LLW_v.0'): #'TRITNHPE-LLW_v.3','
    TRITNHPE-LLW_v.2','TRITNHPE-LLW_v.1','TRITNHPE-LLW_v.0'):
     print(waste_stream,cnum,cid,cut_act, cutd)
"""
"""
if iso == 'AM-242M':
     print(waste_stream,cnum,cid,cut_act) #, cutd)
"""

isod = cutd.get(str(elem_sub))
#print(isod)
#this shouldn't happen - but it does b/c some elem_sub do not exist.
#For example the ISO221S00025-LLW v0 doesn't have a PCT_DIST_CI, but
     v1 does,
#so maybe the WITS_REPORTS code isn't checking for version (need to
    follow up with SRR)
if isod is None:
     print(cnum,'cutd elem_sub')
     print(elem_sub)
     print('software is now hung when running in multiprocessing mode
         !!')
     sys.exit()
#endif

pct_act = isod.get(iso, None)
#print(pct_act)
if pct_act is None and cnum in swf and iso[-2] == '-' and iso[-1].
    isalpha():
     #print("here")
     iso     = iso[:-2]  #chop off the SWF and reset swf-iso to
         generic iso
     pct_act = isod.get(iso, None)
#endif
#print(pct_act)

#print(pct_act)
if pct_act is None:
     #print(iso, "not in this cut",cnum,cid)
     pass
else:
     #print(iso, "in this cut",cnum,cid)
     if calc_method == 'DTC':
          per_bias = 0.50
     elif calc_method == 'CHAR BY PACK':
          per_bias = 1.00
     elif calc_method == 'RAD': #TODO check to see if there might be
         a DB field that could clarify this
```

```
                if cnum == 'CNTR12-1':
                    l_calc_method = random.choices([0.50], weights=[1.00])
                elif cnum =='CNTR13-1':
                    l_calc_method = random.choices([1.00], weights=[1.00])
                else:
                    l_calc_method = random.choices([1.00,0.50], weights
                        =[0.02,0.28])
                #endif
                per_bias = l_calc_method[0]
            elif calc_method == 'STC':
                per_bias = 1.00
            #endif

            #returning actual as opposed to relative (percent) bias
            iso_act = 0
            if calc_method == 'DTC':
                dtc_flag = True
                dtc_cut  = cid
                #don't calculate it here - do it at container level
            else:
                iso_act  = cut_act  * pct_act

                #get the uncertainty
                #print(cnum,cid)
                dpath = [cnum,cid]
                #print(dpath)

                act_unc, act_true = get_unc(iso, iso_act, calc_method, ws,
                    ws_v, dpath, per_bias)
                #l_act_unc.append((iso_act,act_unc))  doesn't want to work
                    with tuple (see get_act2 in zanalysis)
                l_act_unc.append([cid,{iso_act:act_unc},waste_stream,
                    swe_form])  #actual uncertainty

                total_act_true = total_act_true + act_true
            #endif
            tact = tact + iso_act
            #print(dtc_flag)
        #endif
    #endfor

    #print('\t',total_act_true)

    if dtc_flag:
        #print("dtc")
        if total_act_true > 0:
            print(cnum, 'how did this get > 0? DTC and other calc_methods
                should not be mixed?')
            input('pause')
        #endif
```

```
                #get the bias
                per_bias = 0.50
                iso_act  = cntr_act

                #get the uncertainty for DTC
                dpath = [cnum, None]
                if cnum in swf and iso[-2] == '-' and iso[-1].isalpha():
                    iso = iso[:-2]  #chop off the SWF and reset swf-iso to generic
                        iso
                #endif

                act_unc, act_true = get_unc(iso, iso_act, calc_method, ws, ws_v,
                    dpath, per_bias)
                #l_act_unc.append((iso_act,act_unc))  doesn't want to work with
                    tuple
                l_act_unc.append([dtc_cut,{iso_act:act_unc},waste_stream, swe_form])

                total_act_true = total_act_true + act_true
                tact = tact + iso_act

                if len(cuts) > 1:
                    sdtcid = set()
                    for c in cuts:

                        #cid          = c[0]
                        ws           = c[1]
                        ws_v         = c[2]
                        elem_sub     = c[3]
                        calc_method = c[4]
                        #cut_act      = c[5] #total activity of the cut
                        dtcid = ws+str(ws_v)+str(elem_sub)+calc_method
                        sdtcid.add(dtcid)
                    #endfor
                    if len(sdtcid) > 1:
                        print("more than 1 unique DTC cut in a container - is this
                            allowed?") #possible?")
                        #probably - IF they all come from the same waste_stream (but
                            in that case each cut should be called RAD or maybe
                           CHAR BY PACK)
                        print(cnum, cuts)
                        input("xpause")
                    #endif
                #endif
            #endif
        #endif
    #endif

"""
if iso == 'SN-121M': #if cnum in ['AL34750','AL34751','AL34752','AL07022035']:
   #== 'KAL65304':
    print(cnum)
```

```
        print(iso)
        print(cuts)
        print(cntr_act)
        print(total_act_true, l_act_unc, tact)
        print("---- end ---")
    """
    return total_act_true, l_act_unc, tact
#enddef

#-----------------------------------------------------------------------
#recursive call to get the bias and uncertainty in each waste cut / container
#THIS AND get_bias_cuts ARE THE REAL WORKHORSE FOR THE best METHOD
#-----------------------------------------------------------------------
def get_bias(ocnum, cinfo, dcuts, iso, dws, iso_act, dcntract):

    total_act_true = 0
    l_act_unc      = []   #l_act_unc = [l_cuts,l_cntrs]
    tact           = 0
    #opack           = cinfo[0]
    #total_cntnr_act = cinfo[1]
    #cntr_act        = iso_act  #cntainer activity with this isotope
    icntrs    = cinfo[2]

    total_act_true, l_act_unc, tact = get_bias_cuts(ocnum, iso_act, dcuts, iso, dws)
        #get the bias/activity/unc of cuts in container
    #print("total", ocnum, total_act_true, l_act_unc, tact)
    if icntrs == []:
        l_cntrs = []
        #pass
    else:
        #print(ocnum, 'is overpacked')
        inner_total_act_true = 0
        inner_l_act_unc      = []
        inner_tact           = 0

        #get inner container info
        for ic in icntrs:
            id     = dcntract[ic[0]]    #dump_tmp_cntnr_act
            ic_act = id.get(iso, None)  #isotope activity in this container
            #sometimes ic_act is None b/c isotope not in that container or b/c iso
                is a SWF and needs to use the generic ISO value
            if ic_act is None and iso[-2] == '-' and iso[-1].isalpha():
                org_iso = iso[:-2]  #chop off the SWF
                ic_act = id.get(org_iso, None)
            #endif

            #                                                ocnum,       cinfo
                        dcuts  iso  dws  ic_act  dcntract
            ic_total_act_true, ic_l_act_unc, itact = get_bias(ic[0], [ic[1],ic[2],ic
                [3]], dcuts, iso, dws, ic_act, dcntract)
            #print("inner", ic_total_act_true, ic_l_act_unc, itact)
```

```
            inner_l_act_unc.append([ic[0],ic_l_act_unc])

            inner_total_act_true = inner_total_act_true + ic_total_act_true
            inner_tact           = inner_tact + itact
        #endfor
        l_cntrs = inner_l_act_unc
        #l_act_unc.append(inner_l_act_unc)

        total_act_true = total_act_true + inner_total_act_true
        tact           = tact + inner_tact
    #endif
    l_act_unc = [l_act_unc,l_cntrs]

    return total_act_true, l_act_unc, tact
#enddef


def get_swf_a():
        swf_a    = [
                    'RD002993','TF199414','TF199425','TF199419','TF199394','TF199424
                       ',
                    'TF199405','FB32Y00219','A090
                       ','4932','187715','5057','96901306',
                    '5116','5122','5123','5125','4888','5034','5135','5138','96901318',

                    'AL07N70267','TF00000169','KAL46102','KAL46104','KAL46303',
                    'CBALX3267','5136','5144','5165','CBALX3409','96901450',
                    '4887','5148','5162','5167','5185','5193','KAL45901',
                    'AL07N70265','5215','5232','5182','5197','TF00000304','5159',
                    '5131','AL07N60179','5234','5243','5244','TF00000335',
                    'L0006','L0007','L0009','L0008','L0010','AL07N70255',
                    '00020','L0004','TF00000249','ST17209467','ST17961397','
                       AL07N70278',
                    'AL07N60196','L0002','TF00000302','4885','TF00000170',
                    'A117','AL07N70330','TF00000274','5160','TF00000333','AL07N60237
                       ',
                    '180554','5133','CBALX3241
                       ','5124','5166','5191','5231','96901047',
                    'TF00000114','5127','TF199431','AL07N70246','AL07N60172',
                    'A009','5194','5199','5195','5209','5150','5139','96901037',
                    'AL07N80413','AL07N70377','HC27000660','AL07N80500',
                    'HC27000526','ST17981079','TF00000661','AL07N90589',
                    'ST17974386','AL07N80581','FC30N00803','TF00000697',
                    'AL07N90611','AL07N80424','TF00000987','TF00000815',
                    'TF00000715','00157','TF00000955','TF00000884','TF00000724',
                    '00154','AL07N80478','TF00000894','AL07N90672','AL07N70274',
                    '187732','TF00000846','HC27001029','SSD0000009','SSD0000014',
                    'TF00000761','TF00000767','HWLLOST040','SSD0000072',
                    'SC00000301','SC00000414','HT13006483','HT13006589',
                    'HT13006596','HT13006326','CIF0000323','FD00001197',
                    'SSD0000011','FD00001200','FD00001196','FD00001199',
```

```
'CIF0000478', 'CIF0000340', 'CIF0000445', 'CIF0000479',
'SC00000415', 'SC00000417', 'SC00000401', 'FC30S01197',
'FC30S01189', 'HT13006484', 'FT14004385', 'SC00000395',
'TF00000106', 'FC30S01157', 'FC30S01187', 'SC00000394',
'SC00000404', 'SC00000405', 'SC00000413', 'HT13006480',
'SC00000408', 'SC00000410', 'SC00000409', 'TF00001710',
'WS04630015', 'WS04630031', 'KAL65304', 'TF00001719',
'WS04700031', 'WS04750019', 'SC01000050', 'WS04820055',
'TF00001249', 'TF00001261', 'TF00000949', 'TF00001263',
'RD009236', 'TF00001278', 'SSD0000176', 'SSD0000179', 'SSD0000148',
'SSD0000163', 'SSD0000164', 'SSD0000173', 'SSD0000181',
'SSD0000186', 'SSD0000192', 'TF00000632', 'TF00001761',
'SSD0000175', 'RD008996', 'RD008997', 'TF00001789',
'RD009305', 'CSSOURCE', 'TF00001040', 'RD008993', 'SSD0000134', '
    SSD0000110',
'SSD0000077', 'SSD0000189', 'FB32Y01345', 'AL07022035',
'SC00000407', 'SSD0000172', 'SC00000400', 'FB32Y01329',
'RD009349', 'SWDSL00005', 'SC02000068', 'RD008986', 'RD008987', '
    RD008995', 'HT13007092', 'SSD0000187',
'SC02000060', 'AL07022127', 'AL07N90601', 'TF00001340', 'TF00001329
    ', 'WS08900028',
'RD009005', 'SSD0000196', 'SSD0000197', 'SSD0000198', 'AL07022123',
'AL07022122', 'AL07022126', 'WS08890026', 'AL07022130',
'AL07022131', 'AL07022132', 'AL07022133', 'RD009352', 'RD009300',
'FD00008174', 'FD00008176', 'FD00008175', 'HT13006556',
'HC27001644', 'RD009516', 'RD009511', 'RD009510', 'RD009522',
'RD009566', 'RD009425', 'RD009426', 'RD009427',
'00359','00358','00362','RD009416',
'TF00001125', 'TF00001944', 'TF00001945', 'TF00001949',
'TF00001950', 'TF00001951', 'TF00000578', 'TF00001948',
'TF00001599', 'TF00001124', 'ST17991764', 'RD009001', 'RD009002',
'TF00001600', 'TF00001598', 'TF00001897', 'HT13007437',
'TF00001952', '5130', 'RD008992', 'A047', '00361', 'L0011', 'RD009607
    ',
'FC30S01382', 'RD009548', 'AL07N80552', 'SC00000412',
'4886','5129','5142','5147','5198','5224','SSD0000191',
'RD009594', 'FD00002475', 'FD00000852', 'FD00000864', 'FD00000898',
'SSD0000199', 'TF00001818', 'TF00001819', 'TF00001813',
'TF00001133', 'TF00001874', 'TF00001873', 'ST17131377',
'FD00000552', 'TF00001817', 'ST17222678', 'TF00001875',
'ST17240954', 'FD00000551', 'HT13007627', 'SWD041358',
'ST17131380', 'RD009631', 'RD009646', 'AL07042339', 'AL07042340', '
    AL07042357', 'AL07042347',
'SD00005548', 'SD00005549', 'SD00005550', 'SD00005551',
'SD00005552', 'SD00004041', 'FD00000481', 'SD00004022',
'FD00000547', 'FD00000550', 'SWD051172', 'SWD051173', 'RD009653',
'SD00004042', 'SD00004040', 'SW20021023', 'SFP0001002',
'SFP0001003', 'SD00004038', 'SD00004039', 'SD00004002',
'SD00005224', 'SD00004021', 'SFP0001139', 'SD00003950',
'SFP0001080', 'SD00005217', 'SD00005360', 'SD00005348',
'SD00005351', 'AL07070023', 'AL07070025', 'RD00009731',
```

'RD00009732', 'KAL88204', 'CIGXB25',
'AL001067', 'AL001068', 'AL001069', 'AL001070', 'AL001071', 'AL001077', 'AL001078', 'AL001079', 'AL001926', 'AL001928', 'AL002233', 'AL002234', 'AL002235', 'AL002236', 'AL002237', 'AL002238', 'AL002239', 'AL002240', 'AL002241', 'AL002243', 'AL002244', 'AL002246', 'AL002247', 'AL002248', 'AL002524', 'AL002593', 'AL003928', 'AL006893', 'AL006897', 'AL006918', 'AL009285', 'AL009286', 'AL009287', 'AL009291', 'AL009292', 'AL009293', 'AL009294', 'AL009295', 'AL009487', 'AL009766', 'AL009770', 'AL009780', 'AL009781', 'AL009782', 'AL009874', 'AL07C50004', 'AL07C50017', 'AL07C50019', 'AL07C50024', 'AL07C50025', 'AL07C50028', 'AL07C50031', 'AL07C50032', 'AL07C50038', 'AL07C50046', 'AL07C50049', 'AL07C50067', 'AL07C50070', 'AL07C50076', 'AL07C50083', 'AL07C50086', 'AL07C50154', 'AL07C50223', 'AL07C50226', 'AL07C50228', 'AL07C50233', 'AL07C50235', 'AL07C50237', 'AL07C50238', 'AL34749', 'AL34750', 'AL34751', 'AL34752', 'AL37332', 'AL37333', 'AL37335', 'AL37337', 'AL37341', 'AL37342', 'AL37343', 'AL37344', 'AL37345', 'AL37346', 'AL37347', 'AL37349', 'AL37350', 'AL40003', 'AL40004', 'AL40005', 'AL40006', 'AL40007', 'AL40011', 'AL40013', 'AL40014', 'AL40015', 'AL40016', 'AL40017', 'AL40018', 'AL40019', 'AL40020', 'AL40021', 'AL40022', 'AL40023', 'AL40026', 'AL40027', 'AL40848', 'AL41931', 'AL41932', 'AL41933', 'AL41935', 'AL41936', 'AL41937', 'AL41943', 'AL41944', 'AL41946', 'AL46389', 'AL46391', 'AL46392', 'AL46394', 'AL46395', 'AL46396', 'AL46398', 'AL46399', 'AL48035', 'AL48038', 'AL63918', 'AL63919', 'AL64197', 'AL67159', 'AL67160', 'AL67165', 'AL67178', 'AL67180', 'AL67181', 'ALWQL00094', 'ALWQL00095', 'ALWQL00096', 'ALWQL00097', 'ALWQL00099', 'ALWQL00100', 'ALWQL00101', 'ALWQL00102', 'ALWQL00103', 'ALWQL00105', 'ALWQL00106', 'ALWQL00108', 'ALWQL00117', 'ALWQL00121', 'ALWQL00122', 'ALWQL00124', 'ALWQL00126', 'ALWQL00128', 'ALWQL00129', 'ALWQL00132', 'ALWQL00133', 'ALWQL00135', 'ALWQL00136', 'ALWQL00137', 'ALWQL00140', 'ALWQL00144', 'ALWQL00174', 'ALWQL00176', 'ALWQL00258', 'ALWQL00262', 'ALWQL00266', 'ALWQL00378', 'ALWQL00514', 'ALWQL00515', 'ALWQL00523', 'ALWQL00530', 'ALWQL00531', 'ALWQL00544', 'ALWQL00545', 'ALWQL00553', 'ALWQL00574', 'ALWQL00845', 'ALWQL00846', 'ALWQL00847', 'ALWQL00848', 'ALWQL00849', 'ALWQL00850', 'ALWQL00852', 'ALWQL00854', 'ALWQL00855', 'ALWQL00856', 'ALWQL00857', 'ALWQL00858', 'ALWQL00859', 'ALWQL00860', 'ALWQL00861',
'C31582', 'C32011', 'C32019', 'C32020', 'C32021', 'C32118', 'C32119', 'C32121', 'C32122', 'C32123', 'C32124', 'C32126', 'C32127', 'C32128', 'C32129', 'C32130', 'C32131', 'C32132', 'C32133', 'C32134', 'C32135', 'C32136', 'C32137', 'C32138', 'C32557', 'C32861', 'C34035', 'C34036', 'C34038', 'C34039', 'C34040', 'C34043', 'C34047', 'C34048', 'C34088', 'C34092', 'C34093', 'C34094', 'C34095', 'C34096', 'C34097', 'C34098', 'C34099', 'C34100', 'C34101', 'C34102', 'C34103', 'C34104', 'C34105', 'C34106', 'C34108', 'C34109', 'C34110', 'C34111', 'C34112', 'C34113', 'C34114', 'C34115', 'C34116', 'C34117', 'C34118', 'C34120', 'C34121', 'C34122', 'C34123', 'C34124

```
    ', 'C34125', 'C34126', 'C34127', 'C34130', 'C34131', 'C34132', '
    C34141', 'C34143', 'C34144', 'C34146', 'C34147', 'C34148', 'C34149
    ', 'C34150', 'C34151', 'C34152', 'C34156', 'C34165', 'C34166', '
    C34167', 'C34168', 'C34169', 'C34170', 'C34173', 'C34175', 'C34176
    ', 'C34182', 'C34184', 'C34185', 'C34187', 'C34199', 'C34200', '
    C34201', 'C34202', 'C34203', 'C34204', 'C34205', 'C34212', 'C34216
    ', 'C34217', 'C34218', 'C34219', 'C34220', 'C34221', 'C34222', '
    C34223', 'C34227', 'C34228', 'C34229', 'C34230', 'C34231', 'C34232
    ', 'C34421', 'C34431', 'C34652', 'C34653', 'C34668', 'C34670', '
    C34671', 'C34672', 'C34673', 'C34674', 'C34676', 'C34677', 'C34702
    ', 'C34710', 'C34711', 'C34717', 'C34745', 'C36502', 'C36508', '
    C36511', 'C36561', 'C36570', 'C36573', 'C36574', 'C36575', 'C36605
    ', 'C36639', 'C36713', 'C36715', 'C36805', 'C36807', 'C36813', '
    C36814', 'C36815', 'C36816', 'C36817', 'C36834', 'C36837', 'C36847
    ', 'C36868', 'C36873', 'C36875', 'C36876', 'C36910', 'C36915',
'C40035', 'C40036', 'C40099', 'C40100', 'C41004', 'C41253', 'C41420', '
    C41569', 'C41570', 'C41576', 'C41579', 'C41580', 'C41587', 'C41601
    ', 'C41602', 'C41603', 'C41604', 'C41606', 'C41607', 'C41608', '
    C41610', 'C41611', 'C41625', 'C41751', 'C41753', 'C41764', 'C41776
    ', 'C41777', 'C41778', 'C41779', 'C41780', 'C41781', 'C41782', '
    C41783', 'C42726', 'C42732', 'C42737', 'C42741', 'C42742', 'C42743
    ', 'C42744', 'C42815', 'C42816', 'C42822', 'C42823', 'C42826', '
    C42827', 'C42828', 'C42829', 'C42830', 'C42833', 'C42834', 'C42835
    ', 'C42836', 'C42837', 'C42841', 'C42842', 'C42852', 'C42854', '
    C42855', 'C42856', 'C42859', 'C42872', 'C42873', 'C42883', 'C42884
    ', 'C43004', 'C43005', 'C43006', 'C43007', 'C43008', 'C43009', '
    C43010', 'C43011', 'C43012', 'C43013', 'C43015', 'C43016', 'C43017
    ', 'C43018', 'C43019', 'C43021', 'C43022', 'C43023', 'C43024', '
    C43025', 'C43026', 'C43027', 'C43028', 'C43029', 'C43030', 'C43031
    ', 'C43032', 'C43033', 'C43034', 'C43036', 'C43037', 'C43039', '
    C43040', 'C43042', 'C43043', 'C43044', 'C43045', 'C43046', 'C43047
    ', 'C43048', 'C43049', 'C43050', 'C43151', 'C43160', 'C43175', '
    C43176', 'C43177', 'C43178', 'C43179', 'C43180', 'C43182', 'C43184
    ', 'C43186', 'C43187', 'C43194', 'C43197', 'C43200', 'C43312', '
    C43321', 'C43325', 'C43327', 'C43328', 'C43330', 'C43331', 'C43332
    ', 'C43333', 'C43336', 'C43338', 'C43339', 'C43340', 'C43346', '
    C43347', 'C43348', 'C43350', 'C43351', 'C43353', 'C43356', 'C43359
    ',
'CHB0000100', 'CIF0000428', 'CIF0000429', 'CIF0000430', 'CIF0000431
    ', 'CIF0000432', 'CIF0000434', 'CIF0000435', 'CIF0000436', '
    CIF0000437', 'CIF0000438', 'CIF0000439', 'CIF0000440', '
    CIF0000441', 'CIF0000442', 'CIF0000446', 'CIF0000447', '
    CIF0000448', 'CIF0000449', 'CIF0000451', 'CIF0000453', '
    CIF0000454', 'CIF0000455', 'CIF0000456', 'CIF0000457', '
    CIF0000458', 'CIF0000460', 'CIF0000461', 'ET11000288', '
    ET11000289', 'ET11000290', 'ET11000291', 'ET11000292', '
    ET11000293', 'ET11000294', 'ET11000295', 'ET11000296', '
    ET11000297', 'ET11000298', 'ET11000299',
'FB32C01385', 'FB32SC0001',
'FC30000512', 'FC30000526', 'FC30000552', 'FC30000570', 'FC30000572
    ', 'FC30000576', 'FC30C00772', 'FC30C00774', 'FC30C00776', '
```

FC30C00778', 'FC30C00780', 'FC30C00782', 'FC30C00784', '
FC30C00786', 'FC30C00790', 'FC30C00792', 'FC30C00796', '
FC30C00812', 'FC30C00824', 'FC30C00826', 'FC30C00828', '
FC30C00830', 'FC30C00832', 'FC30C00834', 'FC30C00836', '
FC30C00838', 'FC30C00840', 'FC30C00842', 'FC30C00844', '
FC30C00846', 'FC30C00848', 'FC30C00850', 'FC30C00852', '
FC30C00854', 'FC30C00856', 'FC30C00858', 'FC30C00860', '
FC30C00862', 'FC30C00864', 'FC30C00866', 'FC30C00868', '
FC30C00872', 'FC30N24342', 'FC30N24343', 'FC30N24344', '
FC30N24345', 'FC30N24346', 'FC30N24347', 'FC30N24348', '
FC30N24349', 'FC30N24350', 'FC30N24351', 'FC30N24352', '
FC30N24353', 'FC30N24362', 'FC30N24363', 'FC30N24364', '
FC30N24365', 'FC30N24366', 'FC30N24367', 'FC30N24368', '
FC30N24369', 'FC30N24370', 'FC30N24371', 'FC30N24372', '
FC30N24373', 'FC30N24374', 'FC30S28662', 'FC30S28663', '
FC30S28664', 'FC30S28665', 'FC30S28666', 'FC30S28667', '
FC30S28668', 'FC30S28669', 'FC30S28670', 'FC30S28671', '
FC30S28672', 'FC30S28705', 'FC30S28706', 'FC30S28707', '
FC30S28708', 'FC30S28709', 'FC30S28710', 'FC30S28711', '
FC30S28712', 'FC30S28713', 'FC30S28714', 'FC30S28762', '
FC30S28763', 'FC30S28764', 'FC30S28765', 'FC30S28766', '
FC30S28767', 'FC30S28768', 'FC30S28807', 'FC30S28808', '
FC30S28809', 'FC30S28810', 'FC30S28811', 'FD00000005', '
FD00000026', 'FD00001675', 'FD00001676', 'FD00001681', '
FD00001692', 'FD00001700', 'FD00001701', 'FD00001703', '
FD00001812', 'FD00001813', 'FD00001814', 'FD00001821', '
FT14001229', 'FT14001241', 'FT14001243', 'FT14001250', '
FT14001252', 'FT14001254', 'FT14001255', 'FT14001256', '
FT14001257', 'FT14001258', 'FT14001259', 'FT14001260', '
FT14001262', 'FT14001263', 'FT14001265', 'FT14001276', '
FT14001285', 'FT14004417', 'FT14004418', 'FT14004419', '
FT14004420', 'FT14004421', 'FT14004422', 'FT14004423', '
FT14004426', 'FT14004427', 'FT14004428', 'FT14004429', '
FT14004430', 'FT14004431', 'FT14004432', 'FT14004433', '
FT14004434', 'FT14004435', 'FT14004436', 'FT14004437', '
FT14004438', 'FT14004439', 'FT14004440', 'FT14004441', '
FT14004442', 'FT14004443', 'FT14004444', 'FT14004445', '
FT14004446', 'FT14004447',
'HBL00SC002', 'HBL00SC003', 'HBL00SC004', 'HBL00SC005', 'HBL00SC006
', 'HBL00SC007', 'HBL00SC008', 'HBL00SC011', 'HBL00SC022', '
HC27500040', 'HC27500046', 'HC27500102', 'HC27500105', '
HC27500108', 'HC27500126', 'HC27500132', 'HC27500136', '
HC27500142', 'HC27500148', 'HC27500149', 'HC27500151', '
HC27500152', 'HC27500153', 'HC27500157', 'HC27500174', '
HC27500176', 'HC27500223', 'HC27500528', 'HC27500529', '
HC27500530', 'HC27500532', 'HC27500533', 'HC27500540', '
HC27500551', 'HC27500558', 'HC27500574', 'HC27500583', '
HC27500585', 'HC27500588', 'HC27500704', 'HC27500705', '
HC27500710', 'HC27500765', 'HC27500786', 'HC27501408', '
HC27501412', 'HC275261', 'HC2752610', 'HC2752611', 'HC2752612', '
HC2752613', 'HC2752614', 'HC2752615', 'HC2752616', 'HC2752617', '

```
    HC2752618', 'HC2752619', 'HC275262', 'HC2752620', 'HC275263', '
    HC275264', 'HC275265', 'HC275266', 'HC275267', 'HC275268', '
    HC275269', 'HC276601', 'HC2766010', 'HC2766011', 'HC2766012', '
    HC2766013', 'HC2766014', 'HC2766015', 'HC2766016', 'HC2766017', '
    HC2766018', 'HC2766019', 'HC276602', 'HC2766020', 'HC2766021', '
    HC2766022', 'HC2766023', 'HC2766024', 'HC2766025', 'HC276603', '
    HC276604', 'HC276605', 'HC276606', 'HC276607', 'HC276608', '
    HC276609', 'HC27700040', 'HC27700041', 'HC27700168', 'HC27700185
    ', 'HC27700194', 'HC27700204', 'HC27700206', 'HC27700225', '
    HC27700226', 'HC27700257', 'HC27700258', 'HC27700264', '
    HC27700291', 'HC27700413', 'HC27700528', 'HC27701267', '
    HC27701272', 'HC27701273', 'HC27701281', 'HC27701290', '
    HC27701293', 'HC27701295', 'HC27701300', 'HC27701301', '
    HC27701303', 'HC27701327', 'HC27701352', 'HC27702225', '
    HC27702449', 'HC27703017', 'HC27703019', 'HC27703126', '
    HC27703212', 'HC27C00040', 'HC27C00041', 'HC27C00215', '
    HC27C00220', 'HC27C00221', 'HC27C00222', 'HC27C00223', '
    HC27C00224', 'HC27C00227', 'HC27C00252', 'HC27C00334', '
    HT13000625', 'HT13000626', 'HT13000629', 'HT13000631', '
    HT13000632', 'HT13000633', 'HT13000638', 'HT13000639', '
    HT13000640', 'HT13000641', 'HT13000642', 'HT13000643', '
    HT13000645', 'HT13000646', 'HT13000648', 'HT13000649', '
    HT13000651', 'HT13000652', 'HT13000654', 'HT13000655', '
    HT13000656', 'HT13000658', 'HT13000659', 'HT13000660', '
    HT13002825', 'HT13002873', 'HT13002941', 'HT13002984', '
    HT13003792', 'HT13003793', 'HT13003795', 'HT13003797', '
    HT13003798', 'HT13003799', 'HT13003800', 'HT13003801', '
    HT13003802', 'HT13003803', 'HT13003804', 'HT13003805', '
    HT13003806', 'HT13003807', 'HT13003808', 'HT13003811', '
    HT13003812', 'HT13003813', 'HT13003814', 'HT13003815', '
    HT13003816', 'HT13003818', 'HT13003819', 'HT13003820', '
    HT13003821',
'INVBAL0001',
'LAUGW00168', 'LAUGW00206', 'LAUGW00214', 'LAUGW00222', 'LAUGW00223
    ', 'LAUGW00226', 'LAULW00775', 'LAULW00776', 'LAULW00777', '
    LAULW00778', 'LAULW00779', 'LAULW00780', 'LAULW00781', '
    LAULW00782', 'LAULW00783', 'LAULW00784', 'LAULW00785', '
    LAULW00786', 'LAULW00787', 'LAULW00788', 'LAULW00789', '
    LAULW00790', 'LAULW00791', 'LAULW00792', 'LAULW00793', '
    LAULW00794', 'LAULW00795', 'LAULW00796', 'LAULW00797', '
    LAULW00798', 'LAULW00799', 'LAULW00800', 'LAULW00801', '
    LAULW00802', 'LAULW00803', 'LAULW00804', 'LAULW00805', '
    LAULW00806', 'LAULW00807', 'LAULW00808', 'LAULW00809', '
    LAULW00810', 'LAULW00811', 'LAULW00812', 'LAULW00813', '
    LAULW00814', 'LAULW00815', 'LAULW00816', 'LAULW00817', '
    LAULW00818', 'LAULW00819', 'LAULW00820', 'LAULW00821', '
    LAULW00822', 'LAULW00823', 'LAULW00824', 'LAULW00825', '
    LAULW00826', 'LAULW00827', 'LAULW00828', 'LAULW00829', '
    LAULW00833', 'LAULW00835', 'LAULW00837', 'LAULW00838', '
    LAULW00839', 'LAULW00840', 'LAULW00845', 'LAULW00847', '
    LAULW00850', 'LAULW00851', 'LAULW00853', 'LAULW00855', '
```

LAULW00856', 'LAULW00859', 'LAULW00860', 'LAULW00861', '
LAULW00862', 'LAULW00863', 'LAULW00866', 'LAULW00867', '
LAULW00870', 'LAULW00871', 'LAULW00872', 'LAULW00876', '
LAULW00880', 'LAULW00881', 'LAULW00884', 'LAULW00885', '
LAULW00886', 'LAULW00887', 'LAULW00888', 'LAULW00891', '
LAULW00892', 'LAULW00893', 'LAULW00894', 'LAULW00895', '
LAULW00896', 'LAULW00898', 'LAULW00899', 'LAULW00900', '
LAULW00901', 'LAULW00902', 'LAULW00903', 'LAULW00904', '
LAULW00906', 'LAULW00907', 'LAULW00908', 'LAULW00909', '
LAULW00911', 'LAULW00912', 'LAULW00913', 'LAULW00914', '
LAULW00915', 'LAULW00916', 'LAULW00918', 'LAULW00920', '
LAULW00921', 'LAULW00922', 'LAULW00923', 'LAULW00924', '
LAULW00925', 'LAULW00926', 'LAULW00928', 'LAULW00929', '
LAULW00930', 'LAULW00931', 'LAULW00932', 'LAULW00933', '
LAULW00934', 'LAULW00935', 'LAULW00936', 'LAULW00937', '
LAULW00938', 'LAULW00939', 'LAULW00940', 'LAULW00941', '
LAULW00942', 'LAULW00978', 'LAULW01223', 'LAULW01224', '
LAULW01225', 'LAULW01226', 'LAULW01227', 'LAULW01228', '
LAULW01229', 'LAULW01231', 'LAULW01232', 'LAULW01233', '
LAULW01234', 'LAULW01235', 'LAULW01237', 'LAULW01240', '
LAULW01242', 'LAULW01243', 'LAULW01244', 'LAULW01245', '
LAULW01249', 'LAULW01250', 'LAULW01251', 'LAULW01256', '
LAULW01263', 'LAULW01264', 'LAULW01265', 'LAULW01266', '
LAULW01267', 'LAULW01268', 'LAULW01271', 'LAULW01272', '
LAULW01275', 'LAULW01276', 'LAULW01277', 'LAULW01278', '
LAULW01279', 'LAULW01280', 'LAULW01281', 'LAULW01282', '
LAULW01283', 'LAULW01284', 'LAULW01285', 'LAULW01286', '
LAULW01288', 'LAULW01289', 'LAULW01290', 'LAULW01291', '
LAULW01292', 'LAULW01293', 'LAULW01294', 'LAULW01295', '
LAULW01296', 'LAULW01297', 'LAULW01298', 'LAULW01299',
'LAULW01301', 'LAULW01302', 'LAULW01304', 'LAULW01305', 'LAULW01306
', 'LAULW01307', 'LAULW01308', 'LAULW01309', 'LAULW01310', '
LAULW01313', 'LAULW01317', 'LAULW01318', 'LAULW01319', '
LAULW01320', 'LAULW01321', 'LAULW01322', 'LAULW01325', '
LAULW01326', 'LAULW01327', 'LAULW01330', 'LAULW01331', '
LAULW01332', 'LAULW01333', 'LAULW01334', 'LAULW01402', '
LAULW01403', 'LAULW01405', 'LAULW01407', 'LAULW01424',
'LLWIN00001', 'LLWIN00002', 'LLWIN00084', 'LLWIN00085', 'LLWIN00094
', 'LLWIN01030', 'LLWIN01231', 'LLWIN01232', 'RD00000029', '
RD00000104', 'RD00000375', 'RD00001295', 'RD00002260', '
RD00003098', 'RD00003130', 'RD00003131', 'RD00003132', '
RD00003136', 'RD00003137', 'RD00003138', 'RD00003139', '
RD00003140', 'RD00003141', 'RD00003143', 'RD00003144', '
RD00003146', 'RD00003147', 'RD00003148', 'RD00003149', '
RD00003150', 'RD00003151', 'RD00003155', 'RD00003156', '
RD00003157', 'RD00003158', 'RD00003159', 'RD00003160', '
RD00003161', 'RD00003162', 'RD00003165', 'RD00003166', '
RD00003177', 'RD00003178', 'RD00003179', 'RD00003180', 'RD001416
', 'RD003265', 'RD003274', 'RD003275', 'RD003277', 'RD003280', '
RD003395', 'RD003396', 'RD003397', 'RD003398', 'RD003399', '
RD003400', 'RD003401', 'RD003402', 'RD003403', 'RD003413', '

```
    RD003414', 'RD003417', 'RD003436', 'RD003437', 'RD003439', '
    RD003443', 'RD003444', 'RD003445', 'RD003446', 'RD003447', '
    RD003448', 'RD003449', 'RD003493', 'RD003494', 'RD003495', '
    RD003497', 'RD003499', 'RD003500', 'RD003502',
'S00504', 'S00510', 'S00517', 'S00527', 'S00535', 'S00536', 'S00544', '
    S00545', 'S00546', 'S00548', 'S00553', 'S00555', 'S00559', '
    SS00000151', 'SS00000152', 'SS00000153', 'SS00000155', '
    SS00000156', 'SS00000157', 'SS00000162', 'SS00000170', '
    SS00000183', 'SS00000184', 'SS00000190', 'SS00000245', '
    SS00000256', 'SS00000267', 'SS00000270', 'SS00000378', '
    SS00000380', 'SS00000702', 'SS00000729', 'SS00000732', '
    SS00000761', 'SS00000765', 'SS00000767', 'SS00000768', '
    SS00000770', 'SS00000771', 'SS00000775', 'SS00000776', '
    SS00000777', 'SS00000778', 'SS00000779', 'SS00000780', '
    SS00000781', 'SS00000782', 'SS00000783', 'SS00000784', '
    SS00000785', 'SS00000786', 'SS00000787', 'SS00000788', '
    SS00000789', 'SS00000790', 'SS00000791', 'SS00000792', '
    SS00000793', 'SS00000794', 'SS00000795', 'SS00000796', '
    SS00000797', 'SS00000798', 'SS00000799', 'SS00000800', '
    SS00000801', 'SS00000802', 'SS00000803', 'SS00000804', '
    SS00000805', 'SS00000806', 'SS00000807', 'SS00000808', '
    SS00000809', 'SS00000810', 'SS00000811', 'SS00000812', '
    SS00000813', 'SS00000814', 'SS00000815', 'SS00000816', '
    SS00000817', 'SS00000818', 'SS00000819', 'SS00000820', '
    SS00000821', 'SS00000822', 'SS00000823', 'SS00000824', '
    SS00000825', 'SS00000826', 'SS00000827', 'SS00000828', '
    SS00000829', 'SS00000830', 'SS00000831', 'SS00000832', '
    SS00000833', 'SS00000834', 'SS00000835', 'SS00000836', '
    SS00000837', 'SS00000838', 'SS00000839', 'SS00000840', '
    SS00000841', 'SS00000842', 'SS00000843', 'SS00000844', '
    SS00000845', 'SS00000846', 'SS00000847', 'SS00000848', '
    SS00000849', 'SS00000850', 'SS00000851', 'SS00000852', '
    SS00000853', 'SS00000854', 'SS00000858', 'SS00000859', '
    SS00000860', 'SS00000861', 'SS00000863', 'SS00000865', '
    SS00000866', 'SS00000867', 'SS00000868', 'SS00000872', '
    SS00000873', 'SS00000874', 'SS00000875', 'SS00000878', '
    SS00000879',
'ST16000007', 'ST16000010', 'ST16000042', 'ST16000043', 'ST16000045
    ', 'ST16000069', 'ST16000070', 'ST16000071', 'ST16000073', '
    ST16000075', 'ST16000076', 'ST16000078', 'ST16000096', '
    ST16000099', 'ST16000100', 'ST16000102', 'ST16000118', '
    ST16000141', 'ST16000142', 'ST16000143', 'ST16000144', '
    ST16000148', 'ST16000156', 'ST16000159', 'ST16000160', '
    ST16000162', 'ST16000163', 'ST16000164', 'ST16000165', '
    ST16000183', 'ST16000189', 'ST16000193', 'ST16000194', '
    ST16000217', 'ST16000221', 'ST16000223', 'ST16000228', '
    ST16000229', 'ST16000233', 'ST16000235', 'ST16000247', '
    ST16000258', 'ST16000275', 'ST16000276', 'ST16000277', '
    ST16000279', 'ST16000280', 'ST16000282', 'ST16000283', '
    ST16000285', 'ST16000286', 'ST16000287', 'ST16000293', '
    ST16000294', 'ST16000295', 'ST16000296', 'ST16000297', '
```

```
    ST16000298','ST16000300','ST17000070','ST17000434','
    ST17000449','ST17000450','ST17000581','ST17000594','
    ST17000599','ST17000690','ST17901655','ST17906114','
    ST17906115','ST17906257','ST17906258','ST17906259','
    ST17906260','ST17906262','ST17906263','ST17906315','
    ST17906316','ST17906317','ST17906318','ST17906423','
    ST17906424','ST17906425','ST17906426','ST17906435','
    ST17906436','ST17906438','ST17906466','ST17906513','
    ST17908220','ST18901735','ST18901736','ST18901737','
    ST18902916','ST18902917','ST18902918','ST18902919','
    ST18906014','ST18906731','ST18906732','ST18906733','
    ST18906762','ST18906771','ST18906772','ST18906773','
    ST18906774','ST18906775',
'TFC0000601','TFC0000604','TFC0000606','TFC0000610','TFC0002651
    ','TFC0002652','TFC0002654','TFC0002656','TFC0002658','
    TFC0002659','TFC0002661','TFC0002663','TFC0002674','
    TFC0002675','TFC0002677','TFC0002680','TFC0002682','
    TFC0002684','TFC0002687','TFC0002690','TFC0002693','
    TFC0002695','TFC0002698','TFC0002700','TFC0002703','
    TFC0002710','TFC0002715','TFC0002716','TFC0002717','
    TFC0002719','TFC0002720','TFC0002721','TFC0002722','
    TFC0002725','TFC0002733','TFC0002734','TFC0002735','
    TFC0002749','TFC0002751','TFC0002752','TFC0002753','
    TFC0002755','TFC0002756','TFC0002759','TFC0002761','
    TFC0002762','TFC0002763','TFC0002764','TFC0002768','
    TFC0002769','TFC0002772','TFC0002778','TFC0002781','
    TFC0002782','TFC0002784','TFC0002785','TFC0002790','
    TFC0002791','TFC0002793','TFC0002795','TFC0002796','
    TFC0002797','TFC0002798','TFC0002799','TFC0002800','
    TFC0002801','TFC0002802','TFC0002804','TFC0002808','
    TFC0002809','TFC0002810','TFC0002811','TFC0002812','
    TFC0002813','TFC0002814','TFC0002815','TFC0002816','
    TFC0002818','TFC0002820','TFC0002822','TFC0002828','
    TFC0002829','TFC0002830','TFC0002831','TFC0002834','
    TFC0002836','TFC0002839','TFC0002852','TFC0002853','
    TFC0002854','TFC0002861','TFC0002864','TFC0002868','
    TFC0002869','TFC0002873','TFC0002874','TFC0002878','
    TFC0002884','TFC0002894','TFC0002902','TFC0002911','
    TFC0002912','TFC0002917','TFC0002929','TFC0002932','
    TFC0002942','TFC0002944','TFC0002946','TFC0002947','
    TFC0002958','TFC0002959','TFC0002960','TFC0002962','
    TFC0002964','TFC0002968','TFC0002971','TFC0002973','
    TFC0002977','TFC0002980','TFC0002981','TFC0002982','
    TFC0002983','TFC0002996','TFC0002998','TFC0002999',
'TFC0003000','TFC0003001','TFC0003002','TFC0003003','TFC0003004
    ','TFC0003005','TFC0003009','TFC0003011','TFC0003012','
    TFC0003013','TFC0003014','TFC0003015','TFC0003016','
    TFC0003017','TFC0003018','TFC0003019','TFC0003020','
    TFC0003021','TFC0003022','TFC0003023','TFC0003026','
    TFC0003028','TFC0003031','TFC0003035','TFC0003037','
    TFC0003038','TFC0003039','TFC0003040','TFC0003041','
```

```
    TFC0003045','TFC0003046','TFC0003047','TFC0003052','
    TFC0003053','TFC0003054','TFC0003055','TFC0003056','
    TFC0003057','TFC0003059','TFC0003065','TFC0003066','
    TFC0003068','TFC0003069','TFC0003070','TFC0003071','
    TFC0003072','TFC0003073','TFC0003074','TFC0003075','
    TFC0003076','TFC0003077','TFC0003078','TFC0003081','
    TFC0003082','TFC0003083','TFC0003085','TFC0003087','
    TFC0003088','TFC0003090','TFC0003091','TFC0003092','
    TFC0003094','TFC0003103','TFC0003105','TFC0003106','
    TFC0003109','TFC0003111','TFC0003112','TFC0003114','
    TFC0003115','TFC0003117','TFC0003118','TFC0003119','
    TFC0003127','TFC0003139','TFC0003147','TFC0003151','
    TFC0003154','TFC0003155','TFC0003157','TFC0003158','
    TFC0003159','TFC0003161','TFC0003164','TFC0003165','
    TFC0003177','TFC0003178','TFC0003182','TFC0003184','
    TFC0003185','TFC0003186','TFC0003188','TFC0003190','
    TFC0003199','TFC0003201','TFC0003203','TFC0003204','
    TFC0003205','TFC0003207','TFC0003208','TFC0003210','
    TFC0003213','TFC0003215','TFC0003216','TFC0003219','
    TFC0003220','TFC0003221','TFC0003222','TFC0003226','
    TFC0003228','TFC0003233','TFC0003234','TFC0003235','
    TFC0003236','TFC0003237','TFC0003238','TFC0003239','
    TFC0003240','TFC0003242','TFC0003243','TFC0003247','
    TFC0003248','TFC0003249','TFC0003250','TFC0003252','
    TFC0003253','TFC0003254','TFC0003261','TFC0003263','
    TFC0003264','TFC0003267','TFC0003270','TFC0003272','
    TFC0003273','TFC0003277','TFC0003282','TFC0003286','
    TFC0003290','TFC0003291','TFC0003292','TFC0003293','
    TFC0003294','TFC0003298','TFC0003300','TFC0003301','
    TFC0003302','TFC0003303','TFC0003309','TFC0003315','
    TFC0003323','TFC0003324','TFC0003325','TFC0003328','
    TFC0003330','TFC0003336','TFC0003337','TFC0003338','
    TFC0003346','TFC0003351','TFC0003377','TFC0003380',
'TFC0003453','TFC0003454','TFC0003456','TFC0003457','TFC0003458
    ','TFC0003474','TFC0003487','TFC0003488','TFC0003496','
    TFC0003502','TFC0003503','TFC0003504','TFC0003505','
    TFC0003526','TFC0003537','TFC0003539','TFC0003541','
    TFC0003543','TFC0003553','TFC0003555','TFC0003557','
    TFC0003579','TFC0003594','TFC0003626','TFC0003629','
    TFC0003630','TFC0003631','TFC0003633','TFC0004002','
    TFC0004003','TFC0004004','TFC0004005','TFC0004006','
    TFC0004007','TFC0004008','TFC0004009','TFC0004010','
    TFC0004011','TFC0004012','TFC0004013','TFC0004014','
    TFC0004015','TFC0004041','TFC0004062','TFC0004104','
    TFC0004110','TFC0004187','TFC0004188','TFC0004193','
    TFC0004194','TFC0004195','TFC0004196','TFC0004197','
    TFC0004198','TFC0004199','TFC0004205','TFC0004211','
    TFC0004212','TFC0004213','TFC0004214','TFC0004215','
    TFC0004217','TFC0004218','TFC0004219','TFC0004220','
    TFC0004223','TFC0004224','TFC0004231','TFC0004233','
    TFC0004234','TFC0004235','TFC0004239','TFC0004241','
```

TFC0004242','TFC0004245','TFC0004248','TFC0004255','
TFC0004256','TFC0004257','TFC0004263','TFC0004264','
TFC0004265','TFC0004275','TFC0004297','TFC0004298','
TFC0004308','TFC0004309','TFC0004312','TFC0004313','
TFC0004322','TFC0004323','TFC0004358','TFC0004359','
TFC0004360','TFC0004363','TFC0004372','TFC0004375','
TFC0004379','TFC0004387','TFC0004398','TFC0004411','
TFC0004412','TFC0004413','TFC0004423','TFC0004424','
TFC0004425','TFC0004426','TFC0004427','TFC0004428','
TFC0004431','TFC0004432','TFC0004433','TFC0004435','
TFC0004436','TFC0004437','TFC0004438','TFC0004439','
TFC0004442','TFC0004443','TFC0004444','TFC0004445','
TFC0004446','TFC0004447','TFC0004448','TFC0004449','
TFC0004450','TFC0004454','TFC0004457','TFC0004468','
TFC0004469','TFC0004470','TFC0004471','TFC0004473','
TFC0004474','TFC0004483','TFC0004484','TFC0004485','
TFC0004486','TFC0004487','TFC0004488','TFC0004489','
TFC0004490','TFC0004491','TFC0004498','TFC0004499',
'TFC0004500','TFC0004501','TFC0004502','TFC0004503','TFC0004504
','TFC0004505','TFC0004506','TFC0004507','TFC0004508','
TFC0004509','TFC0004510','TFC0004511','TFC0004512','
TFC0004513','TFC0004514','TFC0004515','TFC0004516','
TFC0004517','TFC0004518','TFC0004519','TFC0004520','
TFC0004524','TFC0004528','TFC0004531','TFC0004532','
TFC0004533','TFC0004534','TFC0004536','TFC0004537','
TFC0004538','TFC0004539','TFC0004540','TFC0004541','
TFC0004542','TFC0004543','TFC0004544','TFC0004545','
TFC0004546','TFC0004551','TFC0004554','TFC0004555','
TFC0004556','TFC0004557','TFC0004558','TFC0004564','
TFC0004565','TFC0004567','TFC0004569','TFC0004572','
TFC0004573','TFC0004575','TFC0004576','TFC0004577','
TFC0004580','TFC0004587','TFC0004601','TFC0004602','
TFC0004603','TFC0004604','TFC0004606','TFC0004608','
TFC0004609','TFC0004611','TFC0004625','TFC0004643','
TFC0004644','TFC0004645','TFC0004646','TFC0004647','
TFC0004648','TFC0004649','TFC0004650','TFC0004651','
TFC0004652','TFC0004654','TFC0004655','TFC0004656','
TFC0004657','TFC0004658','TFC0004659','TFC0004662','
TFC0004663','TFC0004664','TFC0004665','TFC0004678','
TFC0004697','TFC0004698','TFC0004699','TFC0004700','
TFC0004868','TFC0005137','TFC0005204','TFC0005322','
TFC0005387','TFC0005549','TFC0005956','TFC0006260','
TFC0006267','TFC0006268','TFC0006270','TFC0006283','
TFC0006284','TFC0006286','TFC0006293','TFC0006296','
TFC0006331','TFC0006374','TFC0006379','TFC0006386','
TFC0006449','TFC0006467','TFC0006482','TFC0006534','
TFC0006552','TFC0006557','TFC0006566','TFC0006578','
TFC0006607','TFC0006627','TFC0006636','TFC0006641','
TFC0006643','TFC0006648','TFC0006650','TFC0006662','
TFC0006666','TFC0006668','TFC0006672','TFC0006678','
TFC0006679','TFC0006732','TFC0006735','TFC0006737','

TFC0006738', 'TFC0006740', 'TFC0006758', 'TFC0006759', '
TFC0006767', 'TFC0006798', 'TFC0006800', 'TFC0006802', '
TFC0006806', 'TFC0006813', 'TFC0006824', 'TFC0006917', '
TFC0006922', 'TFC0007105', 'TFC0007216', 'TFC0007217', '
TFC0007219', 'TFC0007543', 'TFC0007549', 'TFC0007589', '
TFC0007615', 'TFC0007660', 'TFC0007731', 'TFC0007742', '
TFC0007791', 'TFC0007860', 'TFC0007953', 'TFC0008215', '
TFC0008216', 'TFC0008223', 'TFC0008230', 'TFC0008245', '
TFC0008246', 'TFC0008334', 'TFC0008422', 'TFC0008499', '
TFC0008506', 'TFC0008551', 'TFC0008570', 'TFC0008594', '
TFC0008596', 'TFC0008597', 'TFC0008598', 'TFC0008599', '
TFC0008675', 'TFC0008799', 'TFC0008814', 'TFC0009008', '
TFC0009331', 'TFC0009490',
'WS03340001', 'WS03340003', 'WS03340004', 'WS03340005', 'WS03340006
', 'WS03340007', 'WS03340008', 'WS03340010', 'WS03340011', '
WS03340012', 'WS03340024', 'WS03810002', 'WS03810004', '
WS03810005', 'WS03810006', 'WS03810007', 'WS03810008', '
WS03810009', 'WS03810010', 'WS03810011', 'WS03810012', '
WS03810013', 'WS03810014', 'WS03810015', 'WS03810016', '
WS03810018', 'WS03810020', 'WS03810021', 'WS03810022', '
WS03810023', 'WS03810024', 'WS03810027', 'WS03810029', '
WS03810030', 'WS03840001', 'WS03840002', 'WS03840003', '
WS03840004', 'WS03840005', 'WS03840006', 'WS03840007', '
WS03840008', 'WS03840010', 'WS03840011', 'WS03840012', '
WS03840013', 'WS03840014', 'WS03840015', 'WS03840016', '
WS03840017', 'WS03840018', 'WS03840019', 'WS03840020', '
WS03840021', 'WS03840022', 'WS03840023', 'WS03840024', '
WS03840025', 'WS03840026', 'WS03840027', 'WS03840028', '
WS03840029', 'WS03840031', 'WS03840032', 'WS03840033', '
WS03840034', 'WS03840035', 'WS03840036', 'WS03840037', '
WS03840038', 'WS03840039', 'WS03840040', 'WS03840041', '
WS03860002', 'WS03860005', 'WS03860006', 'WS03860009', '
WS03860010', 'WS03860011', 'WS03860014', 'WS03860015', '
WS03860017', 'WS03860020', 'WS03860023', 'WS03860033', '
WS03860049', 'WS03860056', 'WS03860057', 'WS03860058', '
WS03870003', 'WS03870015', 'WS03870016', 'WS03870017', '
WS03870019', 'WS03870020', 'WS03870021', 'WS03870022', '
WS03870025', 'WS03870027', 'WS03870035', 'WS03870049', '
WS03880001', 'WS03880002', 'WS03880009', 'WS03880015', '
WS03880016', 'WS03880026', 'WS03900002', 'WS03900003', '
WS03900004', 'WS03900008', 'WS03900009', 'WS03900010', '
WS03900011', 'WS03900019', 'WS03900020', 'WS03900021', '
WS03900022', 'WS03900023', 'WS03900024', 'WS03900026', '
WS03900027', 'WS03900028', 'WS03900032', 'WS03900033', '
WS03900034', 'WS03900037', 'WS03900043', 'WS03900044', '
WS03900045', 'WS03900046', 'WS03900050', 'WS08510001', '
WS08510002', 'WS08510003', 'WS08510004', 'WS08510005', '
WS08510007', 'WS08510008', 'WS08510009', 'WS08510012', '
WS08510016', 'WS08510017', 'WS08510018', 'WS08510020', '
WS08510021', 'WS08510022', 'WS08510023', 'WS08510024', '
WS08510028', 'WS08510029', 'WS08510030', 'WS08510031', '

WS08510033', 'WS08510035', 'WS08510036', 'WS08510037', '
WS08510038', 'WS08510039', 'WS08510041', 'WS08510042', '
WS08510043', 'WS08510044', 'WS08510046', 'WS08510049', '
WS08530001', 'WS08530002', 'WS08530003', 'WS08530004', '
WS08530005', 'WS08530006', 'WS08530007', 'WS08530008', '
WS08530009', 'WS08530010', 'WS08530011', 'WS08530013', '
WS08530014', 'WS08530015', 'WS08530017', 'WS08530018', '
WS08530019', 'WS08530020', 'WS08530025', 'WS08530026', '
WS08530027', 'WS08530028', 'WS08530029', 'WS08530030', '
WS08530031', 'WS08530032',
'WS08540028', 'WS08540029', 'WS08910003', 'WS08910018', 'WS08910019
', 'WS08910022', 'WS08910023', 'WS08980001', 'WS08980002', '
WS08980003', 'WS08980004', 'WS08980009', 'WS08980010', '
WS08980011', 'WS08980012', 'WS08980015', 'WS08980017', '
WS08980018', 'WS08980019', 'WS08980020', 'WS08980021', '
WS08980022', 'WS08980023', 'WS08980024', 'WS08980025', '
WS08980027', 'WS08980028', 'WS08980029', 'WS08980030', '
WS08980031', 'WS08980032', 'ZA15C00036', 'ZA15C00039', '
ZA15C00040',
'HC27703185', 'HC27703227', 'HC27703266',
'RD00000167', 'RD00001586', 'RD00001611', 'RD00001909', 'RD00002088
', 'RD00002174', 'RD00002176', 'RD00002239', 'RD00002240', '
RD00002248', 'RD00002278', 'RD00002284', 'RD00002332', '
RD00002333', 'RD00002334', 'RD00002336', 'RD00002350', '
RD00002351', 'RD00002362', 'RD00002376', 'RD00002381', '
RD00002384', 'RD00002388', 'RD00002389', 'RD00002391', '
RD00002394', 'RD00002425', 'RD00002436', 'RD00002493', '
RD00002494', 'RD00002497', 'RD00002500',
'S00745', 'S00746', 'S00756', 'S00762', 'S00767', 'S00768', 'S00770',
'TFC0008212', 'TFC0008264', 'TFC0008293', 'TFC0008423', 'TFC0008430
', 'TFC0008437', 'TFC0008485', 'TFC0008543', 'TFC0008579', '
TFC0008581', 'TFC0008592', 'TFC0008595', 'TFC0008622', '
TFC0008636', 'TFC0008728', 'TFC0008730', 'TFC0008735', '
TFC0008857', 'TFC0008904', 'TFC0008910', 'TFC0008933', '
TFC0008955', 'TFC0008973', 'TFC0008981', 'TFC0008982', '
TFC0008984', 'TFC0008985', 'TFC0008988', 'TFC0008989', '
TFC0008990', 'TFC0008991', 'TFC0008992', 'TFC0008995', '
TFC0008996', 'TFC0008998', 'TFC0009209', 'TFC0009214', '
TFC0009222', 'TFC0009249', 'TFC0009257', 'TFC0009259', '
TFC0009263', 'TFC0009269', 'TFC0009271', 'TFC0009273', '
TFC0009274', 'TFC0009275', 'TFC0009287', 'TFC0009290', '
TFC0009291', 'TFC0009296', 'TFC0009301', 'TFC0009302', '
TFC0009304', 'TFC0009308', 'TFC0009353', 'TFC0009357', '
TFC0009453', 'TFC0009465', 'ALWQL00579', 'FD00000002', '
FD00000004', 'FD00000007', 'FD00000011', 'FD00000012', '
FD00000024', 'FD00000036', 'FD00000040', 'FD00000055', '
FD00000060', 'FD00000082', 'FD00000091', 'RD00000267', '
RD00000285', 'RD00000326', 'RD00001433', 'RD00001605', '
RD00001613', 'RD00001821', 'RD00001830', 'RD00001881', '
RD00001882', 'RD00001887', 'RD00002049', 'RD00002051', '
RD00002084', 'RD00002085', 'RD00002092', 'RD00002093', '

```
                    RD00002094','RD00002101','RD00002299','RD00002302','
                    RD00002303','RD00002307','RD00002315','RD00002316','S00744
                    ','S00747','S00752','S00755','S00757','S00769','S00773','
                    S00775','TFC0006274','TFC0006276','TFC0006277','TFC0006633
                    ','TFC0006856','TFC0006874','TFC0006875','TFC0006877','
                    TFC0006989','TFC0007003','TFC0007004','TFC0007005','
                    TFC0007006','TFC0007007','TFC0007009','TFC0007010','
                    TFC0007011','TFC0007012','TFC0007015','TFC0007031','
                    TFC0007086','TFC0007097','TFC0007103','TFC0007104','
                    TFC0007106','TFC0007107','TFC0007109','TFC0007110','
                    TFC0007111','TFC0007130','TFC0007131','TFC0007132','
                    TFC0007134','TFC0007135','TFC0007140','TFC0007141','
                    TFC0007159','TFC0007162','TFC0007163','TFC0007165','
                    TFC0007166','TFC0007167','TFC0007168','TFC0007169','
                    TFC0007170','TFC0007173','TFC0007179','TFC0007180','
                    TFC0007181','TFC0007182','TFC0007251','TFC0007351','
                    TFC0008389','TFC0008426','TFC0008856','TFC0008924','
                    TFC0008926','TFC0008928','TFC0008931','TFC0008932','
                    TFC0008934','TFC0008935','TFC0008936','TFC0008983','
                    TFC0008986','TFC0008987','TFC0008993','TFC0008994','
                    TFC0008999','TFC0009035','TFC0009036','TFC0009248','
                    TFC0009258','TFC0009260','TFC0009261','TFC0009264','
                    TFC0009265','TFC0009266','TFC0009267','TFC0009268','
                    TFC0009270','TFC0009272',
                'TFC0009303','TFC0009305','TFC0009306','TFC0009310','TFC0009311
                    ','TFC0009351','TFC0009352','TFC0009452','TFC0009454'
                ]
        return swf_a
#enddef
def get_swf_b():
        swf_b    = ['FC11BOX19','FCLEBOX35','FT14005104','HC27001819','HC27001820','
            HT13007556']
        return swf_b
def get_swf_s():
    swf_s_nr0 = [
                'NR367442','NR316657','NR316713','NR343991','NR367443','NR367444',
                'NR367445','NR367450','NR343903','NR343902','NR341646','NR367537',
                'NR341638','NR344385','NR367381','NR367382','NR367383','NR367384',
                'NR367386','NR367446','NR367447','NR344597','NR316709','NR316710',
                'NR316711','NR316712','NR316744','NR396950','NR367449','NR316708',
                'NR316410','NR316659','NR343918','NR343953','NR344375','NR367536',
                'NR316408','NR316406','NR316407','NR316409','NR316405'
                ]
    swf_s_nrx = [
                'KNO02401','KNO02402','KNO02403','KNO02404','KNO02405','KNO02406',
                'KNO02407','KNO02408','KNO02501','KNO02502','KNO02503','KNO02504',
                'KNO02505','KNO02506','KNO02507','KNO02508','KNO02701','KNO02702',
                'KNO02703','KNO02704','KNO02705','KNO02706','KNO02707','KNO02708',
                'KNO02801','KNO02802','KNO02803','KNO02804','KNO02805','KNO02806',
                'KNO02807','KNO02808'
                ]
```

```
    swf_s_nr1 = [
                'BNNSCP01','BNNSCP02','BNNSCP03','BNNSCP04','BNNSCP05','BNNSCP06','
                    BNNSCP07','BNNSCP08',
                'BNNSICDC01','BNNSICDC02','BNNSICDC03','BNNSICDC04',
                'BNNSICDC13','BNNSICDC14','BNNSICDC15','BNNSICDC16','BNNSICDC17','
                    BNNSICDC18','BNNSICDC19',
                'BNNSICDC20','BNNSICDC21','BNNSICDC22','BNNSICDC23','BNNSICDC24','
                    BNNSICDC25','BNNSICDC26','BNNSICDC27','BNNSICDC28',
                'KAL45801','KAL49401','KAL5221','KAL62401','KAL65601','KAL66201','
                    KAL66301','KAL70401','KAL72301','KAL72701','KAL82101','KAL83001
                    ','KSO04001'
                ]
    swf_s = []
    swf_s.extend(swf_s_nr0)
    swf_s.extend(swf_s_nr1)
    return swf_s
def get_swf_h():
    swf_h     = ['AC10001497']  #AG-108M is a mistake, should be AG-108M-H
    return swf_h

def fix_iso(iso, swf):
    if iso[-2] == "-" and iso[-1].isalpha():
        pass #iso is swf don't replace it iso = iso[:-2]  #if iso is swf, replace it
            with the new SWF
    else:
        iso = iso + swf
    #endif

    return iso
#enddef

#-----------------------------------------------------------------------
#main code for best-estimate
#-----------------------------------------------------------------------
def get_isod(type, loc, ddir, unctype):

    lisos     = db_info.read_dump(ddir,'dump_tmp_iso.txt')
    dws       = db_info.read_dump(ddir,'dump_tmp_ws.txt')          #{"00001-LLW_v.0":
        ["SWEWSCF19950007_rev.0", {"0": {"H-3": 1.0}}],..
    dcuts     = db_info.read_dump(ddir,'dump_tmp_cuts.txt')        #{"00002": [["1",
        "00621-LLW", 0, 0, "DTC", null]],...
    dcntrs    = db_info.read_dump(ddir,'dump_tmp_outer_'+loc.lower()+'.txt') #maybe
        need to update to have list of isotopes in here, so can skip, but maybe not
    dcntract = db_info.read_dump(ddir,'dump_tmp_cntnr_act.txt')  #{"235F000003": {"
        NI-59": 1.071590863e-13, "H-3": 2.87500413476e-11,....},....

    dbdir     = db_info.get_db_subdir()                    #db
    cdir      = os.path.join(dbdir,'containers')

    #check that code is working (inside container isotope sum = outside container
        isotope sum)
```

```
if type == 'test':
    tdir = db_info.get_data_subdir('test_base','output')
else:
    tdir = db_info.get_data_subdir('base','output')
#endif
tddir       = os.path.join(tdir,'dump')
base_file   = os.path.join(tddir,'dump_'+loc.lower() + '.txt')

basedcntnrs      = {}
basedcntnrs[loc] = {}

with open(base_file) as dfileo:
    #can't do it this way b/c of memoryError
    #   dump_obj = json.load(dfileo)
    #print(dump_obj)
    for line in dfileo:
        aline      = line.split(',',1)
        iso        = aline[0]
        basedcntnrs[loc]={iso:[]}

        l_packages = json.loads(aline[1])
        if l_packages == []: #no packages with this isotope in this DU
            continue
        else:
            basedcntnrs[loc][iso]=l_packages
        #endif
    #endfor
#endwith

#print("Problems in WITS - come check out line 1037 - 1060 in zmake_dump")
#here's a BAD THING TO DO - everything should work, so why aren't these? Likely
   b/c defect in WITS
#'CBALX5575','CBALX5585','CBALX5587' are in ETRENCH2 the elem_sub is not in the
   waste_cut dict for them

#I think this has something to do with these being CHAR BY PACK or maybe b/c PCB
    waste and elemental carbon,
#but could be a bug (b/c the 02297 and 02319 are DTC and are missing PCT_DIST_CI
    in WITS_TBL_WS_ELEMENT
#'AL07N90648' is a package with DTC on inner containers and outer - not sure how
    that works so commenting out for now

#'SWD051276','ST17215720','RD00009948','RD00009983','RD00010012','RD00010015'
   use the wrong waste_stream_version to get PCT_DIST_CI_NORM to calc the
   activity
#'02297','02319' are in ETRENCH3 (I think this is same bug as above)

#'188845','A036','A089','190684' ETRENCH1 188845 has element derivative U-235-D
   not U-235 in container and U-236 shows up in nuclide distribution, but not
   in query - ST17906453
#   since these seem only to be issues with U-23X, perhaps I'm doing the query
```

```
    wrong

#'HT13010061','HT13010087' there are more than one waste stream per DTC - I
    think I could figure this out but skipping for now
#
bad_list = ['CBALX5575','CBALX5585','CBALX5587','CBALX21021','CBALX21140',
            '02297','02319',
            'AL07N90648',
            'SWD051276','ST17215720','RD00009948','RD00009983','RD00010012','
               RD00010015',
            '188845','A036','A089','190684',
            'HT13010061','HT13010087',
            ]

swf_a = get_swf_a()
swf_b = get_swf_b()
swf_s = get_swf_s()
swf_h = get_swf_h()

#GOING TO HAVE TO DO SOMETHING ABOUT CIG1 (ST23) - all containers deposited
    prior to 6/8/2021 are considered CIG (swf_a)
#so either going to have to :
#   get list of these containers
#   get these containers in WITS updated with the correct SWF
#       and include the waste package storage location history date in the dump
    file

#initialize the dictionary with an empty list to start
isod = {}
for iso in lisos:
    isod[iso] = []
#endfor

for iso in lisos:
    #print('=====')
    #print(iso)
    isoact    = 0
    isobias   = 0
    #          cinfo = [opack, total_act, inner_cntnrs]
    for ocnum, cinfo  in dcntrs.items():  #dump_tmp_outer_LOC
        #if ocnum == 'RD009427' and iso == 'I-129':
            #print(ocnum, iso)
            #input("hit enter")
        if ocnum in bad_list: #skip over these containers (see line 1052)
            continue
        #endif

        isodict = dcntract.get(ocnum)   #{ocnum:{iso1:act1,iso2:act2,...} #file:
            dump_tmp_cntnr_act.txt
        if isodict is None:
            print(ocnum,'not in dcntract - try rerunning without --nosql
```

```
        dump_tmp_cntnr_act.txt is incomplete')
    sys.exit()
#endif

iso_act = isodict.get(iso, None) #get this isotopes activity in this
    container, return None if not in container
if iso_act is None:
    #print('iso not in this container')
    pass
else:
    iso_org = iso
    #what about iso-swf-swf case (in CIG1....hmm)
    msg = ocnum + " " + str(iso_act) + " " + iso + " "
    take_action = False

    if ocnum in swf_a:
        if iso[2:5] in ["233","235"] and iso[-1] == "D": #turn U-233-D
            into U-233-E #also U-235-D (U233 doesn't exist, currently)
            iso = fix_iso(iso, "-E")
            take_action = True
        elif iso[-1] != "A":
            iso = fix_iso(iso, "-A")
            take_action = True
        #endif
    elif ocnum in swf_b:
        if iso[2:5] in ["233","235"] and iso[-1] == "D": #turn U-233-D
            into U-233-E #also U-235-D (U233 doesn't exist, currently)
            iso = fix_iso(iso, "-E")
            take_action = True
        elif iso[-1] != "B":
            iso = fix_iso(iso, "-B")
            take_action = True
        #endif
    elif ocnum in swf_s and iso[-1] != "S":
        iso = fix_iso(iso, "-S")
        take_action = True
    elif ocnum in swf_h and iso == 'AG-108M':
        iso = fix_iso(iso, "-H")
        take_action = True
    #endif

    if take_action == True:
        if iso in isod:
            pass
        else:
            print('adding ', iso, 'to isod')
            isod[iso] = []
        #endif
        msg = msg + iso
        #print(msg)
        #print(isod)
```

```
                #take_action = False
            #endif
            #print(ocnum)
            total_act_true, l_act_unc, tact = get_bias(ocnum, cinfo, dcuts, iso,
                dws, iso_act, dcntract)  #MAIN WORK HERE TO GET BIAS AND UNC
            #print("----------")
            isod[iso].append((ocnum, l_act_unc, tact, total_act_true))  #make
               the isod, include tact so don't have to loop twice
            isobias = isobias + total_act_true
            isoact  = isoact  + iso_act

            #Check that inside = outside
            #iso_act = tact THIS SHOULD BE TRUE (iso_act is activity in the
               outer container, tact is from summing cuts and inner containers)
            if not math.isclose(tact, iso_act, rel_tol=0.05): #within 5%
                if take_action == True: #skip this for those special isotopes
                    pass
                else:
                    for lcntr in basedcntnrs[loc][iso]:
                        if lcntr[0] == ocnum:
                            msg =       "O_Cnum base_act iso_act tact base_bias
                               total act true \n"
                            msg = msg + str(ocnum) + ' ' + str(lcntr[2]) + ' ' +
                                str(iso_act)+ ' ' + str(tact)+ ' ' + str(lcntr
                               [4])+ ' ' + str(total_act_true) + '\n'
                            msg = msg + 'iso_act and tact values are not close\n
                               '
                            print(msg)
                            #input(msg)
                    #endfor
                #endif
            #endif
            iso = iso_org
        #endif
    #endfor (loop over ocnum in loc)

    """
    if iso == 'CS-137':
        print(isod['CS-137-A'])
        input("x")

    if iso == 'I-129':
        print(isod['I-129-A'])
        print(isod['I-129'])
        input("X")
    """
#endfor these isos

"""
print("---")
#print(isod['CS-137-A'])
```

```
    #print(isod['CS-137'])
    #print(isod['H-3-A'])
    #print(isod['H-3'])
    print(isod['I-129-A'])
    print(isod['I-129'])
    print("---")
    print(len(isod))  #456 / 372
    """

    return isod
#enddef

#-------------------------------------------------------------------------------
#working code to process locations
#-------------------------------------------------------------------------------
def monoprocess(loc,type,nosql,dddir,tdir,unctype,counter):
    db_info.set_seed.counter = counter

    print('==================================')
    print(loc)
    print('----------------------------------')

    iso_act = {}
    isod    = {}

    #=================================
    #Base Case is a FAST WAY to get FAKE UNCERTAINTIES and BIASES
    #=================================
    if type in ['base','test_base']:
        #get database connection (even nosql still might need it)
        conn = db_info.get_conn(type)
        isod = get_containers2(conn, loc, nosql, dddir, 'uniform')
        conn.close()
    #=================================
    #Best (Effort) Case is A way to get some REAL and some FAKE UNCERTAINTIES and
       BIASES
    #=================================
    elif type in ['best','test']:
        isod = get_isod(type, loc, dddir, 'uniform')
    #endif
    #iso_act[loc] = isod

    #=================================
    #Make the dump file - used by zppp and zanalysis to get results
    #=================================
    dfile = os.path.join(os.path.join(tdir,'dump'),'dump_'+loc.lower()+'.txt')
    """
    with open(dfile,'w') as dfileo:
        if type == 'test':
            json.dump(isod,dfileo,indent=4)
        else:
```

```
            json.dump(isod,dfileo)
        #endif
    #endwith

    """
    with open(dfile,'w') as dfileo:
        """
        if type == 'test' and loc != 'CIG1': #only for test and CIG1 b/c it's small
            enough not to have memory crash in zanalysis
            lines = ''
            for iso, l_pack in isod.items():
                str_l_pack = json.dumps(l_pack,indent=4)
                dfileo.write(iso + "," + str_l_pack + "\n")
            #endfor
        else: #have to do it this way for most locs because of memory crash in
            zanalysis
            lines = ''
            for iso, l_pack in isod.items():
                str_l_pack = json.dumps(l_pack)
                dfileo.write(iso + "," + str_l_pack + "\n")
            #endfor
        #endif
        """
        lines = ''
        for iso, l_pack in isod.items():
            str_l_pack = json.dumps(l_pack)
            dfileo.write(iso + "," + str_l_pack + "\n")
        #endfor
    #endwith

#enddef

#-----------------------------------------------------------------------------
#prepare locations for multiprocessing
#-----------------------------------------------------------------------------
def multiprocess(locs, type, nosql,dddir,tdir,random_type,random_seed):
    #make the iterable for each worker in the pool
    iters = []
    for loc in locs:
        iters.append((loc,type,nosql,dddir,tdir,random_type,random_seed))
    #endfor

    #REAL WORK STARTS HERE (in the "work" function)
    #Run each location in parallel
    pool = multiprocessing.Pool(None)
    pool.starmap(monoprocess,iters)
#enddef

#-----------------------------------------------------------------------------
#Main code to create the dump file, which is a dictionary of each location:
#of the isotopes in that location, each isotope has a list of:
```

```
#    the outermost containers (for base)
#    a list of cuts in a list of containers (for best)
#with that isotope and the uncertainty and the bias attributed to that container
#-------------------------------------------------------------------------------
def main(args):
    xnow    = datetime.datetime.today().date()
    xfuture = datetime.date(2021,10,20)
    if xnow > xfuture:
        msg = "It's the future.\n"
        msg = msg + "Come check:\n"
        msg = msg + "db_info.get_future_locs\n"
        msg = msg + "unc_aux.get_pa_locs\n"
        msg = msg + "\n"
        msg = msg + "msg generated in zmake_dump.main()\n"
        print(msg)
        return
    #endif

    type  = args.Type.lower()
    loc   = args.Location.upper()
    nosql = args.nosql

    time_start = time.time()

    #get the type of analysis dir (base/best/test/test_base)
    tdir   = db_info.get_data_subdir(type,'output')  #base OR best_0, best_1, ... OR
        test
    dbdir  = db_info.get_db_subdir()                 #db
    dddir  = os.path.join(dbdir,'dump_'+ str(type))  #db/dump_base, db/dump_best, db
       /dump_test

    #get the PA locations
    if nosql:
        palocs = db_info.get_dump_palocs(type)
        if palocs == "":
            print("run code without the '--nosql' option")
            return
        #endif
    else:
        conn   = db_info.get_conn(type)        #get database connection
        palocs = unc_aux.get_pa_locs(conn) #REAL WORK HERE
        dfile  = os.path.join(dddir,'dump_pa_locs.txt')
        with open(dfile,'w') as dfileo:
            json.dump(palocs, dfileo)
        #endwith
        conn.close()
    #endif

    if loc == 'ALL':
        locs = palocs
    elif loc in palocs:
```

```
        locs = [loc]
    else:
        print(loc,"not a PA location")
        return
    #endif

    #check dumpfiles exist or make them if not (used in best)
    if type in ['base','test_base']:
        pass
    else: #type is best or test
        make_isotopes(nosql, type, dddir)
        make_wastestreams(nosql, type, dddir)
        make_cntnrisoact(nosql, type, dddir)
        make_inner_cntnrs(nosql, type, dddir)
        make_cuts(nosql, type, dddir)
        for loc in locs:
            make_du_outer_cntnrs(loc, nosql, type, dddir)
        #endfor
    #endif

    if len(locs)==1:
        monoprocess(locs[0],type,nosql,dddir,tdir,'uniform',0)
    else: #fast
        multiprocess(locs,type,nosql,dddir,tdir,'uniform',0)
    #endif
    print('done')
    time_stop = time.time()
    print_time(time_start, time_stop)
#enddef


#-----------------------------------------------------------------------
#create the uncertainty for the E-area PA and dump it out
#-----------------------------------------------------------------------
if __name__ == '__main__':
    arg_parser = argparse.ArgumentParser(description='Make the dump file of the WITS
        database (--nosql fast: takes 5 mins)')
    msg0 = 'the TYPE of dump file to create (base|best|test):'
    msg1 = '    base is the base case - estimated uncertainty, '
    msg2 = '    best is the best case - use uncertainty'
    msg = msg0+msg1+msg2
    arg_parser.add_argument('Type',
                            metavar='type',
                            type=str,
                            choices=['base','best','test','test_base'],
                            help=msg)
    arg_parser.add_argument('Location',
                            metavar='loc',
                            type=str,
                            help='the LOCATION to dump (all|NR0|CIG1|...)')
    arg_parser.add_argument('--nosql',
```

```
                                action="store_true",
                                help='read files vs hit database')
    args = arg_parser.parse_args()

    db_info.set_seed.counter = 0

    main(args)
#endif
```

Listing 2: zanalysis

```
import argparse
import gc
import glob
import json

import matplotlib.pyplot as plt  #<-slow
import math
import multiprocessing
import numpy as np  #<-slow
import os
import pandas
import re

import scipy.stats as ss
import shutil
import sys
import time
import zipfile

import db_info
import unc_aux
import zapp2
import zisochk
import zstats

#=============================================================================
def chk_dir(dir):
    if os.path.isdir(dir):
        pass
    else:
        os.mkdir(dir)
    #endif
#enddef


#do the panda read excel here
def get_path_isos(path, du):
    ifile = os.path.join(os.getcwd(),"zisochk.xlsx")
    df = pandas.read_excel(ifile,path,header=0)
```

```
    val  = df.loc[df[du.upper()]=='x','formatted']
    isos = val.tolist()
    return isos
#enddef

#-------------------------------------------------------------------------------
#This is similar to zisochk, but actually makes the missing files
#uses the avget and avgst values (or a bias_factor=1 and 2sig unc of 10%)
#-------------------------------------------------------------------------------
def do_missing(locs, type, tdir):
    #do missing in the following dirs
    dirs_to_do = ['bias','cdf']
    pathways   = ['gw','ihi','air','radon']

    if type == "test":
        return

    m = 0
    for dir in dirs_to_do:
        print("checking", dir, "for missing")
        for wloc in locs:
            print("checking", wloc)
            loc = db_info.translate_wits_to_pa(wloc)
            for path in pathways:
                isos = get_path_isos(path, loc)

                for iso in isos:
                    #print(tdir, dir, loc, iso)
                    f = os.path.join(os.path.join(tdir, dir), dir + '_' + loc + '_'
                       + iso + '.txt')
                    if os.path.exists(f):
                        pass               #file exists - great! do nothing
                    else:
                        print(dir, iso, "missing from", loc)
                        if dir == 'bias':
                            make_bias_aux(f, 1)
                        elif dir == 'cdf':
                            s = np.random.normal(0, 0.1, 1)
                            act         = 1e-12
                            mu          = act + act * s[0]  #(1 pCi +/- 10%)
                            rel_sig     = 0.1               #10%
                            sig         = rel_sig * mu      #absolute uncertainty

                            x_val, pdf_val, cdf_val = make_dist(mu, sig)
                            x_val_fraction = x_val/mu

                            make_cdf_aux(f,x_val_fraction,cdf_val)

                            sfile = os.path.join(os.path.join(tdir, 'stat'), 'stat_'
                                + loc + '_' + iso + '.txt')
                            make_stat_aux(sfile, mu, sig)
```

```
                    #endif
                #endif
            #endfor
        #endfor
    #endfor
#enddef

def get_avg_future(loc):
    if loc == 'st10':
        aloc = 'st09'
    elif loc in ['st11', 'st17','st18','st19','st20','st21','st22','st23','st24']:
        aloc = 'avgst'
    elif 'st' in loc:
        print(loc, "which one is this?")
        aloc = 'avgst'
    elif 'et' in loc:
        aloc = 'avget'
    else:
        print(loc, 'not ET or ST')
    #endif
    return aloc
#endif

def make_future_files(type, tdir, args):
    flocs = db_info.get_future_locs(type)
    #print(flocs)

    for oloc in flocs:
        print(oloc)
        loc  = db_info.translate_wits_to_pa(oloc)
        aloc = get_avg_future(loc)

        for pre in ['bias','stat','cdf']:
            pre_dir = os.path.join(tdir,pre)
            isos    = get_isos(pre_dir, pre)
            for iso in isos:
                xfile    = os.path.join(pre_dir, pre+'_' + aloc + '_'+iso+'.txt')
                if os.path.isfile(xfile):
                    new_xfile = os.path.join(pre_dir, pre+'_' + loc + '_'+iso+'.txt
                        ')
                    shutil.copy2(xfile, new_xfile)
                #endif
            #endfor
        #endfor
    #endfor
#enddef


#-------------------------------------------------------
def make_du_iso(b):
```

```
    x   = b.replace('.txt','')
    reg = r'.*_.*_'
    iso = re.sub(reg,'',x)
    iso = iso.capitalize()
    if iso[-2] == '-' and iso[-1].isalpha():
        swf = iso[-1].upper()
        iso = iso[:-2]
        iso = iso + '_' + swf
        #print(iso)
    #endif
    return iso
#enddef

def b_to_sfile(b, bias_dir, stat_dir):
    sfile = b.replace(bias_dir,stat_dir)
    sfile = sfile.replace('bias_','stat_')
    return sfile

def make_du_all_stats_files(du_file, bias_dir, stat_dir, loc):
    bfile   = os.path.join(bias_dir, 'bias_' + loc +'_*.txt')
    l_files = sorted(glob.glob(bfile))

    lines = ''
    for b in l_files:
        iso = make_du_iso(b)

        delta_bias = db_info.read_bfile(b)
        sfile      = b_to_sfile(b, bias_dir, stat_dir)
        mu, sig    = db_info.read_statfile(sfile)

        line       = iso + ',' + str(delta_bias)+ ',' + str(mu)+ ',' + str(sig) + '\
           n'
        lines      = lines + line
    #endfor

    with open(du_file,'w') as fh:
        fh.write(lines)
    #endwith
#enddef

def make_du_bias_files(du_file, bias_dir, loc):
    bfile   = os.path.join(bias_dir,'bias_' + loc +'_*.txt')
    l_files = sorted(glob.glob(bfile))

    lines = ''
    for b in l_files:
        iso = make_du_iso(b)

        delta_bias = db_info.read_bfile(b)
        line       = iso + ',' + str(delta_bias) + '\n'
        lines      = lines + line
```

```
    #endfor

    with open(du_file,'w') as fh:
        fh.write(lines)
    #endwith
#enddef


def make_du_cdf(du_dir, cdf_dir, loc):
    chk_dir(du_dir)

    pre     = 'cdf_' + loc
    org_c   = os.path.join(cdf_dir, pre+'_*.txt')
    l_files = sorted(glob.glob(org_c))

    for old_cdf in l_files:
        iso     = make_du_iso(old_cdf)
        new_cdf = os.path.join(du_dir,iso + '.txt')
        shutil.copy2(old_cdf, new_cdf)
    #endfor
#enddef

#-------------------------------------------------------------
#reformat directories and files so Sebastian's code can connect and read
#-------------------------------------------------------------
def make_ca_layout(tdir, all_locs, isos, args):

    #make new bias and cdf dirs
    bias_dir     = os.path.join(tdir,'bias')
    new_bias_dir = os.path.join(tdir,'bias_ca')

    cdf_dir      = os.path.join(tdir,'cdf')
    new_cdf_dir  = os.path.join(tdir,'cdf_ca')

    stat_dir     = os.path.join(tdir,'stat')
    #new_stat_dir = os.path.join(tdir,'stat_ca')

    all_stats_dir = os.path.join(tdir,'all_stats_ca')

    for d in [new_bias_dir,new_cdf_dir,all_stats_dir]:
        if os.path.exists(d):
            pass
        else:
            os.mkdir(d)
        #endif
    #endfor

    for oloc in all_locs:
        loc = db_info.translate_wits_to_pa(oloc).upper()

        #for doing the all the stats
```

```
        du_file = os.path.join(all_stats_dir,loc+'.txt')
        make_du_all_stats_files(du_file, bias_dir, stat_dir, loc.lower())

        #for doing the bias (not really needed now)
        du_file = os.path.join(new_bias_dir,loc+'.txt')
        make_du_bias_files(du_file, bias_dir, loc.lower())

        #doing the CDF (not really needed now)
        du_dir = os.path.join(new_cdf_dir,loc)
        make_du_cdf(du_dir, cdf_dir, loc.lower())
    #endfor
#enddef


#-------------------------------------------------------------------------
#get list of files in dir of du_type
#-------------------------------------------------------------------------
def get_lfiles(pre, du_type, iso, tdir):
    l_files = []
    sfile   = pre + '_' + du_type + '*' + '_' + iso + '.txt'
    ofile   = os.path.join(os.path.join(tdir,pre),sfile)
    l_files = glob.glob(ofile)
    return l_files
#enddef

#-------------------------------------------------------------------------
#real work (pre is 'stat') for making the average activities and uncertainties,
#used for future - current thinking is "worst" is best (most conservative)
#was based on ELLWF_Composition_Vectors_Rev-1
#-------------------------------------------------------------------------
def make_avg_out_aux(loc, iso, tdir, pre, args):
    nofig   = args.nofig
    to_show = args.show
    avg_type= args.avg

    #set up the figure here (one time), replot each time, otherwise run out of
        memory
    if nofig:
        pass
    else:
        fig = plt.figure()
    #endif

    #print(iso)
    for du_type in ('et','st','test'):
        #print(du_type)

        #get list of files that match DU type (eg ET* = ET01, ET02, ET03)
        l_stat_files = get_lfiles(pre, du_type, iso, tdir)

        l_du_stats = []
```

```
for f in l_stat_files:
    afile = f.split('_')
    du    = afile[-2]          #second from last value in file string (may
       have _ in dir name, ie earea_unc)

    #values from ELLWF_Composition_Vectors_Rev-1.xlsx sheet CVEC
    avg_factor = 1
    sof_total  = 1
    if du in ['et01','st01','st02','st03','st04','st05']:
        avg_factor = 1
        sof_total  = 1
    elif du in ['et02', 'et03','st06','st07','st08','st09','st14']:
        avg_factor = 0.95
        if   du == 'et02':
            sof_total = 0.751
        elif du == 'et03':
            sof_total = 0.352
        elif du == 'st06':
            sof_total = 0.143
        elif du == 'st07':
            sof_total = 0.456
        elif du == 'st08':
            sof_total = 0.370
        elif du == 'st09':
            sof_total = 0.936
        elif du == 'st14':
            sof_total = 0.943
       #endif
    elif du in ['test']:
        print("come check make_avg_out_aux")
        avg_factor = 1
        sof_total  = 1
    else:
        continue
    #endif

    #Do not adjust mu and sigma based on future
    #avg_factor = 1
    #sof_total  = 1

    bfile       = os.path.join(os.path.join(tdir,'bias'),'bias_'+du+'_'+iso
       +'.txt')
    bias_factor = db_info.read_bfile(bfile)

    with open(f,'r') as fh:
        a_lines = fh.readlines()
        floats = [float(x) for x in a_lines]
        mu = floats[0] * avg_factor/sof_total  #I_WITS_now  (see Excel file
           for how calc is done)
        sig= floats[1] * avg_factor/sof_total  #abs sig
        tru= mu * bias_factor
```

```
        #print(du_type,du,iso,mu,sig,tru)

        l_du_stats.append((mu,sig,tru,sig/mu,tru/mu))  #see excel file (
            columns C and K)

    #endwith
#endfor

if l_du_stats == []:
    pass
else:
    a = np.array(l_du_stats)

    if avg_type == 'wav':           #weighted average (don't do this)
        #if iso in ['h-3','sr-90']:
        #    print(a)
        aweights = 1/(a[:,1]**2)    #weights = 1/(sig^2)
        avg_mu   = np.average(a[:,0], weights=aweights)
        avg_tru  = np.average(a[:,2], weights=aweights)
        #avg_sig  = 1/np.sqrt(np.sum(a[:,1]))
        avg_sig  = 1/np.sqrt(np.sum(aweights))
    elif avg_type == 'worst-org':  #use the values with the maximum frac
        uncertainty and no bias
        max_sig = np.amax(a[:,3])
        vloc    = np.where(a[:,3] == max_sig)
        i       = vloc[0][0]   #index
        avg_mu  = a[i][0]      #
        avg_tru = avg_mu       #set true value = now value (aka unbiased =
            del_bias = 1)
        avg_sig = a[i][1]
    elif avg_type == 'worst':  #use the values with the maximum frac
        uncertainty and min bias (worst 'reasonable' values)
        max_sig = np.amax(a[:,3])  #relative sigma
        avg_mu  = np.mean(a[:,0])  #average activity using composition
            vectors
        avg_sig = avg_mu * max_sig #convert rel to absolute sigma

        min_del_bias = a[np.argmin(np.abs(a[:,4]-1))][4]  #find the del_bias
            closest to 1 and then return that array and del_bias
        avg_tru      = avg_mu * min_del_bias #set true value = min_del_bias
            (aka closest to 1)

    else:                          #ordinary average - see Excel file
        avg_mu     = np.mean(a[:,0])
        avg_tru    = np.mean(a[:,2])
        if len(a)  > 1:  #more than one DU with this isotope, get the sample
            standard deviation of the mu's as the uncertainty
            avg_sig = np.std(a[:,0], ddof=1)
        else:            #just this DU has this isotope, use the sigma of
            this DU.
            avg_sig = a[0,1]
```

```
    #endif
#endif

delta_bias = avg_tru/avg_mu

#if du_type =='et' and iso=='pu-241':
#print(l_du_stats,du_type,loc,iso, avg_mu, avg_tru, avg_sig) #, a[:,1])

#avg files
avgsfile = pre    + '_avg' + du_type + '_' + iso + '.txt'
avgbfile = 'bias' + '_avg' + du_type + '_' + iso + '.txt'

#make avg bias
bdir  = os.path.join(tdir,'bias')
bfile = os.path.join(bdir, avgbfile)
make_bias_aux(bfile,delta_bias)

#make avg stat file
sdir  = os.path.join(tdir,'stat')
sfile = os.path.join(sdir, avgsfile)
make_stat_aux(sfile, avg_mu, avg_sig)

#get info for PDF and CDF
#print(iso, avg_mu, avg_sig)
x_val, pdf_val, cdf_val = make_dist(avg_mu, avg_sig)
x_val_fraction          = x_val/avg_mu

#make PDF
pdir    = os.path.join(tdir,'pdf')
pfile   = os.path.join(pdir,'pdf_avg' + du_type + '_' + iso + '.txt')
make_pdf_aux(pfile,x_val_fraction,pdf_val)

#get the info for the CDF
cdir    = os.path.join(tdir,'cdf')
cfile   = os.path.join(cdir,'cdf_avg' + du_type + '_' + iso + '.txt')
make_cdf_aux(cfile,x_val_fraction,cdf_val)

#plot the data
if nofig:
    pass
else:
    du_per_unc = avg_sig/avg_mu*100

    if loc.upper() == 'ALL':
        loc2 = 'AVG ' + du_type
    else:
        loc2 = loc
    #endif

    fig, ax1, ax2 = plot_fig(fig, loc2, iso, avg_mu, delta_bias,
       du_per_unc, x_val_fraction, pdf_val, cdf_val)
```

```
                fdir  = os.path.join(tdir,'figs')
                ffile = os.path.join(fdir,'fig_avg'+du_type+'_'+iso+'.png')
                make_figure_aux(ffile,fig)

                if to_show:
                    win = plt.gcf().canvas.manager.window                    #disable
                        the close button (use the ENTER button keyboard)
                    win.protocol("WM_DELETE_WINDOW",(lambda: 'pass')())

                    fig.canvas.draw()
                    fig.canvas.flush_events()
                    input('hit ENTER to continue')
                #endif
                ax1.cla()
                ax2.cla()
                fig.clf()
            #endif
        #endif
    #endfor

    plt.close(fig)
#enddef


"""
#-------------------------------------------------------------------------------
#real work (pre is 'stat') for making the average activities, used for future
#based on ELLWF_Composition_Vectors_Rev-1
#-------------------------------------------------------------------------------
def make_avg_out_aux_NOT_USED(loc, iso, tdir, pre, args):
    nofig   = args.nofig
    to_show = args.show
    avg_type= args.avg

    #set up the figure here (one time), replot each time, otherwise run out of
        memory
    if nofig:
        pass
    else:
        fig = plt.figure()
    #endif

    if avg_type == 'etst':
        etst_l_du_stats = []
    #endif

    #print(iso)
    for du_type in ('et','st','test'):
        #print(du_type)
```

```
#get list of files that match DU type (eg ET* = ET01, ET02, ET03)
l_stat_files = get_lfiles(pre, du_type, iso, tdir)

l_du_stats = []
for f in l_stat_files:
    afile = f.split('_')
    du    = afile[-2]          #second from last value in file string (may
       have _ in dir name, ie earea_unc)

    #values from ELLWF_Composition_Vectors_Rev-1.xlsx sheet CVEC
    avg_factor = 1
    sof_total  = 1
    if du in ['et01','st01','st02','st03','st04','st05']:
        avg_factor = 1
        sof_total  = 1
    elif du in ['et02', 'et03','st06','st07','st08','st09','st14']:
        avg_factor = 0.95
        if   du == 'et02':
            sof_total = 0.751
        elif du == 'et03':
            sof_total = 0.352
        elif du == 'st06':
            sof_total = 0.143
        elif du == 'st07':
            sof_total = 0.456
        elif du == 'st08':
            sof_total = 0.370
        elif du == 'st09':
            sof_total = 0.936
        elif du == 'st14':
            sof_total = 0.943
       #endif
    elif du in ['test']:
        print("come check make_avg_out_aux")
        avg_factor = 1
        sof_total  = 1
    else:
        continue
    #endif

    #do not adjust based on future
    avg_factor = 1
    sof_total  = 1

    bfile       = os.path.join(os.path.join(tdir,'bias'),'bias_'+du+'_'+iso
       +'.txt')
    bias_factor = db_info.read_bfile(bfile)

    with open(f,'r') as fh:
        a_lines = fh.readlines()
        floats = [float(x) for x in a_lines]
```

```
            mu = floats[0] * avg_factor/sof_total  #I_WITS_now  (see Excel file
                for how calc is done)
            sig= floats[1] * avg_factor/sof_total  #abs sig
            tru= mu * bias_factor
            #print(du_type,loc,iso,mu,sig,tru)

            l_du_stats.append((mu,sig,tru,sig/mu))  #see excel file (columns C
                and K)

            etst_l_du_stats.append((mu,sig,tru,sig/mu))

        #endwith
    #endfor

    if l_du_stats == []:
        pass
    else:
        a = np.array(l_du_stats)

        if avg_type == 'wav':           #weighted average (don't do this)
            #if iso in ['h-3','sr-90']:
            #    print(a)
            aweights = 1/(a[:,1]**2)    #weights = 1/(sig^2)
            avg_mu   = np.average(a[:,0], weights=aweights)
            avg_tru  = np.average(a[:,2], weights=aweights)
            #avg_sig  = 1/np.sqrt(np.sum(a[:,1]))
            avg_sig  = 1/np.sqrt(np.sum(aweights))
        elif avg_type == 'worst':  #use the values with the maximum frac
            uncertainty and no bias
            max_sig = np.amax(a[:,3])
            vloc    = np.where(a[:,3] == max_sig)
            i       = vloc[0][0]   #index
            avg_mu  = a[i][0]
            avg_tru = avg_mu
            avg_sig = a[i][1]
        elif avg_type == 'etst':
            avg_mu     = np.mean(a[:,0])
            avg_tru    = np.mean(a[:,2])
            if len(a) > 1:  #more than one DU with this isotope, get the
                population standard deviation of the mu's as the uncertainty
                avg_sig = np.std(a[:,0], ddof=0)
            else:           #just this DU has this isotope, use the sigma of
                this DU.
                avg_sig = a[0,1]
            #endif
        else:                           #ordinary average - see Excel file
            avg_mu     = np.mean(a[:,0])
            avg_tru    = np.mean(a[:,2])
            if len(a) > 1:  #more than one DU with this isotope, get the sample
                standard deviation of the mu's as the uncertainty
                avg_sig = np.std(a[:,0], ddof=1)
```

```
    else:            #just this DU has this isotope, use the sigma of
        this DU.
        avg_sig = a[0,1]
    #endif
#endif

delta_bias = avg_tru/avg_mu

#if du_type =='et' and iso=='pu-241':
#print(l_du_stats,du_type,loc,iso, avg_mu, avg_tru, avg_sig) #, a[:,1])

#avg files
avgsfile = pre    + '_avg' + du_type + '_' + iso + '.txt'
avgbfile = 'bias' + '_avg' + du_type + '_' + iso + '.txt'

#make avg bias
bdir  = os.path.join(tdir,'bias')
bfile = os.path.join(bdir, avgbfile)
make_bias_aux(bfile,delta_bias)

#make avg stat file
sdir  = os.path.join(tdir,'stat')
sfile = os.path.join(sdir, avgsfile)
make_stat_aux(sfile, avg_mu, avg_sig)

#get info for PDF and CDF
#print(iso, avg_mu, avg_sig)
x_val, pdf_val, cdf_val = make_dist(avg_mu, avg_sig)
x_val_fraction          = x_val/avg_mu

#make PDF
pdir    = os.path.join(tdir,'pdf')
pfile   = os.path.join(pdir,'pdf_avg' + du_type + '_' + iso + '.txt')
make_pdf_aux(pfile,x_val_fraction,pdf_val)

#get the info for the CDF
cdir    = os.path.join(tdir,'cdf')
cfile   = os.path.join(cdir,'cdf_avg' + du_type + '_' + iso + '.txt')
make_cdf_aux(cfile,x_val_fraction,cdf_val)

#plot the data
if nofig:
    pass
else:
    du_per_unc = avg_sig/avg_mu*100
    fig, ax1, ax2 = plot_fig(fig, loc, iso, avg_mu, delta_bias,
        du_per_unc, x_val_fraction, pdf_val, cdf_val)

    fdir  = os.path.join(tdir,'figs')
    ffile = os.path.join(fdir,'fig_avg'+du_type+'_'+iso+'.png')
    make_figure_aux(ffile,fig)
```

```
            if to_show:
                win = plt.gcf().canvas.manager.window                    #disable
                    the close button (use the ENTER button keyboard)
                win.protocol("WM_DELETE_WINDOW",(lambda: 'pass')())

                fig.canvas.draw()
                fig.canvas.flush_events()
                input('hit ENTER to continue')
            #endif
            ax1.cla()
            ax2.cla()
            fig.clf()
        #endif
    #endif
#endfor

#average etst
if avg_type == 'etst':
    print('in etst')
    print(etst_l_du_stats)
    du_type = 'etst'
    if etst_l_du_stats == []:
        pass
    else:
        a = np.array(etst_l_du_stats)
        if avg_type == 'etst':
            avg_mu     = np.mean(a[:,0])
            avg_tru    = np.mean(a[:,2])
            if len(a)  > 1:  #more than one DU with this isotope, get the
                population standard deviation of the mu's as the uncertainty
                avg_sig = np.std(a[:,0], ddof=0)
            else:            #just this DU has this isotope, use the sigma of
                this DU.
                avg_sig = a[0,1]
            #endif
        #endif

        delta_bias = avg_tru/avg_mu

        #avg files
        avgsfile = pre    + '_avg' + du_type + '_' + iso + '.txt'
        avgbfile = 'bias' + '_avg' + du_type + '_' + iso + '.txt'

        print(avgbfile)


        #make avg bias
        bdir  = os.path.join(tdir,'bias')
        bfile = os.path.join(bdir, avgbfile)
        make_bias_aux(bfile,delta_bias)
```

```
        #make avg stat file
        sdir  = os.path.join(tdir,'stat')
        sfile = os.path.join(sdir, avgsfile)
        make_stat_aux(sfile, avg_mu, avg_sig)

        print(avgsfile)

        #get info for PDF and CDF
        #print(iso, avg_mu, avg_sig)
        x_val, pdf_val, cdf_val = make_dist(avg_mu, avg_sig)
        x_val_fraction          = x_val/avg_mu

        #make PDF
        pdir    = os.path.join(tdir,'pdf')
        pfile   = os.path.join(pdir,'pdf_avg' + du_type + '_' + iso + '.txt')
        make_pdf_aux(pfile,x_val_fraction,pdf_val)

        #get the info for the CDF
        cdir    = os.path.join(tdir,'cdf')
        cfile   = os.path.join(cdir,'cdf_avg' + du_type + '_' + iso + '.txt')
        make_cdf_aux(cfile,x_val_fraction,cdf_val)

        #plot the data
        if nofig:
            pass
        else:
            du_per_unc = avg_sig/avg_mu*100
            fig, ax1, ax2 = plot_fig(fig, loc, iso, avg_mu, delta_bias,
               du_per_unc, x_val_fraction, pdf_val, cdf_val)

            fdir  = os.path.join(tdir,'figs')
            ffile = os.path.join(fdir,'fig_avg'+du_type+'_'+iso+'.png')
            make_figure_aux(ffile,fig)

            if to_show:
                win = plt.gcf().canvas.manager.window                    #disable
                   the close button (use the ENTER button keyboard)
                win.protocol("WM_DELETE_WINDOW",(lambda: 'pass')())

                fig.canvas.draw()
                fig.canvas.flush_events()
                input('hit ENTER to continue')
            #endif
            ax1.cla()
            ax2.cla()
            fig.clf()
        #endif
    #endif
#endif
```

```
    plt.close(fig)
#enddef
"""

#-------------------------------------------------------------------------------
#run each iso
#-------------------------------------------------------------------------------
def monoprocess2(loc, iso, tdir, pre, args):
    make_avg_out_aux(loc, iso, tdir, pre, args)
#enddef

def multiprocess2(loc, isos, tdir, pre, args):
    iters = []
    for iso in isos:
        iters.append((loc, iso, tdir, pre, args))
    #endfor

    #REAL multiprocessing starts here
    pool = multiprocessing.Pool(None)
    pool.starmap(monoprocess2, iters)
#enddef

#-------------------------------------------------------------------
#get the isotopes in dir
#-------------------------------------------------------------------
def get_isos(outdir, pre):
    dup_isos = []
    all_files = os.listdir(outdir) #technically files and dirs, but should just be
       files
    for f in all_files:            #replace all text except for iso
        x = f.replace('.txt','')
        y = x.replace(pre,'')

        reg = r'_.*_'
        z   = re.sub(reg,'',y)
        dup_isos.append(z)
    #endfor
    isos = set(dup_isos) #unquify list to make list of isotopes in outdir (stat dir)
    return isos
#enddef

#-------------------------------------------------------------------
#average the output files of each DU type
#-------------------------------------------------------------------
def make_avg_out(loc,type,tdir,args):
    loc    = loc.lower() #used in make fig
    pre    = 'stat'
    outdir = os.path.join(tdir,pre)
    if os.path.isdir(outdir):
        pass
    else:
```

```
        print(outdir,"doesn't exist, not making avg")
        return
    #endif

    #get set of isotopes in outdir
    isos = get_isos(outdir, pre)

    multiprocess2(loc, isos, tdir, pre, args)

#enddef

#-----------------------------------------------------------------
#save the plot of the CDF and PDF as a figure
#-----------------------------------------------------------------
def make_figure_aux(bfile,fig):
    fig.savefig(bfile)
#enddef

def make_figure(type, loc, iso, fig):
    new_loc = db_info.translate_wits_to_pa(loc)
    tdir    = db_info.get_data_subdir(type,'input') #not really input, but don't
       create a new best
    bdir    = os.path.join(tdir,'figs')
    chk_dir(bdir)

    bfile   = os.path.join(bdir,'fig_'+new_loc+'_'+iso.lower()+'.png')
    make_figure_aux(bfile, fig)
#enddef

#-----------------------------------------------------------------
#create the figure plot
#-----------------------------------------------------------------
def plot_fig(fig, loc, iso, mu, delta_bias, du_per_unc, x_val_fraction, pdf_val,
    cdf_val):
    ax1 = fig.add_subplot(111)
    ax2 = ax1.twinx() #make y2 axis
    #ax1.set_xlabel(r'$\delta$' + ' (uncertainty)')
    #ax1.set_xlabel("normalized "+ r'$I_{WITS_{now}}$')
    ax1.set_xlabel("normalized "+ r'$I_{W}$')
    ax1.set_ylabel('pdf')
    ax2.set_ylabel('cdf')

    inow    = "{x:.2e}".format(x=mu)
    db      = "{x:.2f}".format(x=delta_bias)
    per_unc = "{x:.0f}".format(x=2*du_per_unc)

    #print(iso, per_unc)

    b_hl      = ""
    bias_text = "Unbiased"
    diff = 1-delta_bias
```

```
    diff_per = "{x:.0f}".format(x=diff*100)
    if diff > 0:
        b_hl = "high"
        bias_text = "WITS value biased "+diff_per+"% high"
    elif diff < 0:
        b_hl = "low"
        bias_text = "WITS value biased "+diff_per+"% low"
    #endif

    pa_loc = db_info.translate_wits_to_pa(loc)

    ftitle = iso.capitalize() + ' ' + pa_loc.upper() + '\n'
    #ftitle = ftitle + r'$I_{WITS_{now}}=$' + inow + ' Ci' + '\n'
    ftitle = ftitle + r'$I_{W} $= ' + inow + ' Ci' + '\n'
    ftitle = ftitle + r'$\delta_B $= ' + db + ' ('+bias_text+')\n'
    ftitle = ftitle + 'Uncertainty ('+r'$2\sigma$'+') = ' + per_unc + '%'

    ax1.set_title(ftitle)
    #ax1.plot(x_val, y_val, '+-',label="PDF") #show lines with + markers
    ax1.plot(x_val_fraction, pdf_val, label="pdf")

    #ax1.plot(sorted_data,cdf2,label="CDF")
    ax2.plot(x_val_fraction,cdf_val,color='orange',label="cdf")

    #ax1.legend(loc="upper left")

    fig.tight_layout()

    return fig, ax1, ax2
#enddef

#-------------------------------------------------------------
#bias output
#delta_bias - currently total_bias/total_act of isotope
#-------------------------------------------------------------
def make_bias_aux(bfile, delta_bias):
    with open(bfile,'w') as bfileh:
        bfileh.write("{:.4e}".format(delta_bias))
    #endwith
#enddef

def make_bias(type, loc, iso, l_bias, sum_act):
    sum_bias = sum(l_bias)  #sum_bias = true_value, sum_act = now_value

    delta_bias = sum_bias/sum_act

    if delta_bias > 1:
        print('bias > 1                                          ',loc,iso,delta_bias)

    new_loc = db_info.translate_wits_to_pa(loc)
    tdir    = db_info.get_data_subdir(type,'input')#not really input, but don't
```

```
        create a new best
    bdir    = os.path.join(tdir,'bias')
    chk_dir(bdir)
    bfile   = os.path.join(bdir,'bias_'+new_loc+'_'+iso.lower()+'.txt')
    make_bias_aux(bfile,delta_bias)
    return delta_bias
#enddef


#-----------------------------------------------------------------
#PDF output
#-----------------------------------------------------------------
def make_pdf_aux(pfile,xval,yval):
    with open(pfile,'w') as pfileh:
        lines = ""
        for i in range(0,len(xval)):
            #line  = "{:.5e}".format(xval[i]) + "," + "{:.5e}".format(yval[i]) + "\n
                "
            line  = str(xval[i]) + "," + str(yval[i]) + "\n"  #not formatting
            lines = lines + line
        #endfor
        pfileh.write(lines)
    #endwith
#enddef

def make_pdf(type, loc, iso, xval, yval):
    new_loc = db_info.translate_wits_to_pa(loc)
    tdir    = db_info.get_data_subdir(type,'input') #not really input, but don't
       create a new best
    pdir    = os.path.join(tdir,'pdf')
    chk_dir(pdir)
    pfile   = os.path.join(pdir,'pdf_'+new_loc+'_'+iso.lower()+'.txt')
    make_pdf_aux(pfile,xval,yval)
#enddef

#-----------------------------------------------------------------
#CDF output
#-----------------------------------------------------------------
def make_cdf_aux(cfile,sdata,cdf):
    with open(cfile,'w') as cfileh:
        lines = ""
        for i in range(0,len(sdata)):
            line  = "{:.5e}".format(sdata[i]) + "," + str(cdf[i]) + "\n"  #not
               rounding y-axis so it's unique
            lines = lines + line
        #endfor
        cfileh.write(lines)
    #endwith
#enddef

def make_cdf(type, loc, iso, sdata, cdf):
```

```
    new_loc = db_info.translate_wits_to_pa(loc)
    tdir    = db_info.get_data_subdir(type,'input')#not really input, but don't
       create a new best
    cdir    = os.path.join(tdir,'cdf')
    chk_dir(cdir)
    cfile   = os.path.join(cdir,'cdf_'+new_loc+'_'+iso.lower()+'.txt')
    make_cdf_aux(cfile,sdata,cdf)
#enddef

#-------------------------------------------------------------
#statistics output (mu 1st line, sigma 2nd line of DU statistics)
#-------------------------------------------------------------
def make_stat_aux(sfile,mu,sig):
    with open(sfile,'w') as sfileh:
        lines = ""
        for i in (mu, sig):
            line  = str(i) + "\n"
            lines = lines + line
        #endfor
        sfileh.write(lines)
    #endwith
#enddef


def make_stat(type, loc, iso, mu, sig):
    new_loc = db_info.translate_wits_to_pa(loc)
    tdir    = db_info.get_data_subdir(type,'input')#not really input, but don't
       create a new best
    sdir    = os.path.join(tdir,'stat')
    chk_dir(sdir)

    sfile   = os.path.join(sdir,'stat_'+new_loc+'_'+iso.lower()+'.txt')
    make_stat_aux(sfile,mu,sig)
#enddef

#-------------------------------------------------------------
#not fake pdf, but fake samples
#make a normal PDF using the reported uncertainty
#assume uncertainty is mu and that 0 is -5sigma, so 1sigma is mu/5
#then get 30 samples from this distribution to return

#make a set of num_samples using the 1 package (each sample has activity 1/30 of
    total act)
#-------------------------------------------------------------
def fake_pdf2(pkg_act, pkg_unc):
    db_info.set_seed()

    frac_unc = pkg_unc/pkg_act
    #print(frac_unc)

    mu  = frac_unc
```

```
    sig = frac_unc/5 #0 is -5sig
    num_samples = 30

    s = np.random.normal(mu, sig, num_samples)

    xset = []
    for unc in s:
        xset.append((unc,1/num_samples)) #pkg_act/num_samples/pkg_act))
    #endfor
    #print(xset)
    return xset
#enddef

#-------------------------------------------------------------
# THIS IS TRICKY TO UNDERSTAND - pastTad was smarter than futureTad
# best way to do this is walk through non-overpacked 1 cut first,
#     then non-overpacked multiple cut containers,
#     finally look at overpacked nested
#
#recursive read of dumpfile -
#read the dumpfile dictionary of activity and uncertainty in each cut/container
#return the total activity in the and the list of fractions (unc/act)
#
#interesting sidebars:
#this doesn't seem to want to work with l_act_unc stored as tuples (I tried)
#it also "eats" l_act_unc - somehow overwrites it and sets it to [],
#   so if l_act_unc is called back in main function it shows up as []
#   I don't understand how it does that
#
#l_act_unc = [l_cuts,l_inner_cntrs]
#-------------------------------------------------------------
def get_act2(l_act_unc, sum_act):
    #print(l_act_unc,sum_act)
    if not l_act_unc:
        return 0, []
    #endif
    el = l_act_unc.pop(0)
    if isinstance(el, list):
        head_act, head_lfrac = get_act2(el,sum_act)
        tail_act, tail_lfrac = get_act2(l_act_unc,sum_act)
        return head_act+tail_act, head_lfrac + tail_lfrac
    else:
        act = list(el.keys())   #el[1]
        unc = list(el.values()) #el[1]
        head_act = float(act[0])
        head_unc = float(unc[0])
        tail_act, tail_lfrac = get_act2(l_act_unc,sum_act)
        if head_act == 0:  #shouldn't happen - but it does (should list pkg here
            where that happens)
            pass
        else:
```

```
            per_act      = head_unc/head_act  #convert act unc to fraction of
                activity unc
            frac_tot_act = head_act/sum_act   #frac total act is activity of cut/
                container vs total activity of DU
            tail_lfrac.append((per_act,frac_tot_act,head_act,head_unc))
        #endif
        return head_act+tail_act, tail_lfrac
    #endif
#enddef

def get_cut_info(l_cuts, sum_act):
    tcutact = 0
    cinfo   = []
    for cut in l_cuts:
        cid = cut[0]
        el  = cut[1]
        lact = list(el.keys())
        lunc = list(el.values())
        act = float(lact[0])
        unc = float(lunc[0])

        if act == 0:
            #print("cut activity is 0")
            pass
        else:
            tcutact      = tcutact + act
            per_act      = unc/act
            frac_tot_act = act/sum_act
            cinfo.append((per_act,frac_tot_act,act,unc))
        #endif
    #endfor
    return tcutact, cinfo
#enddef

def get_act3(l_act_unc, sum_act):
    [l_cuts,l_inner_cntrs] = l_act_unc
    tact  = 0
    cinfo = []

    tact, cinfo = get_cut_info(l_cuts, sum_act)

    if l_inner_cntrs == []:
        pass
    else:
        for cntr in l_inner_cntrs:
            itact, icinfo     = get_act3(cntr[1], sum_act)
            tact = tact + itact
            cinfo.extend(icinfo)
        #endfor
    #endif
    return tact, cinfo
```

```
#enddef


#-------------------------------------------------------------------
#heart of the problem
#-------------------------------------------------------------------
def make_dist(mu, sig):
    num_samples = 1000

    rv    = ss.norm(mu, sig) #freeze the distribution

    """ I don't think this is right function (or I don't know how to implement it
        correctly)
    a  = 0
    b  = np.inf
    rv = ss.truncnorm(a,b,loc=mu,scale=sig)
    """

    x_val = np.linspace(rv.ppf(0.00001),rv.ppf(0.99999),num_samples)

    pdf_val = rv.pdf(x_val)
    cdf_val = rv.cdf(x_val)

    for v in [x_val, pdf_val, cdf_val]:
        v[v<0]=0 #convert everything less than 0 to 0 b/c activity cannot be
            negative
    #endfor

    return x_val, pdf_val, cdf_val
#enddef

#-------------------------------------------------------------------
#REAL WORK DONE HERE
#-------------------------------------------------------------------
def make_hist_pdf_cdf(type, iso, l_packages, loc, args, fig):
    to_show = args.show
    nofig   = args.nofig

    #type = base:
    #iso_act looks like:
    #iso[loc] = {iso1:[cnum,olvl,act,unc,bias],
    #            iso2:[cnum,olvl,act,unc,bias]}

    #type = best:
    #iso_act looks like:
    #iso[loc] = {iso1:[[cnum,l_act_unc,tact,bias],...],
    #            iso2:[[cnum,l_act_unc,tact,bias],...]}
    #
    #l_act_unc = [l_cut,l_inner_cntr]
    #
    #l_cut     = [[cutid,{str(act):unc}],...]
```

```
#l_inner_cntr = [[cnum,[l_cut,l_inner_cntr]],...]


#{str(act):unc},{str(act):unc},...

#unc is the absolute/actual uncertainty (curies, not %)
#bias is the true (unbiased) value

#set up the figure here (one time), replot each time, otherwise run out of
    memory
"""
if nofig:
    pass
else:
    fig = plt.figure()
#endif
"""

sum_act = 0
l_bias  = []  #list of list of list of true_values (un biased) (waste cut)

#type = base
#                                                                    0,      1,
    2,  3,   4
#each data in l_packages is a container with that isotope = [cntnr,  opack,  act
    ,unc,bias]

#type = best
#                                                                   0,       1,
    2,       3
#each data in l_packages is a container with the  isotope = [cnum,l_act_unc,tact
    ,     bias]
#l_act_unc = [[l_cuts],[l_inner_cntrs]]  l_cuts={str(act):unc},{str(act):unc
    },...] #has to be dict b/c get_act2 needs it (breaks when it's a tuple)
for d in l_packages:
    sum_act = sum_act + d[2]  #total activity in DU (this works for both base
        and best)
#endfor

#testing larry pdf
larry_tot = 0
larry_unc = []
ttact     = 0
xset      = []
for d in l_packages:
    if d[2] == 0:
        if d[0] in ['A014','RD00009750','CBALX9111']:  #ok to skip these (A014 Z
            -0 is 0, RD00009750 PB-1 is 0, CBALX9111 all are 0) in WITS
            pass
        else: #warn about skipping these
```

```
            print(loc, d[0], iso, "activity was stored as 0 - skipping this
                container", d[1], d[2])
    else:
        if type == 'base':
            per_act      = d[3]/d[2]    #convert actual unc to fraction of
                activity unc
            frac_tot_act = d[2]/sum_act #frac_total_act is activity of container
                vs total activity of DU

            """ USED to create fake PDF/CDF for the PA Chapter 2 report
            if frac_tot_act > 0.065 and iso == 'NP-237':
                per_act = 0.25
            """
            xset.append((per_act,frac_tot_act,d[2],d[3]))
            l_bias.append(d[4])          #list of un-biased activities
        else: #type == 'best'
            #print('cnum',d[0],'tact',d[2])

            tact, lfrac = get_act3(d[1], sum_act) #d[1] is the l_act_unc = [
                l_cuts,l_inner_cntrs] TRICKY!! but necessary
            #print(sum_act)
            #print(tact)
            #print(lfrac)
            for f in lfrac:
                xset.append(f) #each f is (fractional_uncert, frac_tot_act,
                    activity, absolute uncertainty)
                larry_tot = larry_tot + f[2]
                """
                if iso == 'AM-242M':
                    print(f[3],f[0])
                """
                larry_unc.append(f[3]) #absolute uncertainties (activity*
                    per_unc)
            #endfor
            l_bias.append(d[3])          #list of un-biased activities - (inner
                containers already summed together in zmake_dump - I think)
            ttact = ttact + tact
        #endif
    #endif
#endfor

#print("-------------------------")
#print(loc, iso)
#print(sum_act)  #should be same as larry_tot (it is)
#print(larry_tot)
utot = 0
for u in larry_unc:
    utot = utot + u**2
#endfor
l_unc = math.sqrt(utot) #absolute uncertainty calculated in quadrature
```

```
"""
if iso == 'AM-242M':
    print(l_unc)
    print(sum_act, larry_tot, ttact)
    print(l_unc/larry_tot) #fraction uncertainty  (just to see what this looks
        like)
"""
if larry_tot == 0:
    print(iso, loc, "does not exist")
    return fig
else:
    du_per_unc = l_unc/larry_tot * 100
#endif

#may need to do something with this info below, when len_data = 1

#-----------------------------------------
#make the total inventory PDF and CDF
#-----------------------------------------
#two ways to do this:
#assume total inventory uncertainty is random and independent and inventory_now
    (larry_tot and l_unc)
#   describes a gaussian (or maybe a log-normal) distribution and
    inventory_close scales linearily with inventory_now
#   and uncertainty_close distribution is same as uncertainty_now distribtuion

#other way assumes inventory_now is equal to weighted average of invetory *
    uncertainty distribution
#THIS may turn out to be the same thing

mu          = larry_tot  #I_wits_now (Ci)
sig         = l_unc      #absolute uncertainty (Ci)

""" #see make_dist
num_samples = 1000
#s = np.random.normal(mu, sig, num_samples)
#s = ss.norm(x,mu,sig)

rv = ss.norm(mu, sig) #freeze the distribution

#x = np.linspace(0,mu+5*sig,num_samples) #cover from 0 to invetory+5sigma
x_val = np.linspace(rv.ppf(0.00001),rv.ppf(0.99999),num_samples)
x_val[x_val<0]=0 #convert everything less than 0 to 0
pdf_val = rv.pdf(x_val)
cdf_val = rv.cdf(x_val)
"""

x_val, pdf_val, cdf_val = make_dist(mu, sig)
x_val_fraction = x_val/mu
#print(x_val_fraction)
```

```
"""
print(x)
print(y)
print(z)

ax1.plot(x, y, 'k-')
ax2.plot(x, z, 'k-')
plt.show
input("x")

return


#x_val = x
#y_val = y
print("here2")
nsxset = list(map(list, zip(x,y)))
#print(nsxset)
#return
print("there")
"""

"""

len_data = len(xset)
if len_data > 1: #more than 1 package/container with activity
    pass
elif len_data == 1:
    print("    only 1 cut or container with this isotope, making normal
       distribution data, based on package uncertainty")
    act = xset[0][2] #float(list(tmp_l_act_unc.keys())[0]) #get the activity out
        of the dictionary
    unc = xset[0][3] #list(tmp_l_act_unc.values())[0]      #get the uncertainty
       out of the dictionary
    xset = fake_pdf2(act,unc)
else:
    print("len_data = []")
    print(len_data)
    print("breaking out of this isotope loop")
    input("is this OK - should this ever happen?") #this should never happen,
       but just in case
    break
#endif

#sort by container list by uncertainty
sxset = sorted(xset, key = lambda x: x[0])

#group values that have same uncertainty and add their frac_tot_activities
   together (to make "histogram" for PDF)
df      = pandas.DataFrame(sxset)
df2     = df.groupby(0,as_index=False)[1].sum()
```

```
#records = df2.to_records(index=False) #this doesn't appear to be used
nsxset  = df2.values.tolist()

#-----------------------------------------
#making PDF and CDF and figure
#-----------------------------------------
x_val = [x[0] for x in nsxset] #unc        #This is the PDF
y_val = [x[1] for x in nsxset] #frac act
make_pdf(type, loc, iso, x_val, y_val)

print('This many cntnrs: ',len(x_val))

scale = 1

cdfdata = []
for x in nsxset:
    m = round(x[1]*scale) #scale the "height" of the histogram by 1000 (frac tot
        act * 1000)
    if m == 0:            #it has very low frac of total activity
        m = 1
    cdfdata.extend([x[0]]*m) #count this value 'm' number of times depending on
       fraction of activity (ie from histogram)
#endfor

#get the info for the CDF
sorted_data = np.sort(cdfdata)
cdf2        = np.arange(len(cdfdata)) / float(len(cdfdata)-1) #y-axis - these
   will be unique
make_cdf(type, loc, iso, sorted_data, cdf2)

"""

#-----------------------------------------
#making PDF and CDF and figure
#-----------------------------------------
#print('This many samples: ',len(x_val))

make_pdf(type, loc, iso, x_val_fraction, pdf_val)

#get the info for the CDF
make_cdf(type, loc, iso, x_val_fraction, cdf_val)

#save the mu and sigma so can average it later
make_stat(type, loc, iso, mu, sig)

#-----------------------------------------
#make the bias
#-----------------------------------------
delta_bias = make_bias(type, loc, iso, l_bias, sum_act)

#-----------------------------------------
```

```
    #plot the data and show it
    #-----------------------------------------
    if nofig:
        pass
    else:
        fig, ax1, ax2 = plot_fig(fig, loc, iso, mu, delta_bias, du_per_unc,
            x_val_fraction, pdf_val, cdf_val)

        make_figure(type, loc, iso, fig)

        if to_show:
            win = plt.gcf().canvas.manager.window                    #disable the
                close button (use the ENTER button keyboard)
            win.protocol("WM_DELETE_WINDOW",(lambda: 'pass')())

            fig.canvas.draw()
            fig.canvas.flush_events()
            input('hit ENTER to continue')
        #endif
        #print("clearing axis and figure")
        ax1.cla()
        ax2.cla()
        fig.clf()
    #endif

    """
    if nofig: #closing here because there is a memory leak somewhere in matplotlib
        pass
    else:
        plt.clf()
    #endif
    """

    return fig
#enddef

#-------------------------------------------------------------------------------
#get the files to include in the zipfile
#-------------------------------------------------------------------------------
def get_files(tdir):
    filepaths = []
    # Read all directory, subdirectories and file lists
    lentdir = len(tdir)
    for root, _directories, files in os.walk(tdir):
        for filename in files:
            #print(root, _directories, files)

            # Create the full filepath by using os module.
            filep = os.path.join(root, filename)
            filepaths.append((filep,filep[lentdir:]))
            #print(filepaths)
```

```
            #input("x")
        #endfor
    #endfor

    # return all paths
    return filepaths
#enddef

#-------------------------------------------------------------------------------
#run each location
#-------------------------------------------------------------------------------
def monoprocess(loc,type,dtdir,isos,args):
    db_info.set_seed.counter = 0

    msg = "==========\n"
    msg = msg + loc + "\n"
    print(msg)

    #dump file
    dtfile = os.path.join(dtdir,'dump_'+loc.lower()+'.txt')
    if os.path.isfile(dtfile):
        short_list = unc_aux.get_short_list_isos()

        if args.show:  #make it interactive, but hide toolbar
            plt.ion()
            plt.rcParams['toolbar'] = 'None'
        #endif

        #READ dump OBJECT
        with open(dtfile) as dfileo:
            #can't do it this way b/c of memoryError
            #   dump_obj = json.load(dfileo)
            #print(dump_obj)
            #for iso, l_packages in dump_obj.items():
            fig = plt.figure()
            #print("fig opened")

            for line in dfileo:

                aline      = line.split(',',1)
                iso        = aline[0]
                l_packages = json.loads(aline[1])
                #print(iso,l_packages)
                if l_packages == []: #no packages with this isotope in this DU
                    continue
                #endif

                #if iso != 'K-40': #for debugging single isotopes
                #    continue

                if isos == 'SHORT' and iso not in short_list: #skip this non-short
```

```
                    list isotope
                    continue
                #endif

                #DO HISTOGRAM AND MAKE PDF AND CDF FILE HERE
                fig = make_hist_pdf_cdf(type, iso, l_packages, loc, args, fig)
            #endfor
            plt.close(fig)

        #endwith

        #plt.close('all')
        #gc.collect()
    else:
        print(dtfile + " not found")
        return
    #endif
    print("----------------------------------")
#enddef

#-------------------------------------------------------------------------------
#run in parallel (won't work if toshow is True b/c slow
#-------------------------------------------------------------------------------
def multiprocess(locs,type,dtdir,isos,args):
    #preparing for multiprocessing
    iters = []
    for loc in locs:
        iters.append((loc,type,dtdir,isos,args))
    #endfor

    #REAL WORK STARTS HERE
    #Run each location in parallel
    pool = multiprocessing.Pool(None)
    pool.starmap(monoprocess,iters)
#enddef

#-------------------------------------------------------------------------------
#Main code to read the dump file, which is a dictionary of each location
#then create the PDF, CDF, BIAS, and a figure of each isotope in each location
#optionally show the figure
#optionally create a zip-file so it's easy to upload
#-------------------------------------------------------------------------------
def main(type,loc,isos,args):
    to_show    = args.show
    to_ap2     = args.ap2
    to_zip     = args.zip
    do_stats   = args.stats

    time_start = time.time()

    tdir  = db_info.get_data_subdir(type,'input')
```

```
#make sure all but dump is empty
dirs = os.listdir(tdir)
for d in dirs:
    d = os.path.join(tdir,d)
    #print(d)
    if 'dump' in d:
        pass
    else:
        shutil.rmtree(d)
        os.mkdir(d)
    #endif
#endfor
dtdir = os.path.join(tdir,'dump')

palocs = db_info.get_dump_palocs(type)

if palocs == "":
    return
#endif

if loc == 'ALL':
    locs = palocs
elif loc in palocs:
    locs = [loc]
else:
    print(loc,"not a PA location")
    return
#endif

#main analysis done here
if to_show: #slow
    for loc in locs:
        monoprocess(loc,type,dtdir,isos,args)
    #endfor
else: #fast
    multiprocess(locs,type,dtdir,isos,args)
#endif

if loc != 'ALL':
    print("only 1 loc (",loc,") specified, not doing avg")
else:
    #check for missing isotopes and make them (for current PA locs)
    print("making missing iso current DU files...")
    locs.remove("CIG1") #because isos missing in here now (5/1/2021)...may be in
        the future
    do_missing(locs, type, tdir)
    print("done")

    #make the averages (using the current PA locs)
    print("making average DU files...")
```

```
    make_avg_out(loc,type,tdir,args)
    print("done")

    #make future values (based on averages)
    print("making future files...")
    make_future_files(type, tdir, args)
    print("done")

    #make missing futures (if any)
    print("making missing iso future DU files...")
    flocs = db_info.get_future_locs(type) #CIG1 is done here
    do_missing(flocs, type, tdir)
    print("done")

    #reformat files for Sebastian's code
    print("reformatting CDF and bias for Sebastian's code...")
    all_locs = []
    all_locs.extend(locs)
    all_locs.extend(flocs)
    make_ca_layout(tdir, all_locs, isos, args)
    print("done")
#endif

if to_ap2:
    print("making Appendix2 figs...")
    zapp2.make_ap2(type, locs, isos, args)
    print("done")
#endif

if do_stats:
    print("making statistics...")
    locs.append("CIG1")
    zstats.zstats_main(type, locs, args)
    print("done")
#endif

if to_zip:
    print("making zip...")
    zname = os.path.basename(tdir)

    fileps = get_files(tdir)

    zzfile = os.path.join(os.path.dirname(tdir),zname + '.zip')

    zfile = zipfile.ZipFile(zzfile,'w')
    with zfile:
        for (file,aname) in fileps:
            zfile.write(file, arcname=aname ,compress_type=zipfile.ZIP_DEFLATED)
        #endfor
    #endwith
    print("done")
```

```
        print("\n\n\nCopy zip to PROJWORK")

        print("-----------------------------------")
    #endif


    time_stop = time.time()
    time_diff = time_stop-time_start

    print("This took: ", "{:.2f}".format(time_diff),"secs")
    print("           ","{:.2f}".format(time_diff/60),"mins")
#enddef

#-------------------------------------------------------------------------------
#analyze the dump files to create the histogram, pdf, and cdf files
#for the E-area pa
#-------------------------------------------------------------------------------
if __name__ == '__main__':
    arg_parser = argparse.ArgumentParser(description='Analyze the dump files to make
        the PDF and CDF (< 2 min)')
    group      = arg_parser.add_mutually_exclusive_group()
    msg0 = 'the TYPE of dump file to read (base|best|test):'
    msg1 = '    base is the base case - estimated uncertainty, '
    msg2 = '    best is the best case - use uncertainty, '
    msg3 = '    test is the test case - same as best '
    msg = msg0+msg1+msg2+msg3
    arg_parser.add_argument('Type',
                            metavar='type',
                            type=str,
                            choices=['base','best','test','test_base'],
                            help=msg)
    arg_parser.add_argument('Location',
                            metavar='loc',
                            type=str,
                            help='the LOCATION to analyze (all|NR0|CIG1|...)')
    arg_parser.add_argument('Isotopes',
                            metavar='iso',
                            type=str,
                            choices=['all','short'],
                            help='the ISOTOPES to analyze (all|short)')
    group.add_argument(     '--show',
                            action="store_true",
                            help='show the pdf-cdf plot')
    group.add_argument(     '--nofig',
                            action="store_true",
                            help='do not make the pdf-cdf plot')
    group.add_argument(     '--avg',
                            type=str,
                            choices=['worst'],
                            help='worst=max unc and min bias') #help='ord=calculate
```

```
                                the ordinary arithmetic average, worst=calc max
                                uncert and no bias, etst=calc avg et and st and avg
                                together') #'wav' will calc the weighted mean ')
    arg_parser.add_argument('--stats',
                            action="store_true",
                            help='create cumulative statistics figures')
    arg_parser.add_argument('--zip',
                            action="store_true",
                            help='zip the TYPE directory, so it is fast to upload to
                                HPC')
    arg_parser.add_argument('--ap2',
                            action="store_true",
                            help='make inventory/uncertainty figures for locations')
    arg_parser.add_argument('--ap2ppp',
                            action="store_true",
                            help='if specified, use zppp (prioritized) isotopes,
                                otherwise use short list')

    args  = arg_parser.parse_args()

    type  = args.Type
    loc   = args.Location
    isos  = args.Isotopes

    db_info.set_seed.counter = 0

    main(type.lower(),loc.upper(),isos.upper(),args)
#endif
```

Listing 3: db_info

```
#file has information on the database and where to find files
import json
import numpy as np
import os
import pyodbc
import random
import sys

import unc_aux

#-------------------------------------------------------------
#Translate WITS location to PA location (naming convention)
#-------------------------------------------------------------
def translate_wits_to_pa(loc):
    #key = WITS, value = PA
    rosetta={'CIG1'     :'ST23',
             'CIG2'     :'ST24',
             'ETRENCH1':'ET01',
             'ETRENCH2':'ET02',
             'ETRENCH3':'ET03',
```

```
                'ETRENCH4': 'ET04',
                'ETRENCH5': 'ET05',
                'ETRENCH6': 'ET06',
                'ETRENCH7': 'ET07',
                'ETRENCH8': 'ET08',
                'ETRENCH9': 'ET09',
                'ILV1'    : 'ILV',
                'LAW1'    : 'LAWV',
                'NR0'     : 'NR07E',
                'NR1'     : 'NR26E',
                'SLIT'    : 'ST01',
                'SLIT2'   : 'ST02',
                'SLIT3'   : 'ST03',
                'SLIT4'   : 'ST04',
                'SLIT5'   : 'ST05',
                'SLIT6'   : 'ST06',
                'SLIT7'   : 'ST07',
                'SLIT8'   : 'ST08',
                'SLIT9'   : 'ST09',
                'SLIT10'  : 'ST10',
                'SLIT11'  : 'ST11',
                'SLIT14'  : 'ST14',
                'SLIT17'  : 'ST17',
                'SLIT18'  : 'ST18',
                'SLIT19'  : 'ST19',
                'SLIT20'  : 'ST20',
                'SLIT21'  : 'ST21',
                'SLIT22'  : 'ST22',
                'SLIT24'  : 'ST24',
               }
    if loc in rosetta:
        new_loc = rosetta[loc].upper()
    else:
        new_loc = loc
    #endif
    new_loc = new_loc.lower()
    return new_loc
#enddef

#-------------------------------------------------------------------------------
#read the dump_pa_locs file
#-------------------------------------------------------------------------------
def get_dump_palocs(type):
    dbdir = get_db_subdir()                          #db base OR best OR test OR
       test_base
    dddir = os.path.join(dbdir,'dump_'+ str(type))  #dump_base, dump_best

    #read palocs from dumpfile
    palocs = ""
    dfile  = os.path.join(dddir,'dump_pa_locs.txt')
    if os.path.isfile(dfile):
```

```
        with open(dfile) as dfileo:
            palocs = json.load(dfileo)
        #endwith
    else:
        print("dump_pa_locs.txt is missing!!!")
    #endif
    return palocs
#endif

#-------------------------------------------------------------------------------
#get future DU locations
#-------------------------------------------------------------------------------
def get_future_locs(type):
    palocs = get_dump_palocs(type)
    #print(palocs)
    #as of 9/3/2021
    current_future_locs = ['ET04','ET05','ET06','ET07','ET08','ET09','ST10','ST11','
        ST17','ST18','ST19','ST20','ST21','ST22','ST23','ST24']
    #print(current_future_locs)
    flocs = []
    for loc in current_future_locs:
        if loc in palocs:
            pass
        else:
            flocs.append(loc)
        #endif
    #endfor

    return flocs
#enddef

#-------------------------------------------------------------------------------
#filter isos by priority in file
#-------------------------------------------------------------------------------
def get_isos_to_do(isos_from_file, color, xisos, loc, type):
    isos_to_do = []
    if isos_from_file:
        filter_value = 0
        if color == 'red':
            filter_value = 1
        elif color == 'yellow':
            filter_value = 0.01
        elif color == 'blue':
            filter_value = 0.001
        else:
            filter_value = 0
        #endif
        d_gw_sum        = read_dump(os.getcwd(),"zppp_isos.txt")
        isos_priorities = d_gw_sum[loc]
        for ips in isos_priorities:
            if float(ips[1]) >= filter_value:
```

```
                isos_to_do.append([ips[0], ips[1]])
            #endif
        #endfor
    elif xisos == 'SHORT':
        lisos = unc_aux.get_short_list_isos()
        #maybe need to sort by atomic number
        for i in lisos:
            isos_to_do.append([i, 1])
    else: #all
        dbdir  = get_db_subdir()                    #db
        dddir  = os.path.join(dbdir,'dump_'+ str(type))  #db/dump_base, db/dump_best
            , db/dump_test
        lisos  = read_dump(dddir,'dump_tmp_iso.txt')
        for i in lisos:
            isos_to_do.append([i, 1])
    #endif
    return isos_to_do
#enddef

#read bias file, get bias (only 1 line in file)
def read_bfile(bfile):
    bias = 1
    if os.path.isfile(bfile):
        with open(bfile,'r') as fh:
            for line in fh:
                bias = float(line)
            #endfor
        #endwith
    #endif
    return bias
#enddef

#read stat file, get mu, sigma (absolute values)
def read_statfile(sfile):
    mu  = 1
    sig = 0
    if os.path.isfile(sfile):
        i = 0
        with open(sfile,'r') as fh:
            for line in fh:
                if i == 0:
                    mu = float(line)
                elif i == 1:
                    sig = float(line)
                else:
                    print("error more than 2 lines in file")
                #endif
                i = i + 1
            #endfor
        #endwith
    #endif
```

```
    return mu, sig
#enddef

#read the cdf file
def read_cdf(cfile):
    acdf = []
    with open(cfile,'r') as fh:
        for line in fh:
            aline = line.split(',')
            newx  = float(aline[0])
            newy  = float(aline[1])
            acdf.append([newx,newy])
        #endfor
    #endwith
    return acdf
#enddef

#-----------------------------------------------------------------------------
#set the random seed, update it by 1 every time this function is called
#-----------------------------------------------------------------------------
def set_seed():
    set_seed.seed    = 123456
    set_seed.counter += 1

    seed = set_seed.seed + set_seed.counter
    #print(seed)

    random.seed(seed)
    np.random.seed(seed)
#enddef

#-----------------------------------------------------------------------------
#set up the connection to the database
#-----------------------------------------------------------------------------
def get_conn(type):
    db      = ''
    db_file = ''
    if type in ["test","test_base"]:
        db_file = r'''C:\projects\earea_unc\db\WITS-test.accdb'''
    else:
        db_file = r'''C:\Program Files (x86)\WITS Reports\WITS Reports.accde''' #
            wits reports now points to new location
    #endif

    user          = ''
    password      = ''
    odbc_conn_str = 'DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};DBQ=%s;UID=%s
        ;PWD=%s' %(db_file, user, password)
    conn          = pyodbc.connect(odbc_conn_str)
    return conn
#enddef
```

```
#-------------------------------------------------------------------------------
#data/base or data/best_i location
#usage is either output (make a new one) or input (read location)
#-------------------------------------------------------------------------------
def get_data_subdir(subdirname,usage):
    pdir   = os.path.dirname(os.getcwd())

    if subdirname == 'best':
        i = 0
        while True:
            subdir = os.path.join(pdir,os.path.join('data',subdirname+'_'+str(i)))
               #best_i: best_0, best_1, ...
            if os.path.isdir(subdir):
                i = i + 1
            else: #subdir doesn't exist, create it and necessary subdirs
                if usage == 'input':
                    i = i-1
                    subdir = os.path.join(pdir,os.path.join('data',subdirname+'_'+
                        str(i)))  #best_i: best_0, best_1, ...
                else:
                    os.mkdir(subdir)
                    for sdir in ['bias','cdf','dump','figs','pdf']:
                        ssdir = os.path.join(subdir,sdir)
                        os.mkdir(ssdir)
                    #endfor
                #endif
                break
            #endif
        #endwhile
    else:
        subdir = os.path.join(pdir,os.path.join('data',subdirname))
    #endif

    return subdir
#enddef

#-------------------------------------------------------------------------------
#db subdir
#-------------------------------------------------------------------------------
def get_db_subdir():
    pdir   = os.path.dirname(os.getcwd())
    subdir = os.path.join(pdir,'db')
    return subdir
#enddef

#-------------------------------------------------------------------------------
#read the dump file
#-------------------------------------------------------------------------------
def read_dump(ddir,dfilename):
    dfile = os.path.join(ddir,dfilename)
```

```
    if os.path.isfile(dfile): #try to read cuts from dumpfile
        with open(dfile) as dfileo:
            dump = json.load(dfileo) #READ dump OBJECT
        #endwith
    else:
        print(dfile, "is missing!!!")
        print("what happened?")
        sys.exit()
    #endif
    return dump
#enddef
```

Listing 4: unc_aux

```
#-----------------------------------------------------------------------------
#get the disposal unit locations of interest in the E-area PA
#-----------------------------------------------------------------------------
def get_pa_locs(conn):
    not_pa_list = ['643E',
                   'CIGTSA',
                   'CP-PASSTSA',
                   'EMPTIES',
                   'ET-TSA',
                   'GCOTRUOPS',
                   'INT1',
                   'IT1',
                   'LLW-RMA',
                   'NTS-READY',
                   'SCF',
                   'SLITPCBTSA',
                   'SLITTSA',
                   'UESA',
                   'ETRENCH4','SLIT15','SLIT10',
                   'TRAN1','TRAN2','TRAN3','TRAN4','TRAN5','TRAN6','TRAN7']

    placeholders = ", ".join(["?"] * len(not_pa_list))

    SQL = """
    SELECT LOCATION_CODE
    FROM WITS_TBL_LOCATION
    WHERE FACILITY_CODE = 'EAV'
          AND LOCATION_CODE NOT IN (""" + placeholders + """)
    GROUP BY LOCATION_CODE
    ORDER BY LOCATION_CODE ASC;
    """

    cursor = conn.cursor()
    params = (not_pa_list)
    cursor.execute(SQL,params)
    print('slow part...')
    rows = cursor.fetchall()
```

```
    print('done')

    locs = []
    for row in rows:
        locs.append(row.LOCATION_CODE)
    #endfor

    return locs
#enddef

#-------------------------------------------------------------------------------
#Isotopes *most* important to PA
#-------------------------------------------------------------------------------
def get_short_list_isos():
    return ["H-3","C-14","SR-90","TC-99","I-129","NP-237"]
#enddef

#-------------------------------------------------------------------------------
#Get waste_cuts in a container
#-------------------------------------------------------------------------------
def get_waste_cuts(conn, cntnr):

    SQL = """
    SELECT a.CNTNR_NUM, a.WASTE_CUT_ID, a.ACTIVITY_CI, a.CALC_METHOD, a.STRM_ID_CODE
        , a.STRM_VERSION, a.ELEM_SUB_STRM_ID_CODE, b.WCF_FORM_DOC_NUM, b.
        WCF_FORM_REV_NUM
    FROM (WITS_TBL_WASTE_CUT as a
          INNER JOIN WITS_TBL_WASTE_STREAM as b ON a.STRM_ID_CODE =b.STRM_ID_CODE
              AND a.STRM_VERSION=b.STRM_VERSION)
    WHERE CNTNR_NUM = ? ; """

    cursor = conn.cursor()
    params = (cntnr)
    cursor.execute(SQL,params)
    rows = cursor.fetchall()

    cuts = []
    for row in rows:
        cuts.append([row.WASTE_CUT_ID,row.ACTIVITY_CI,row.CALC_METHOD,row.
            STRM_ID_CODE,row.STRM_VERSION,row.ELEM_SUB_STRM_ID_CODE,row.
            WCF_FORM_DOC_NUM,row.WCF_FORM_REV_NUM])
    #endfor

    return cuts
#enddef
```

Listing 5: zapp2

```
import matplotlib.pyplot as plt  #<-slow
import numpy as np
```

```
import os

import db_info


#read files to make picture
def get_vals(tdir,pre,bias_dir,loc,iso):
    bdir        = os.path.join(tdir,bias_dir)
    bfile       = os.path.join(bdir,'bias_'+loc+'_'+iso+'.txt')
    bias_factor = db_info.read_bfile(bfile)

    sdir        = os.path.join(tdir,pre)
    sfile       = os.path.join(sdir, pre + '_'+loc+'_'+iso+'.txt')
    if os.path.isfile(sfile):
        with open(sfile,'r') as fh:
            a_lines = fh.readlines()
            floats = [float(x) for x in a_lines]
            inow = floats[0]  #I_WITS_now
            sig  = floats[1]  #sig
            itru = inow * bias_factor
        #endwith
    else:
        print(sfile, "not found")
        inow = 0
        itru = 0
        sig  = 0
    #endif

    return inow, itru, sig
#enddef

#sort isos by atomic mass number
def sort_isos(isos, to_filter, loc):
    if to_filter:
        x = []
        for (i,ranking) in isos:
            """
            if loc == 'nr7e':
                x = []
                for i in isos:
                    if i[-1] == 'S':
                        i = i.replace('-S','')
                    #endif
                    x.append(i)
                #endfor
            else:
                x = isos
            #endif
            """
            x.append(i)
    else:
```

```
        x = sorted(isos, key=lambda x: (int(x.split("-")[1].replace("m","").replace
            ("M","")))))
    #endif
    x = [z.lower() for z in x]
    return x
#enddef

def make_ap2(type, locs, xisos, args):
    to_filter = args.ap2ppp
    to_show   = args.show

    fig, ax = plt.subplots()

    #do the average ones too (for future)
    locs.extend(['AVGET','AVGST'])

    for oloc in locs:
        print(oloc)
        loc  = db_info.translate_wits_to_pa(oloc).lower()
        tdir = db_info.get_data_subdir(type,'input') #not really input, but don't
            create a new best (get's the type dir, eg best_7)

        pre = 'stat'
        outdir = os.path.join(tdir, pre)
        if os.path.isdir(outdir):
            pass
        else:
            print(outdir,"doesn't exist, not making appendix2 figs")
            return
        #endif

        isos_from_file = False
        if to_filter:
            isos_from_file = True
            color          = 'yellow'  #get those >= 1% of SOF
        #endif
        isos_to_do = db_info.get_isos_to_do(isos_from_file, color, xisos, oloc)
                #either priority or short

        if to_filter and len(isos_to_do)==1:
            isos_to_do = db_info.get_isos_to_do(isos_from_file, 'white', xisos, oloc
                ) #either priority or short
            isos_to_do = isos_to_do[:2]
        #endif

        if to_filter:
            r_prev = 1e9
            temp_isos_to_do = []
            for i,r in isos_to_do:
                #print(i,r)
                #print(i, r_prev*100, r*100)
```

```
        if np.abs(r_prev - r) > 1/100:
            temp_isos_to_do.append([i,r])
        #endif
        r_prev = r
    #endfor
    isos_to_do = temp_isos_to_do
#endif

#fix the NR07E
"""
if to_filter and loc == 'nr07e':
    temp_isos_to_do = []
    for i,r in isos_to_do:
        i = i.replace('-S','')
        temp_isos_to_do.append([i,r])
    #endfor
    isos_to_do = temp_isos_to_do
#endif
"""

#print(isos_to_do)

#sisos = sort_isos(isos_to_do, to_filter, loc) #maybe later
#print(sisos)
#input("x")

ainow   = []
aitru   = []
asignow = []
asigtru = []
loc_isos= []

width    = [] #100 #0.25                    #width of bars
index    = []
index_now= []
index_tru= []

for iso, rank in isos_to_do:
    inow, itru, sig = get_vals(tdir, pre,'bias',loc, iso.lower())
    if inow == 0:
        continue
    else:
        if to_filter:
            index.append(rank*100)
        else:
            index.append(i)
            i = i + 1
        #endif

        #print(iso, inow, itru, sig)
        ainow.append(inow)
```

```
            aitru.append(itru)
            asignow.append(2*sig)

            rel = sig/inow

            #print(rel)
            tsig = itru * rel
            asigtru.append(2*tsig)

            loc_isos.append(iso + '\n'+'{x:.2f}'.format(x=rank*100)+'%')
        #endif
    #endfor


    color_sig_now = 'green'
    color_sig_tru = 'purple'
    ecapsize      = 3

    #print(len(sisos), len(index),len(ainow),len(aitru))

    """
    if to_filter:
        rects1 = ax.bar(index, ainow, width=width, label='$I_{now}$', yerr=
            asignow, error_kw=dict(ecolor=color_sig_now, capsize=ecapsize))
        rects2 = ax.bar(index_tru, aitru, width=width, label='$I_{true}$', yerr=
            asigtru, error_kw=dict(ecolor=color_sig_tru, capsize=ecapsize))
        #rects1 = ax.bar(index_now, ainow, width=np.log10(index_now), label='$I_
            {now}$', yerr=asignow, error_kw=dict(ecolor=color_sig_now, capsize=
            ecapsize))
        #rects2 = ax.bar(index_tru, aitru, width=np.log10(index_tru), label='$I_
            {true}$', yerr=asigtru, error_kw=dict(ecolor=color_sig_tru, capsize=
            ecapsize))
    else:
    """
    width  = 1
    i_now  = [i - width/2 for i in index]
    i_tru  = [i + width/2 for i in index]
    #index  = np.arange(len(ainow)) #the iso locations
    #rects1 = ax.bar(index - width/2, ainow, width, label='$I_{now}$', yerr=
        asignow, error_kw=dict(ecolor=color_sig_now, capsize=ecapsize))
    #rects2 = ax.bar(index + width/2, aitru, width, label='$I_{true}$', yerr=
        asigtru, error_kw=dict(ecolor=color_sig_tru, capsize=ecapsize))
    rects1 = ax.bar(i_now, ainow, width, label='$I_{now}$', yerr=asignow,
        error_kw=dict(ecolor=color_sig_now, capsize=ecapsize))
    rects2 = ax.bar(i_tru, aitru, width, label='$I_{true}$', yerr=asigtru,
        error_kw=dict(ecolor=color_sig_tru, capsize=ecapsize))

    #ax.bar_label(rects1,labels=['SOF: %.2f%%' % i for i in index], padding=5)

    #endif
```

```
#dummy plots to get labels on legend
ax.plot([],[],color_sig_now,label="2$\sigma_{now}$ uncertainty")
ax.plot([],[],color_sig_tru,label="2$\sigma_{true}$ uncertainty")

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

ax.set_title(loc.upper())
ax.set_ylabel('Inventory (Ci)')

if to_filter:
    #ax.set_xlabel('Relative Priority of Groundwater Isotopes (Low to High)
        ')
    ax.set_xlabel('Isotope SOF Groundwater Limits')
    #ax.set_xscale('log')
else:
    ax.set_xlabel('Isotopes')
#endif
ax.set_xticks(index)
ax.set_xticklabels(loc_isos, fontsize=7, rotation=90)
"""
i   = 0
pad = 0
prev_d = 200
for tick in ax.xaxis.get_major_ticks():
    d = tick.get_loc()
    #print(tick.get_loc())

    if prev_d - d < 10:
        pad = pad + 10
    else:
        pad = 0
    #endif
    tick.set_pad(pad)

    i      = i + 1
    prev_d = d
#endfor
"""
ax.set_yscale('log')
ax.legend()
fig.tight_layout()

fdir  = os.path.join(tdir,'figs')
ffile = os.path.join(fdir,'fig_ap2_'+loc+'.png')
fig.savefig(ffile)

if to_show:
    win = plt.gcf().canvas.manager.window                    #disable the
        close button (use the ENTER button keyboard)
    win.protocol("WM_DELETE_WINDOW",(lambda: 'pass')())
```

```
            fig.canvas.draw()
            fig.canvas.flush_events()
            input('hit ENTER to continue')
        #endif
        ax.cla()

    #endfor
    plt.close()
#enddef

def make_ap2_org():
    aloc = ['Loc1']
    isos = ['iso1','iso2'] #sorted by neutron number

    for loc in aloc:

        index = np.arange(len(isos)) #the iso locations
        width = 0.25                 #width of bars

        fig, ax = plt.subplots()

        ainow   = []
        aitru   = []
        asignow = []
        asigtru = []

        for iso in isos:
            if iso == 'iso1':
                inow, itru, sig = (10,5,1)    #now,tru,sig is absolute (I think)
            elif iso == 'iso2':
                inow, itru, sig = (20,19,5)
            #endif
            ainow.append(inow)
            aitru.append(itru)
            asignow.append(2*sig)

            rel = sig/inow
            #print(rel)
            tsig = itru * rel
            asigtru.append(2*tsig)
        #endfor

        color_sig_now = 'green'
        color_sig_tru = 'purple'
        ecapsize      = 3

        rects1 = ax.bar(index - width/2, ainow, width, label='$I_{now}$', yerr=
            asignow, error_kw=dict(ecolor=color_sig_now, capsize=ecapsize))
        rects2 = ax.bar(index + width/2, aitru, width, label='$I_{true}$', yerr=
            asigtru, error_kw=dict(ecolor=color_sig_tru, capsize=ecapsize))
```

```
        #dummy plots to get labels on legend
        ax.plot([],[],color_sig_now,label="2$\sigma_{now}$ uncertainty")
        ax.plot([],[],color_sig_tru,label="2$\sigma_{true}$ uncertainty")

        ax.set_ylabel('Inventory')
        ax.set_title(loc)
        ax.set_xticks(index)
        ax.set_xticklabels(isos)
        ax.legend() #[rects1,rects2,sigma_txt],("","","2sigma"))

        fig.tight_layout()

        plt.show()
    #endfor
#enddef


def main():
    import db_info
    type  = 'best'
    loc   = 'ALL'
    xisos = 'ALL'

    palocs = db_info.get_dump_palocs(type)
    if palocs == "":
        return
    #endif

    if loc == 'ALL':
        locs = palocs
    elif loc in palocs:
        locs = [loc]
    else:
        print(loc,"not a PA location")
        return
    #endif

    class args:
        pass

    args.ap2ppp = True
    args.show   = False


    make_ap2(type, locs, xisos, args)
#enddef

#-------------------------------------------------------------------------------
#analyze the dump files to create the histogram, pdf, and cdf files
#for the E-area pa
```

```
#-------------------------------------------------------------------------------
if __name__ == '__main__':
    main()
#endif
```

Listing 6: zisochk

```
#loop over the ziso_chk excel file and see if the isos are in the bias and cdf dirs
    for the latest best
#maybe automatically create the excel file by linking to the files provided by Larry
#maybe integrate this in either zmake_dump or the zanalysis routines

import argparse
import os
import pandas

import db_info


#do the panda read excel here
def get_isos(path, du):
    ifile = os.path.join(os.getcwd(),"zisochk.xlsx")
    df = pandas.read_excel(ifile,path,header=0)
    val  = df.loc[df[du.upper()]=='x','formatted']
    isos = val.tolist()
    return isos
#enddef

def do_missing(locs, type, tdir):
    flocs    = db_info.get_future_locs()
    all_locs = []
    all_locs.extend(locs)
    all_locs.extend(flocs)

    #check the following dirs (maybe do the bias_ca/cdf_ca instead)
    dirs_to_check = ['bias'] #,'cdf']

    pathways = ['gw','ihi','air','radon']

    #get the list of isos to look for
    #isos = get_isos()

    m = 0
    for dir in dirs_to_check:
        for wloc in all_locs:
            loc = db_info.translate_wits_to_pa(wloc)
            if wloc in flocs:
                if "st" in loc:
                    xloc = "avgst"
                elif "et" in loc:
                    xloc = "avget"
```

```
                #endif
            else:
                xloc = loc
            #endif
            missing_isos = []
            for path in pathways:
                isos = get_isos(path, loc)
                #print(dir, wloc, loc, xloc, path)
                for iso in isos:
                    #print(tdir, dir, loc, iso)
                    f = os.path.join(os.path.join(tdir, dir), dir + '_' + xloc + '_'
                        + iso + '.txt')
                    if os.path.exists(f):
                        pass                #file exists - great! do nothing
                    else:
                        #print(iso, "is missing from", dir, loc)
                        missing_isos.append(iso)
                        m = m + 1
                    #endif
                #endfor
            #endfor
            x = set(missing_isos)

            print("There are ", len(x), "missing isos for", loc)
            print(x)
        #endfor
        print("-"*50)
        if m == 0:
            print("Nothing missing")
        else:
            print("There are", str(m), "missing")
        #endif
        m = 0
        print("="*50)
    #endfor
#enddef



def main():
    #get latest best
    type = 'best'
    tdir  = db_info.get_data_subdir(type,'input')
    dtdir = os.path.join(tdir,'dump')

    #get pa locations
    palocs = db_info.get_dump_palocs(type)
    if palocs == "":
        print("no pa locs - What!!??")
        return
    #endif
```

```
    loc = "ALL"
    if loc == 'ALL':
        locs = palocs
    elif loc in palocs:
        locs = [loc]
    else:
        print(loc,"not a PA location")
        return
    #endif

    do_missing(locs, type, tdir)
#enddef


if __name__ == '__main__':
    main()
#endif
```

Listing 7: zstats

```
import argparse
import json
import matplotlib.pyplot as plt  #<-slow
import multiprocessing
import numpy as np

import os

import sys
import time

import db_info
import unc_aux


#-----------------------------------------------------------------
#save the plot of the CDF and PDF as a figure
#-----------------------------------------------------------------
def make_figure_aux(bfile,fig):
    fig.savefig(bfile)
#enddef

def save_figure(last_bdir, iso, fig):
    bdir    = os.path.join(last_bdir,'figs')
    bfile   = os.path.join(bdir,'z_cum_stat_'+iso.lower()+'.png')
    make_figure_aux(bfile, fig)
#enddef

#-----------------------------------------------------------------
#create the figure plot
```

```
#-------------------------------------------------------------------
def plot_fig(fig, iso, iso_results):
    ax1 = fig.add_subplot(111)
    ax2 = ax1.twinx() #make y2 axis
    ax1.set_xlabel('bestN')
    sigma_label = r'$\sigma$'
    bias_label  = r'$\delta_B$'
    ax1.set_ylabel(sigma_label)
    ax2.set_ylabel(bias_label)
    ax2.set_prop_cycle('color', plt.cm.Spectral(np.linspace(0,1,len(iso_results))))
       #make unique colors
    ax1.set_prop_cycle('color', plt.cm.Spectral(np.linspace(0,1,len(iso_results))))

    ftitle = iso.upper()

    ax1.set_title(ftitle)

    #                du          bias            ,other
    #iso_results = [[DU-iso,[[x,y],[x,y],...],[]],
    #                [DU-iso,[[x,y],[x,y],...],[]],
    #                ....
    #]

    for duiso in iso_results:
        #[DU-iso,[[x,y],[x,y],...],[]]
        bias_set = duiso[1]
        xs = [x[0] for x in bias_set]
        ys = [x[1] for x in bias_set]


        ax2.plot(xs,ys,'+-',label=duiso[0])

        sig_set  = duiso[2]
        xs = [x[0] for x in sig_set]
        ys = [x[1] for x in sig_set]
        ax1.plot(xs,ys,'.-',label=duiso[0])

    #endfor

    ax1.legend(loc="upper left", fontsize="x-small", ncol=1, bbox_to_anchor=(1.15,1)
        , title=sigma_label)
    #ax2.legend(loc="upper left", fontsize="x-small", ncol=2, bbox_to_anchor=(0,1),
        title=bias_label)


    fig.tight_layout()

    return fig
#enddef

def make_figure_top(last_bdir, iso, iso_results):
```

```
    fig = plt.figure()

    fig = plot_fig(fig, iso, iso_results)
    save_figure(last_bdir, iso, fig)

    plt.clf()
    plt.close('all')
#enddef


#-----------------------------------------------------------------------
#get info in each location
#-----------------------------------------------------------------------
def locprocess(type, iso, loc, l_bdirs, args):
    l_bias = []
    l_sig  = []
    for bdir in l_bdirs:
        abnum = bdir.split('_')
        bnum  = float(abnum[-1])

        #get bias
        bias_dir = os.path.join(bdir,'bias')
        bias_f   = os.path.join(bias_dir,'bias_' + loc.lower() + '_' + iso.lower() +
            '.txt')
        if os.path.isfile(bias_f):
            with open(bias_f,'r') as fh:
                bias = float(fh.readline())
            #endwith
            l_bias.append((bnum,bias))
        #else:
            #file doesn't exist
        #endif

        #get stat (mu, sig)
        stat_dir = os.path.join(bdir,'stat')
        stat_f   = os.path.join(stat_dir,'stat_' + loc.lower() + '_' + iso.lower() +
            '.txt')
        if os.path.isfile(stat_f):
            with open(stat_f,'r') as fh:
                mu      = float(fh.readline())
                abs_sig = float(fh.readline())
                rel_sig = abs_sig/mu
            #endwith
            l_sig.append((bnum,rel_sig))
        #else:
            #file doesn't exist
        #endif

    #endfor
    key = loc.lower()
```

```
    return [key, l_bias, l_sig]
#enddef

#-------------------------------------------------------------------------------
# each isotope in each location
#-------------------------------------------------------------------------------
def isoprocess2(type, iso, locs, l_bdirs, args):
    print("=======================================")
    print(iso)

    iso_results = []
    for loc in locs:
        loc     = db_info.translate_wits_to_pa(loc)
        results = locprocess(type, iso, loc, l_bdirs, args)
        iso_results.append(results)
    #endfor

    #done with this iso, make the figure
    last_bdir = l_bdirs[-1]
    make_figure_top(last_bdir, iso, iso_results)

    print("---------------------------------")

#enddef


#-------------------------------------------------------------------------------
#run in parallel (won't work if toshow is True b/c slow
#-------------------------------------------------------------------------------
def multiprocess(type, locs, isos, l_bdirs, args):
    #preparing for multiprocessing
    iters = []
    for iso in isos:
        iters.append((type, iso, locs, l_bdirs, args))
    #endfor

    #run all the isos in parallel
    with multiprocessing.Pool(None) as pool: #None)
        pool.starmap(isoprocess2,iters)
    #endwith
#enddef

#-------------------------------------------------------------------------------
#data/best_i location
#-------------------------------------------------------------------------------
def get_best_dirs():
    pdir       = os.path.dirname(os.getcwd())
    subdirname = 'best'

    l_bdirs = []
    i = 0
```

```
    while True:
        subdir = os.path.join(pdir,os.path.join('data',subdirname+'_'+str(i)))  #
            best_i: best_0, best_1, ...
        if os.path.isdir(subdir):
            l_bdirs.append(subdir)
            i = i + 1
        else: #subdir doesn't exist, so at the end
            break
        #endif
    #endwhile

    return l_bdirs
#enddef

#-----------------------------------------------------------------------
#get isos (all same in each loc) - (for the last best dir)
#-----------------------------------------------------------------------
def get_isos(locs, l_bdirs):
    loc       = locs[0]

    last_bdir = l_bdirs[-1]
    l_isos    = []
    iso_act   = None

    #location dump file
    dtdir  = os.path.join(last_bdir,'dump')
    dtdir  = os.path.join(os.path.join(os.path.dirname(os.path.dirname(last_bdir)),'
        db'),'dump_best')

    isos = db_info.read_dump(dtdir,'dump_tmp_iso.txt')

    """
    dtfile = os.path.join(dtdir,'dump_tmp_iso.txt')
    if os.path.isfile(dtfile):
        with open(dtfile) as dfileo:
            isos = json.load(dfileo)
        #endwith

    else:
        print(dtfile + " not found")
        sys.exit()
    #endif
    """

    return isos
#enddef

#-----------------------------------------------------------------------
#main code to run the stats process (so can call from zanalysis)
#-----------------------------------------------------------------------
def zstats_main(type, locs, args):
```

```
    if type == 'best':
        l_bdirs = get_best_dirs()
        isos    = get_isos(locs, l_bdirs)
        multiprocess(type, locs, isos, l_bdirs, args)
    #else no stats to do
    #endif
#enddef

#-------------------------------------------------------------------------
#Main code to read the bias and stats files, in each bestN
#then create a figure of each isotope in each location over N
#two y-axis: uncertainty and bias
#-------------------------------------------------------------------------
def main(loc,args):
    type = 'best'
    #to_show    = args.show
    #to_zip     = args.zip
    #to_avg     = args.avg

    time_start = time.time()


    #get list of isotopes (multiprocess each of these)

    #loop over bdirs, get bias and stat
    #read file for each istope
    #put it into a list, then plot it,
    #then save it: fig_iso_per_n.png

    #loop over l_bdirs
    #

    ######################
    palocs = db_info.get_dump_palocs(type)
    if palocs == "":
        return
    #endif

    if loc == 'ALL':
        locs = palocs
    elif loc in palocs:
        locs = [loc]
    else:
        print(loc,"not a PA location")
        return
    #endif

    zstats_main(type, locs, args)


    ######################
```

```
    time_stop = time.time()
    time_diff = time_stop-time_start

    print("This took: ", "{:.2f}".format(time_diff),"secs")
    print("           ","{:.2f}".format(time_diff/60),"mins")
#enddef

#-------------------------------------------------------------------------------
#analyze the dump files to create the histogram, pdf, and cdf files
#for the E-area pa
#-------------------------------------------------------------------------------
if __name__ == '__main__':
    arg_parser = argparse.ArgumentParser(description='Analyze the bias and stats
        files over time to see trends')
    #group      = arg_parser.add_mutually_exclusive_group()
    """
    msg0 = 'the TYPE of dump file to read (base|best|test):'
    msg1 = '    base is the base case - estimated uncertainty, '
    msg2 = '    best is the best case - use uncertainty, '
    msg3 = '    test is the test case - same as best '
    msg = msg0+msg1+msg2+msg3
    arg_parser.add_argument('Type',
                            metavar='type',
                            type=str,
                            choices=['base','best','test'],
                            help=msg)
    """
    arg_parser.add_argument('Location',
                            metavar='loc',
                            type=str,
                            help='the LOCATION to analyze (all|NR0|CIG1|...)')
    """
    arg_parser.add_argument('Isotopes',
                            metavar='iso',
                            type=str,
                            choices=['all','short'],
                            help='the ISOTOPES to analyze (all|short)')
    group.add_argument(     '--show',
                            action="store_true",
                            help='show the pdf-cdf plot')
    group.add_argument(     '--nofig',
                            action="store_true",
                            help='do not make the pdf-cdf plot')
    arg_parser.add_argument('--zip',
                            action="store_true",
                            help='zip the TYPE directory, so it is fast to upload to
                                HPC')
    arg_parser.add_argument('--avg',
                            action="store_true",
                            help='make output files that are the average DU type (ET
```

```
                                    , ST, ILV, LAW, NR')
    """
    args  = arg_parser.parse_args()

    #type  = args.Type
    loc   = args.Location
    #isos  = args.Isotopes

    main(loc.upper(),args)
#endif
```

Listing 8: zppp

```
#Print Priority Packages
#Read the DUMP FILE and get the packages of interest

import argparse
import itertools
import json
import os
import sys
import time

import db_info
import unc_aux



#see zanalysis
def get_cut_info(l_cuts, d_waste_streams):
    tcutact = 0
    cinfo   = []
    for cut in l_cuts:
        cid  = cut[0]

        el   = cut[1]
        lact = list(el.keys())
        lunc = list(el.values())
        act  = float(lact[0])
        unc  = float(lunc[0])

        ws   = cut[2]
        swef = cut[3]

        if ws in d_waste_streams.keys():
            d_waste_streams[ws][0] = d_waste_streams[ws][0] + act
        else:
            d_waste_streams[ws] = [act, swef]
        #endif
    #endfor
    return d_waste_streams
```

```
#enddef

#see zanalysis get_act3
def get_ws_act(l_act_unc, d_waste_streams):
    [l_cuts, l_inner_cntrs] = l_act_unc
    d_waste_streams = get_cut_info(l_cuts, d_waste_streams)

    if l_inner_cntrs == []:
        pass
    else:
        for cntr in l_inner_cntrs:
            d_waste_streams  = get_ws_act(cntr[1], d_waste_streams)
        #endfor
    #endif

    return d_waste_streams
#enddef


#convert to [[act,ws,swe],...]
def ws_list(d_waste_streams):
    wsl = []
    for k,v in d_waste_streams.items():
        v.append(k)
        wsl.append(v)
    #endfor
    return wsl
#enddef

#-------------------------------------------------------------------------------
#process the dump file
#-------------------------------------------------------------------------------
def proc_dfile(loc, iso, l_packages, args, all_d_waste_streams):
    to_filter      = args.filter

    num_dtc   = 0
    num_stc   = 0
    num_cbp   = 0
    num_rad   = 0
    num_other = 0

    #waste_streams = set()
    waste_streams = []

    #print('{0:<4}'.format(' '),'{0:<16}'.format('pkg'),'{0:>9}'.format('sum short')
        ,'{0:<30}'.format('calc type(s)'),'waste stream(s)')

    #-----------------------------------------
    #MAIN WORK HERE
    #-----------------------------------------
    d_waste_streams = {}
```

```
#sort the l_packages by the activity
l_packages.sort(key = lambda x: x[2],reverse=True)
for d in l_packages:  #d = pkg name, l_act_unc, wits_activity, true_activity
    pack_act  = d[2]
    d_waste_streams = get_ws_act(d[1], d_waste_streams)
#endfor

#-----------------------------------------------
#MAKE THE OUTPUT PRETTY
#-----------------------------------------------
high_ws_act = 0
prev_ws_act = 0
filtered = "unfiltered"

oom    = 1
plimit = 99
i      = 0

wsl = ws_list(d_waste_streams)  #convert dict to [[act,ws,swe],...]
wsl.sort(key = lambda x: x[0], reverse=True)
for w in wsl:
    ws_act = w[0]
    ws     = w[2]
    swef   = w[1]
    if i == 0:
        high_ws_act = ws_act
    #endif

    #add this to the complete list
    if ws in all_d_waste_streams.keys():
        all_d_waste_streams[ws][0] = all_d_waste_streams[ws][0] + ws_act
    else:
        all_d_waste_streams[ws] = [ws_act, swef]
    #endif

    #     act,  wsv   swe
    print("{:.2e}  ".format(ws_act), "{0:<24}".format(ws), swef)

    if to_filter:
        if prev_ws_act / ws_act > 10**oom:
            filtered = "filtered by prev "+str(oom)+" OOM"
            break
        #endif
        if high_ws_act / ws_act > 10**oom:
            filtered = "filtered by high "+str(oom)+" OOM"
            break
        #endif
        if i > plimit:
            filtered = "filtered by max pkgs per DU ("+ str(plimit+1)+")"
            break
        #endif
```

```
        #endif

        prev_ws_act = ws_act
        i           = i + 1
    #endfor
    print(filtered)
    print("-"*70)

    """
    d_waste_streams = {i:xwaste_streams.count(i) for i in xwaste_streams}

    sorted_d_waste_streams = {k: v for k, v in sorted(d_waste_streams.items(), key=
        lambda item: item[1], reverse=True)}
    for k, v in sorted_d_waste_streams.items():
        print('{0:>4}'.format(str(v)), k)
    print("_"*20)
    """

    return all_d_waste_streams #num_dtc, num_stc, num_cbp, num_rad, num_other,
        waste_streams
#enddef


#-------------------------------------------------------------------------------
#main code to read the dump file for each location and count info and print results
#-------------------------------------------------------------------------------
def main(type, loc, args):
    time_start = time.time()

    tdir  = db_info.get_data_subdir(type,'input')
    dtdir = os.path.join(tdir,'dump')

    palocs = db_info.get_dump_palocs(type)
    if palocs == "":
        return
    #endif

    if loc == 'ALL':
        locs = palocs
    elif loc in palocs:
        locs = [loc]
    else:
        print(loc,"not a PA location")
        return
    #endif

    """
    num_dtc   = 0
    num_stc   = 0
    num_cbp   = 0
    num_rad   = 0
```

```
num_other = 0
"""

isos_from_file = args.file
num            = args.num

all_d_waste_streams = {}
for loc in locs:
    dtfile = os.path.join(dtdir,'dump_'+loc.lower()+'.txt')
    if os.path.isfile(dtfile):
        print("="*20)
        print(loc)
        print("-"*20)

        color = args.file_filter #for filtering
        #The isos to do and the order in which to do them
        isos_to_do = db_info.get_isos_to_do(isos_from_file, color, '', loc, type
            )

        j = 1
        for iso, sof in isos_to_do:
            if iso in ['AVGET','AVST']:
                pass
            else:
                #READ dump OBJECT
                with open(dtfile) as dfileo:
                    for line in dfileo:
                        aline      = line.split(',',1)
                        diso       = aline[0]
                        if iso == diso:
                            l_packages = json.loads(aline[1]) #load it after
                                knowing this is the one to do

                            print(iso, '-', "{0:.2%}".format(sof))
                            #process dump file (real work here)
                            all_d_waste_streams = proc_dfile(loc, iso,
                                l_packages, args, all_d_waste_streams)
                            j = j + 1
                            if j > num:
                                break
                            #endif
                        #endif
                    #endfor
                #endwith
                if j > num:
                    break
                #endif
            #endif
        #endfor
    #endif
    #ldtc, lstc, lcbp, lrad, lother, lws = proc_dfile(loc, iso_act, args)
```

```
    """
    #count items
    num_dtc   = num_dtc + ldtc
    num_stc   = num_stc + lstc
    num_cbp   = num_cbp + lcbp
    num_rad   = num_rad + lrad
    num_other = num_other + lother

    waste_streams.extend(lws)
    """
#endfor

print("="*70)
print("Sum of all waste streams, sorted by total activity")
print("-"*70)

"""
print("--------------------------------------------------------------------------------")
print("WasteCut calc_type and number:")
print("          DTC:",   num_dtc)
print("          STC:",   num_stc)
print("          CBP:",   num_cbp)
print("          RAD:",   num_rad)
print("        OTHER:", num_other)
print("")
print("--------------------------------------------------------------------------------")
print(" Num","{0:<24}".format("WasteStream_Version") , "WCF_FormNum_Revision")

d_waste_streams = {i:waste_streams.count(i) for i in waste_streams}

sorted_d_waste_streams = {k: v for k, v in sorted(d_waste_streams.items(), key=
    lambda item: item[1], reverse=True)}
for k, v in sorted_d_waste_streams.items():
    print('{0:>4}'.format(str(v)), k)
"""

#-----------------------------------------------
#MAKE THE OUTPUT PRETTY
#-----------------------------------------------
high_ws_act = 0
prev_ws_act = 0
filtered = "unfiltered"

oom    = 1
plimit = 99
i      = 0

wsl = ws_list(all_d_waste_streams)  #convert dict to [[act,ws,swe],...]
wsl.sort(key = lambda x: x[0], reverse=True)
for w in wsl:
    ws_act = w[0]
```

```
        ws      = w[2]
        swe     = w[1]
        if i == 0:
            high_ws_act = ws_act
        #endif

        #     act,  wsv   swe
        print("{:.2e}  ".format(ws_act), "{0:<24}".format(ws), swe)

        """
        if to_filter:
            if prev_ws_act / ws_act > 10**oom:
                filtered = "filtered by prev "+str(oom)+" OOM"
                break
            #endif
            if high_ws_act / ws_act > 10**oom:
                filtered = "filtered by high "+str(oom)+" OOM"
                break
            #endif
            if i > plimit:
                filtered = "filtered by max pkgs per DU ("+ str(plimit+1)+")"
                break
            #endif
        #endif

        prev_ws_act = ws_act
        """
        i           = i + 1
    #endfor
    print(filtered)
    print("-"*70)

    time_stop = time.time()
    time_diff = time_stop-time_start

    print("This took: ", time_diff, "secs")
    print("           ",time_diff/60,"mins")
#enddef


#-----------------------------------------------------------------------
#prints the packages with priority in each location
#-----------------------------------------------------------------------
if __name__ == '__main__':
    arg_parser = argparse.ArgumentParser(description='Print the packages with
       priority in each DU')
    msg0 = 'the TYPE of dump file to read (base|best):'
    msg1 = '    base is the base case - estimated uncertainty, '
    msg2 = '    best is the best case - use uncertainty, '
    msg = msg0+msg1+msg2
    arg_parser.add_argument('Type',
```

```
                                metavar='type',
                                type=str,
                                choices=['base','best'],
                                help=msg)
    arg_parser.add_argument('Location',
                                metavar='loc',
                                type=str,
                                help='the LOCATION to print priority packages [all|NR0|
                                    CIG1|...]')
    arg_parser.add_argument('--file',
                                action="store_true",
                                help='read priority file')
    arg_parser.add_argument('--filter',
                                action="store_true",
                                help='filter the priority list by order-of-magnitude')
    arg_parser.add_argument('--num',
                                type=float,default=1e99,
                                help='number of isotopes to print')
    arg_parser.add_argument('--file_filter',
                                type=str,
                                choices=['red','yellow','blue','white'],
                                help='show isotopes >= to this value: 1 red, 0.01 yellow
                                    , 0.001 blue, 0 white')

    args  = arg_parser.parse_args()
    type  = args.Type
    loc   = args.Location

    main(type,loc.upper(),args)
#endif
```

Listing 9: zpartials

```
def main(values_first,values_last,deltas_first,deltas_last):

    col = 'B'
    sum_val_range = 'SUM('+col+values_first+':'+col+values_last+')'

    val_delta_diff1 = int(int(values_first) - int(deltas_first))
    val_delta_diff2 = int(int(values_last) - int(deltas_last))

    if val_delta_diff1 != val_delta_diff2:
        print("differ")
        return
    #endif

    vd_diff = val_delta_diff1
```

```
    results = []

    #inital deltas
    ideltas = list(range(int(deltas_first)+1,int(deltas_last)+1))

    for j in range(int(values_first),int(values_last)+1):
        delta   = j-vd_diff
        deltam1 = delta - 1
        if deltam1 < int(deltas_first):
            deltas = ideltas
        else:
            deltas = [deltam1 if i==delta else i for i in deltas]
        #endif

        delta = str(j-vd_diff)

        res = '=SQRT('

        res = res + '(('+ sum_val_range + '-'+col+str(j)+')/(' + sum_val_range +
            '^2)*'+col+delta+')^2+'

        for i in deltas:
            res = res + '(-'+col+ str(j) + '/('+sum_val_range+'^2)*'+col+str(i)+')
                ^2+'
        #endfor
        res = res[:-1]
        res = res + ')'
        results.append(res)
    #endfor

    for r in results:
        print(r)

#enddef


#-----------------------------------------------------------------------------
#create the partial text for excel uncertainties
#-----------------------------------------------------------------------------
if __name__ == '__main__':
    values_first = input('Enter the first row in the Values range:')
    values_last = input('Enter the last row in the Values range:')

    deltas_first = input('Enter the first row in the Deltas range:')
    deltas_last = input('Enter the last row in the Deltas range:')

    main(values_first,values_last,deltas_first,deltas_last)
#endif
```

Appendix C Prioritized Waste Streams

CIG1

I−129 − 100.00%

4.17e−06	105X6025−LLW_v.0	SWEWSCF20030052_rev.0
3.18e−06	10502105−LLW_v.0	SWEWSCF20020037_rev.0
1.03e−06	105X6029−LLW_v.0	SWEWSCF20030065_rev.1
6.59e−07	105X6026−LLW_v.0	SWEWSCF20030064_rev.1
6.30e−07	HMWST00001−LLW_v.0	SWEWSCF20000006_rev.1
5.96e−07	105X6030−LLW_v.0	SWEWSCF20040029_rev.0
5.08e−07	105L6034−LLW_v.0	SWEWSCF20050048_rev.0
4.81e−07	00686−LLW_v.1	SWEWSCF19970038_rev.1
3.12e−07	221S00016−LLW_v.1	SWEWSCF20010095_rev.1

filtered by high 1 OOM

TC−99 − 74.90%

4.61e−03	00689−LLW_v.2	SWEWSCF19970047_rev.2
5.62e−04	105X6025−LLW_v.0	SWEWSCF20030052_rev.0
1.84e−04	ASSAYB25−LLW_v.0	SWEWSCF20030032_rev.0

filtered by high 1 OOM

NI−63 − 1.99%

8.69e−02	NAVY1−LLW_v.0	None
5.28e−02	00503−LLW_v.0	None
3.84e−02	00486−LLW_v.0	None
1.16e−02	00483−LLW_v.0	None
3.34e−03	HMWST00001−LLW_v.0	SWEWSCF20000006_rev.1

filtered by high 1 OOM

NI−59 − 1.89%

5.80e−04	NAVY1−LLW_v.0	None
3.32e−04	00486−LLW_v.0	None
3.03e−04	00503−LLW_v.0	None
2.89e−04	HMWST00001−LLW_v.0	SWEWSCF20000006_rev.1
9.60e−05	H3DECAY−LLW_v.0	SWEWSCF20070006_rev.0
7.74e−05	00483−LLW_v.0	None
5.06e−05	105X2005−LLW_v.0	SWEWSCF19960033_rev.2

filtered by high 1 OOM

ETRENCH1

H−3 − 100.00%

4.18e−01	00475−LLW_v.0	SWE−WSCF−1996−0011
2.52e−01	00460−LLW_v.0	SWE−WSCF−1995−0141
1.28e−01	105−Z−1001−LLW_v.1	SWEWSCF19980078_rev.2
1.08e−01	00100−LLW_v.0	None
1.06e−01	00617−LLW_v.2	SWEWSCF19960020_rev.4
9.81e−02	00618−LLW_v.0	SWE−WSCF−1996−0023

```
7.02e-02    FAGW-0002-LLW_v.1          SWEWSCF19990030_rev.1
6.42e-02    00617-LLW_v.0              SWE-WSCF-1996-0020_rev.2
5.28e-02    00618-LLW_v.2              SWEWSCF19960023_rev.3
4.46e-02    ETF-001-125-LLW_v.0        SWEWSCF19990026_rev.1
4.16e-02    NAVY1-LLW_v.0              None
filtered by high 1 OOM
```

==========

ETRENCH2

```
NP-237 - 100.00%
6.27e-03    FHLABSTC-LLW_v.2           SWEWSCF20080003_rev.2
3.88e-03    FHLABSTC-LLW_v.1           SWEWSCF20080003_rev.1
3.32e-03    SRNLJCW-LLW_v.1            SWEWSCF20060036_rev.1
2.40e-03    FHLABSTC-LLW_v.0           SWEWSCF20080003_rev.0
2.37e-03    FHLABSTC-LLW_v.3           SWEWSCF20080003_rev.3
1.48e-03    ASSAYBYPACKAGE-LLW_v.4     SWEWSCF20100009_rev.3
1.22e-03    772F022B-LLW_v.4           SWEWSCF19980062_rev.6
8.60e-04    SRNLJCW-LLW_v.0            SWEWSCF20060036_rev.0
7.64e-04    SRNLJCW-LLW_v.4            SWEWSCF20060036_rev.4
6.88e-04    772FHLGEN-LLW_v.2          SWEWSCF20140001_rev.2
5.25e-04    HBLTRUTOLLW01-LLW_v.0      SWEWSCF20070030_rev.1
filtered by high 1 OOM
```

==========

ETRENCH3

```
NP-237 - 100.00%
2.23e-03    221H0021-LLW_v.7           SWEWSCF19970052_rev.8
1.49e-03    SRNLJCW-LLW_v.4            SWEWSCF20060036_rev.4
1.42e-03    221H0021-LLW_v.6           SWEWSCF19970052_rev.7
1.34e-03    772FHLGEN-LLW_v.1          SWEWSCF20140001_rev.1
8.49e-04    772FHLGEN-LLW_v.2          SWEWSCF20140001_rev.2
4.60e-04    ETF-001-125-LLW_v.7        SWEWSCF19990026_rev.9
4.24e-04    221H0021-LLW_v.5           SWEWSCF19970052_rev.6
2.68e-04    221H0036-LLW_v.5           SWEWSCF20030060_rev.5
2.57e-04    772FHLGEN-LLW_v.0          SWEWSCF20140001_rev.0
2.00e-04    221H0021-LLW_v.8           SWEWSCF19970052_rev.9
filtered by high 1 OOM
```

==========

ILV1

```
I-129 - 100.00%
2.71e-04    211F813SLUDGE-LLW_v.0      SWEWSCF20070001_rev.1
1.42e-04    ETF0032-LLW_v.1            SWEWSCF20090003_rev.2
3.07e-05    FCXBOX157-LLW_v.0          SWEWSCF20110040_rev.0
1.25e-05    105L9005-LLW_v.0           SWEWSCF20070021_rev.0
filtered by high 1 OOM
```

==========

```
LAW1
_____________________
I-129 - 100.00%
5.25e-05    FAREANONROUTINE-LLW_v.0    SWEWSCF20070035_rev.0
1.99e-05    NAVY1-LLW_v.0              None
1.83e-05    ASSAYBYPACKAGE-LLW_v.7     SWEWSCF20100009_rev.6
1.33e-05    BAL001A-LLW_v.0            SWEWSCF20050007_rev.0
7.88e-06    STP055LLW1-LLW_v.0         SWEWSCF19970054_rev.0
5.58e-06    772FHLGEN-LLW_v.1          SWEWSCF20140001_rev.1
5.38e-06    00689-LLW_v.1              SWEWSCF19970047_rev.1
3.44e-06    221F0006HD-LLW_v.0         SWEWSCF19970048_rev.1
filtered by high 1 OOM
_____________________________________________________________________________

=====================
NR0
_____________________
=====================
NR1
_____________________
TC-99 - 100.00%
1.57e-01    NAVY1-LLW_v.0              None
1.06e-02    00473-LLW_v.0              None
filtered by prev 1 OOM
_____________________________________________________________________________
H-3 - 42.70%
4.50e+01    NAVY1-LLW_v.0              None
unfiltered
_____________________________________________________________________________

=====================
SLIT
_____________________
H-3-F - 100.00%
3.87e+00    00564-LLW_v.0              SWE-WSCF-1996-0060
unfiltered
_____________________________________________________________________________
H-3 - 14.50%
2.75e-01    105X3001-LLW_v.0           SWEWSCF19990086_rev.0
1.28e-01    00475-LLW_v.0              SWE-WSCF-1996-0011
8.57e-02    FAGW-0001-LLW_v.3          SWEWSCF19970030_rev.5
4.55e-02    DPCONCRETE02-LLW_v.0       SWEWSCF20020079_rev.0
3.61e-02    FAGW-0001-LLW_v.0          SWEWSCF19970030_rev.0
3.30e-02    FAGW-0004-LLW_v.0          SWEWSCF19980073_rev.1
3.29e-02    00613-LLW_v.0              SWE-WSCF-1996-0068
2.91e-02    00617-LLW_v.0              SWE-WSCF-1996-0020_rev.2
2.89e-02    00618-LLW_v.0              SWE-WSCF-1996-0023
2.52e-02    105X3501-LLW_v.0           SWEWSCF20000021_rev.0
filtered by high 1 OOM
_____________________________________________________________________________

=====================
SLIT14
_____________________
```

```
TC-99-H - 100.00%
3.19e-03    770URXCERCLA-LLW_v.0       SWEWSCF20100047_rev.0
unfiltered
-------------------------------------------------------------------------------
TC-99 - 13.40%
2.97e-03    MCU00004-LLW_v.0           SWEWSCF20110045_rev.0
2.24e-03    728NDEBRIS-LLW_v.0         SWEWSCF20190004_rev.0
2.23e-03    HTK00002-LLW_v.3           SWEWSCF20030014_rev.3
2.02e-03    MCU00004-LLW_v.1           SWEWSCF20110045_rev.1
1.87e-03    HTK00002-LLW_v.6           SWEWSCF20030014_rev.6
1.83e-03    FHW00001HTFJCW-LLW_v.7     SWEWSCF20120030_rev.3
1.28e-03    773-A VAR. CASK-LLW_v.7    SWEWSCF19980004_rev.7
1.13e-03    221S00024-LLW_v.1          SWEWSCF20120005_rev.1
1.10e-03    512S01701-LLW_v.1          SWEWSCF20170008_rev.1
1.02e-03    HTK00002-LLW_v.5           SWEWSCF20030014_rev.5
9.10e-04    221S00022-LLW_v.0          SWEWSCF20100050_rev.0
8.51e-04    HTK00002-LLW_v.7           SWEWSCF20030014_rev.7
7.65e-04    105L2004-LLW_v.2           SWEWSCF20040067_rev.2
7.61e-04    FHW00001FTFJCW-LLW_v.7     SWEWSCF20120036_rev.3
7.51e-04    MCU00003-LLW_v.0           SWEWSCF20100052_rev.0
6.01e-04    HTK00002-LLW_v.4           SWEWSCF20030014_rev.4
4.84e-04    MCU00004-LLW_v.2           SWEWSCF20110045_rev.2
3.15e-04    LTR60006CERCLA-LLW_v.0     SWEWSCF20120010_rev.0
2.87e-04    FHW00001HTFJCW-LLW_v.5     SWEWSCF20120030_rev.1
filtered by high 1 OOM
-------------------------------------------------------------------------------
I-129 - 7.67%
2.64e-06    MCU00004-LLW_v.0           SWEWSCF20110045_rev.0
2.63e-06    221H0042-LLW_v.1           SWEWSCF20070019_rev.1
2.34e-06    FHW00001HTFJCW-LLW_v.7     SWEWSCF20120030_rev.3
1.79e-06    MCU00004-LLW_v.1           SWEWSCF20110045_rev.1
1.75e-06    ETF-002-12-LLW_v.5         SWEWSCF19990035_rev.7
1.30e-06    221S00024-LLW_v.1          SWEWSCF20120005_rev.1
9.76e-07    FHW00001FTFJCW-LLW_v.7     SWEWSCF20120036_rev.3
8.14e-07    512S01701-LLW_v.1          SWEWSCF20170008_rev.1
7.83e-07    221F0002-LLW_v.6           SWEWSCF19950094_rev.9
6.68e-07    221S00022-LLW_v.0          SWEWSCF20100050_rev.0
5.26e-07    MCU00003-LLW_v.0           SWEWSCF20100052_rev.0
4.39e-07    772FHLGEN-LLW_v.0          SWEWSCF20140001_rev.0
3.82e-07    MCU00004-LLW_v.2           SWEWSCF20110045_rev.2
3.79e-07    221HBL-LLW_v.5             SWEWSCF20090032_rev.5
3.69e-07    NAVY1-LLW_v.0              None
2.39e-07    772F022B-LLW_v.6           SWEWSCF19980062_rev.9
filtered by high 1 OOM
-------------------------------------------------------------------------------
NI-59 - 2.45%
2.08e-02    NAVY1-LLW_v.0              None
8.35e-03    221H0040-LLW_v.1           SWEWSCF20070018_rev.1
2.12e-03    105CDISAREA-LLW_v.0        SWEWSCF20110028_rev.1
1.74e-03    5PDPCBBULKCERCLA-LLW_v.1   SWEWSCF20100038_rev.3
filtered by high 1 OOM
```

SLIT2

I−129−J − 100.00%

3.45e−04	FAGW−0001−LLW_v.3	SWEWSCF19970030_rev.5

unfiltered

I−129 − 45.60%

2.98e−06	00685−LLW_v.1	SWEWSCF19970039_rev.1
2.72e−06	FHW00001HTFJCW−LLW_v.0	None
1.94e−06	105X3512−LLW_v.0	SWEWSCF20020073_rev.0
1.57e−06	FHW00003HTF−LLW_v.0	SWEWSCF20030035_rev.0
1.32e−06	FHG02002−LLW_v.0	SWEWSCF20020025_rev.1
1.24e−06	FAGW−0002−LLW_v.1	SWEWSCF19990030_rev.1
1.05e−06	221F0006SOIL−LLW_v.1	SWEWSCF19980088_rev.1
9.15e−07	105X3513−LLW_v.0	SWEWSCF20020077_rev.0
7.19e−07	CIF99017−LLW_v.0	SWEWSCF19990038_rev.0
4.93e−07	FHW00001FTFJCW−LLW_v.0	SWEWSCF20010039_rev.0
4.72e−07	BGC00002−LLW_v.0	SWEWSCF20010005_rev.0
4.35e−07	FBI02001−LLW_v.0	SWEWSCF20020026_rev.0
4.25e−07	244H4005−LLW_v.0	SWEWSCF20010099_rev.0
4.17e−07	NAVY1−LLW_v.0	None
3.60e−07	244H4006−LLW_v.0	SWEWSCF20020056_rev.0
3.29e−07	00666−LLW_v.4	SWEWSCF19970022_rev.4
2.94e−07	FHW00001HTFJCW−LLW_v.1	SWEWSCF20010039_rev.1

filtered by high 1 OOM

TC−99 − 32.50%

7.07e−03	FAGW−0005−LLW_v.1	SWEWSCF19990005_rev.2
2.24e−03	FHW00001HTFJCW−LLW_v.0	None
1.99e−03	00523−LLW_v.3	SWEWSCF19960027_rev.4
1.64e−03	00689−LLW_v.2	SWEWSCF19970047_rev.2
1.47e−03	FHW00003HTF−LLW_v.0	SWEWSCF20030035_rev.0
7.07e−04	PAD02031−LLW_v.0	SWEWSCF20030048_rev.1
6.84e−04	ETF−002−12−LLW_v.0	SWEWSCF19990035_rev.1

filtered by high 1 OOM

I−129−G − 16.90%

2.79e−05	FAGW−0004−LLW_v.2	SWEWSCF19980073_rev.3
2.36e−05	FAGW−0004−LLW_v.1	SWEWSCF19980073_rev.2

unfiltered

I−129−D − 12.10%

4.32e−03	FAGW−0005−LLW_v.1	SWEWSCF19990005_rev.2
8.78e−05	FAGW−0005−LLW_v.0	SWEWSCF19990005_rev.0

filtered by prev 1 OOM

I−129−H − 5.76%

1.18e−04	HAGW−0004A−LLW_v.0	SWEWSCF19980051_rev.1

unfiltered

NI−63 − 1.76%
4.76e+00 NAVY1−LLW_v.0 None
8.83e−01 00473−LLW_v.0 None
5.36e−01 00568−LLW_v.0 None
3.01e−01 NAVY2−LLW_v.0 None
filtered by high 1 OOM

NI−59 − 1.57%
3.10e−02 NAVY1−LLW_v.0 None
3.01e−03 NAVY2−LLW_v.0 None
filtered by prev 1 OOM

==========

SLIT3

TC−99 − 100.00%
1.21e−02 244H2004−LLW_v.1 SWEWSCF20000051_rev.2
3.09e−03 ZSS0043−LLW_v.0 SWEWSCF20040031_rev.0
2.15e−03 ETF−00003−1−LLW_v.2 SWEWSCF19940014_rev.4
1.69e−03 00523−LLW_v.3 SWEWSCF19960027_rev.4
1.57e−03 221H0036−LLW_v.0 SWEWSCF20030060_rev.0
1.02e−03 PAD02031−LLW_v.0 SWEWSCF20030048_rev.1
filtered by high 1 OOM

I−129 − 88.90%
1.20e−05 FAGW−0002−LLW_v.1 SWEWSCF19990030_rev.1
5.15e−06 FHG02002−LLW_v.0 SWEWSCF20020025_rev.1
4.34e−06 FAGW−0001−LLW_v.0 SWEWSCF19970030_rev.0
3.37e−06 HBLBLACKBOX−LLW_v.0 SWEWSCF20030073_rev.0
2.46e−06 221S−00011−LLW_v.1 SWEWSCF19980049_rev.2
1.50e−06 221FB25HOTSIDE−LLW_v.0 SWEWSCF19990134_rev.0
1.15e−06 00685−LLW_v.1 SWEWSCF19970039_rev.1
filtered by high 1 OOM

I−129−C − 27.70%
1.64e−02 ETF−00003−1−LLW_v.2 SWEWSCF19940014_rev.4
unfiltered

I−129−J − 6.31%
3.42e−05 FAGW−0001−LLW_v.3 SWEWSCF19970030_rev.5
unfiltered

NP−237 − 5.33%
5.63e−03 772F037O−LLW_v.0 SWEWSCF20020023_rev.0
2.88e−03 FHLABSTC−LLW_v.2 SWEWSCF20080003_rev.2
1.58e−03 772F023C−LLW_v.0 SWEWSCF19980063_rev.0
1.42e−03 00281−LLW_v.0 SWE−WSCF−1995−0036
9.96e−04 772F039Q−LLW_v.0 SWEWSCF20040004_rev.0
8.24e−04 773AC−PROCESS−LLW_v.2 SWEWSCF19990022_rev.2
7.92e−04 221H0036−LLW_v.0 SWEWSCF20030060_rev.0

```
5.57e-04    221H0021-LLW_v.1          SWEWSCF19970052_rev.1
filtered by high 1 OOM
--------------------------------------------------------------------------------

NI-59 - 1.18%
4.62e-03    NAVY1-LLW_v.0             None
2.64e-03    00427-LLW_v.1             SWEWSCF19950075_rev.2
1.14e-03    247F10032004-LLW_v.0      SWEWSCF20040015_rev.0
1.12e-03    ETF-00003-1-LLW_v.2       SWEWSCF19940014_rev.4
9.01e-04    221H0021-LLW_v.4          SWEWSCF19970052_rev.4
6.95e-04    221H0021-LLW_v.0          SWEWSCF19970052_rev.0
3.57e-04    00364-LLW_v.1             SWEWSCF19950045_rev.3
filtered by high 1 OOM
--------------------------------------------------------------------------------

================================================================================
SLIT4
--------------------------------------------------------------------------------

H-3 - 100.00%
7.93e+00    SEGSLUDGE3-LLW_v.1        SWEWSCF19990032_rev.1
1.59e-01    FDDTRITIUM-LLW_v.0        SWEWSCF20020030_rev.0
filtered by prev 1 OOM
--------------------------------------------------------------------------------

================================================================================
SLIT5
--------------------------------------------------------------------------------

TC-99 - 100.00%
1.48e-02    77710A0001-LLW_v.0        SWEWSCF20050001_rev.1
6.96e-03    221H0034-LLW_v.0          SWEWSCF20030058_rev.0
4.22e-03    221H0036-LLW_v.0          SWEWSCF20030060_rev.0
3.51e-03    105L2004-LLW_v.0          SWEWSCF20040067_rev.0
3.36e-03    HIGHLEVELCEL06-LLW_v.1    SWEWSCF20060020_rev.2
2.73e-03    ZSS0043-LLW_v.0           SWEWSCF20040031_rev.0
2.60e-03    ZSS001125-LLW_v.1         SWEWSCF20020018_rev.1
8.92e-04    100ROOM-LLW_v.0           SWEWSCF20030069_rev.0
filtered by high 1 OOM
--------------------------------------------------------------------------------

I-129 - 94.20%
3.28e-05    100TRITIUM-LLW_v.0        SWEWSCF20050045_rev.0
8.36e-06    HIGHLEVELCEL06-LLW_v.1    SWEWSCF20060020_rev.2
4.83e-06    SOL99001CERCLA-LLW_v.0    SWEWSCF20010031_rev.0
3.99e-06    FAGW-0002-LLW_v.1         SWEWSCF19990030_rev.1
1.56e-06    BOX332005-LLW_v.0         SWEWSCF20050050_rev.0
filtered by high 1 OOM
--------------------------------------------------------------------------------

NI-63 - 1.71%
1.57e+00    105L9003-LLW_v.0          SWEWSCF20040051_rev.0
1.14e+00    NAVY1-LLW_v.0             None
3.55e-03    109RBASIN-LLW_v.0         SWEWSCF20060013_rev.0
filtered by prev 1 OOM
--------------------------------------------------------------------------------

================================================================================
SLIT6
```

I−129 − 100.00%

3.79e−05	230HBGI–LLW_v.0	SWEWSCF20060039_rev.0
1.07e−05	211FTANKOSF–LLW_v.0	SWEWSCF20060059_rev.0
6.86e−06	HIGHLEVELCEL06–LLW_v.1	SWEWSCF20060020_rev.2
5.99e−06	FCANBLBX06–LLW_v.0	SWEWSCF20060055_rev.0
3.73e−06	211FTANKOSF–LLW_v.1	SWEWSCF20060059_rev.1

filtered by high 1 OOM

TC−99 − 31.60%

8.58e−03	230HBGI–LLW_v.0	SWEWSCF20060039_rev.0
2.76e−03	HIGHLEVELCEL06–LLW_v.1	SWEWSCF20060020_rev.2
2.16e−03	221FOXIDE–LLW_v.0	SWEWSCF20040032_rev.0
1.60e−03	ZSS001125−LLW_v.1	SWEWSCF20020018_rev.1
1.43e−03	105L2004–LLW_v.1	SWEWSCF20040067_rev.1
1.40e−03	FHW00001HTFJCW–LLW_v.1	SWEWSCF20010039_rev.1
8.17e−04	221H0036–LLW_v.0	SWEWSCF20030060_rev.0

filtered by high 1 OOM

SLIT7

H–3 − 100.00%

7.64e−02	4202DTRITCERCLA–LLW_v.0	SWEWSCF20110001_rev.0
4.11e−02	105RDISAREA–LLW_v.1	SWEWSCF20090015_rev.1
2.80e−02	211FPIPRMCERCLA–LLW_v.0	SWEWSCF20080028_rev.0
2.23e−02	TRITSORT01–LLW_v.0	SWEWSCF19980006_rev.0
2.09e−02	00618−LLW_v.0	SWE–WSCF−1996−0023
2.04e−02	105PDISAREA–LLW_v.0	SWEWSCF20080004_rev.0
1.69e−02	00100−LLW_v.0	None
1.62e−02	105PDISAREA–LLW_v.2	SWEWSCF20080004_rev.2
1.43e−02	105−Z−1001−LLW_v.1	SWEWSCF19980078_rev.2
1.28e−02	P007OUTFALCERCLA–LLW_v.0	SWEWSCF20100023_rev.0
1.10e−02	5PDPCBBULKCERCLA–LLW_v.1	SWEWSCF20100038_rev.3
1.08e−02	105RPROCESS–LLW_v.2	SWEWSCF20070026_rev.2
1.08e−02	211FCERCLA–LLW_v.0	SWEWSCF20070014_rev.0
9.36e−03	00132−LLW_v.0	SWE–WSCF−1995−0005
9.03e−03	00475−LLW_v.0	SWE–WSCF−1996−0011
8.76e−03	221F0006–LLW_v.0	SWEWSCF19950097_rev.2
8.12e−03	105L3004–LLW_v.0	SWEWSCF20070027_rev.1
7.55e−03	105RPROPCBBULK–LLW_v.1	SWEWSCF20090014_rev.1

filtered by high 1 OOM

TC−99 − 2.45%

1.36e−03	MCU00003–LLW_v.0	SWEWSCF20100052_rev.0
1.30e−03	105L2004–LLW_v.1	SWEWSCF20040067_rev.1
8.98e−04	221FCOMPOSITEJCW–LLW_v.0	SWEWSCF20060030_rev.0
4.84e−04	211FPIPRMCERCLA–LLW_v.0	SWEWSCF20080028_rev.0
4.53e−04	221FALINE–LLW_v.2	SWEWSCF20020039_rev.2
3.94e−04	221FCOMPOSITEJCW–LLW_v.1	SWEWSCF20060030_rev.1
3.54e−04	221FALINE–LLW_v.3	SWEWSCF20020039_rev.3

```
2.76e-04    ZSS007001-LLW_v.0          SWEWSCF20070012_rev.0
1.85e-04    211FCERCLA-LLW_v.0         SWEWSCF20070014_rev.0
1.84e-04    ASSAYB25-LLW_v.0           SWEWSCF20030032_rev.0
1.82e-04    221F0006-LLW_v.0           SWEWSCF19950097_rev.2
1.66e-04    BAL001A-LLW_v.1            SWEWSCF20050007_rev.1
1.55e-04    221H0034-LLW_v.1           SWEWSCF20030058_rev.1
1.50e-04    FHW00001HTFJCW-LLW_v.2     SWEWSCF20010039_rev.2
1.39e-04    ETF-00003-1-LLW_v.3        SWEWSCF19940014_rev.5
1.31e-04    772F022B-LLW_v.5           SWEWSCF19980062_rev.7
filtered by high 1 OOM
```

==========

SLIT8

```
I-129 - 100.00%
4.23e-05    211FPIPEOSF-LLW_v.0        SWEWSCF20060064_rev.1
3.95e-05    211FTANKOSF-LLW_v.1        SWEWSCF20060059_rev.1
5.24e-06    HBLBLACKBOX-LLW_v.0        SWEWSCF20030073_rev.0
9.97e-07    211F813OSF-LLW_v.1         SWEWSCF20060061_rev.1
filtered by high 1 OOM
```

```
TC-99 - 6.56%
3.12e-03    SDDSTCROOM-LLW_v.0         SWEWSCF20060043_rev.0
4.65e-04    221H0040-LLW_v.0           SWEWSCF20070018_rev.0
4.39e-04    FHW00001HTFJCW-LLW_v.3     SWEWSCF20010039_rev.3
1.58e-04    221FCOMPOSITEJCW-LLW_v.0   SWEWSCF20060030_rev.0
filtered by high 1 OOM
```

```
SR-90 - 2.85%
1.41e+00    FHW00001HTFJCW-LLW_v.3     SWEWSCF20010039_rev.3
1.11e+00    221H0040-LLW_v.0           SWEWSCF20070018_rev.0
6.12e-01    221S00017-LLW_v.0          SWEWSCF20030076_rev.0
1.61e-01    221FCOMPOSITEJCW-LLW_v.0   SWEWSCF20060030_rev.0
1.15e-01    221H0034-LLW_v.1           SWEWSCF20030058_rev.1
filtered by high 1 OOM
```

```
NI-59 - 1.48%
3.76e-02    221H0034-LLW_v.1           SWEWSCF20030058_rev.1
2.40e-03    5RDPCBBULKCERCLA-LLW_v.0   SWEWSCF20100034_rev.1
filtered by prev 1 OOM
```

==========

SLIT9

```
H-3-X - 100.00%
1.85e+02    745NHXPHS2-LLW_v.0         SWEWSCF20120013_rev.0
unfiltered
```

```
C-14-X - 26.60%
8.62e-01    745NHXPHS2-LLW_v.0         SWEWSCF20120013_rev.0
unfiltered
```

H–3 – 9.43%

3.51e−02	FHW00001FTFJCW–LLW_v.0	SWEWSCF20010039_rev.0
2.89e−02	ETF−001−125−LLW_v.6	SWEWSCF19990026_rev.8
2.87e−02	FHW00001HTFJCW–LLW_v.0	None
1.85e−02	ETF−001−125−LLW_v.5	SWEWSCF19990026_rev.7
1.30e−02	745NHXDECON–LLW_v.0	SWEWSCF20110043_rev.1
1.11e−02	00475−LLW_v.0	SWE–WSCF−1996−0011
9.71e−03	221H0021–LLW_v.5	SWEWSCF19970052_rev.6
8.50e−03	00524−LLW_v.4	SWEWSCF19960043_rev.5
7.66e−03	00524−LLW_v.3	SWEWSCF19960043_rev.4
7.00e−03	00001−LLW_v.0	SWEWSCF19950007_rev.0
4.59e−03	105C803SLUDGE–LLW_v.0	SWEWSCF20120017_rev.0
3.67e−03	BAL001A–LLW_v.3	SWEWSCF20050007_rev.3
2.89e−03	NAVY1–LLW_v.0	None

filtered by high 1 OOM

Sum of all waste streams, sorted by total activity

1.86e+02	745NHXPHS2–LLW_v.0	SWEWSCF20120013_rev.0
5.12e+01	NAVY1–LLW_v.0	None
7.93e+00	SEGSLUDGE3–LLW_v.1	SWEWSCF19990032_rev.1
3.87e+00	00564−LLW_v.0	SWE–WSCF−1996−0060
1.57e+00	105L9003–LLW_v.0	SWEWSCF20040051_rev.0
1.41e+00	FHW00001HTFJCW–LLW_v.3	SWEWSCF20010039_rev.3
1.11e+00	221H0040–LLW_v.0	SWEWSCF20070018_rev.0
8.94e−01	00473−LLW_v.0	None
6.12e−01	221S00017–LLW_v.0	SWEWSCF20030076_rev.0
5.66e−01	00475−LLW_v.0	SWE–WSCF−1996−0011
5.36e−01	00568−LLW_v.0	None
3.04e−01	NAVY2–LLW_v.0	None
2.75e−01	105X3001–LLW_v.0	SWEWSCF19990086_rev.0
2.52e−01	00460−LLW_v.0	SWE–WSCF−1995−0141
1.63e−01	221FCOMPOSITEJCW–LLW_v.0	SWEWSCF20060030_rev.0
1.59e−01	FDDTRITIUM–LLW_v.0	SWEWSCF20020030_rev.0
1.52e−01	221H0034–LLW_v.1	SWEWSCF20030058_rev.1
1.48e−01	00618−LLW_v.0	SWE–WSCF−1996−0023
1.42e−01	105−Z−1001−LLW_v.1	SWEWSCF19980078_rev.2
1.25e−01	00100−LLW_v.0	None
1.06e−01	00617−LLW_v.2	SWEWSCF19960020_rev.4
9.33e−02	00617−LLW_v.0	SWE–WSCF−1996−0020_rev.2
8.61e−02	FAGW−0001−LLW_v.3	SWEWSCF19970030_rev.5
7.64e−02	4202DTRITCERCLA–LLW_v.0	SWEWSCF20110001_rev.0
7.02e−02	FAGW−0002−LLW_v.1	SWEWSCF19990030_rev.1
5.31e−02	00503−LLW_v.0	None
5.28e−02	00618−LLW_v.2	SWEWSCF19960023_rev.3
4.55e−02	DPCONCRETE02–LLW_v.0	SWEWSCF20020079_rev.0
4.46e−02	ETF−001−125−LLW_v.0	SWEWSCF19990026_rev.1
4.11e−02	105RDISAREA–LLW_v.1	SWEWSCF20090015_rev.1
3.88e−02	00486−LLW_v.0	None

3.61e−02	FAGW−0001−LLW_v.0	SWEWSCF19970030_rev.0
3.51e−02	FHW00001FTFJCW−LLW_v.0	SWEWSCF20010039_rev.0
3.30e−02	FAGW−0004−LLW_v.0	SWEWSCF19980073_rev.1
3.29e−02	00613−LLW_v.0	SWE−WSCF−1996−0068
3.10e−02	FHW00001HTFJCW−LLW_v.0	None
2.89e−02	ETF−001−125−LLW_v.6	SWEWSCF19990026_rev.8
2.85e−02	211FPIPRMCERCLA−LLW_v.0	SWEWSCF20080028_rev.0
2.52e−02	105X3501−LLW_v.0	SWEWSCF20000021_rev.0
2.23e−02	TRITSORT01−LLW_v.0	SWEWSCF19980006_rev.0
2.04e−02	105PDISAREA−LLW_v.0	SWEWSCF20080004_rev.0
1.97e−02	ETF−00003−1−LLW_v.2	SWEWSCF19940014_rev.4
1.85e−02	ETF−001−125−LLW_v.5	SWEWSCF19990026_rev.7
1.62e−02	105PDISAREA−LLW_v.2	SWEWSCF20080004_rev.2
1.48e−02	77710A0001−LLW_v.0	SWEWSCF20050001_rev.1
1.30e−02	745NHXDECON−LLW_v.0	SWEWSCF20110043_rev.1
1.28e−02	P007OUTFALCERCLA−LLW_v.0	SWEWSCF20100023_rev.0
1.27e−02	5PDPCBBULKCERCLA−LLW_v.1	SWEWSCF20100038_rev.3
1.21e−02	244H2004−LLW_v.1	SWEWSCF20000051_rev.2
1.17e−02	00483−LLW_v.0	None
1.14e−02	FAGW−0005−LLW_v.1	SWEWSCF19990005_rev.2
1.10e−02	211FCERCLA−LLW_v.0	SWEWSCF20070014_rev.0
1.08e−02	105RPROCESS−LLW_v.2	SWEWSCF20070026_rev.2
1.01e−02	221H0021−LLW_v.5	SWEWSCF19970052_rev.6
9.36e−03	00132−LLW_v.0	SWE−WSCF−1995−0005
9.15e−03	FHLABSTC−LLW_v.2	SWEWSCF20080003_rev.2
8.94e−03	221F0006−LLW_v.0	SWEWSCF19950097_rev.2
8.61e−03	230HBGI−LLW_v.0	SWEWSCF20060039_rev.0
8.50e−03	00524−LLW_v.4	SWEWSCF19960043_rev.5
8.35e−03	221H0040−LLW_v.1	SWEWSCF20070018_rev.1
8.12e−03	105L3004−LLW_v.0	SWEWSCF20070027_rev.1
7.66e−03	00524−LLW_v.3	SWEWSCF19960043_rev.4
7.55e−03	105RPROPCBBULK−LLW_v.1	SWEWSCF20090014_rev.1
7.40e−03	221H0036−LLW_v.0	SWEWSCF20030060_rev.0
7.00e−03	00001−LLW_v.0	SWEWSCF19950007_rev.0
6.96e−03	221H0034−LLW_v.0	SWEWSCF20030058_rev.0
6.25e−03	00689−LLW_v.2	SWEWSCF19970047_rev.2
6.13e−03	HIGHLEVELCEL06−LLW_v.1	SWEWSCF20060020_rev.2
5.82e−03	ZSS0043−LLW_v.0	SWEWSCF20040031_rev.0
5.63e−03	772F037O−LLW_v.0	SWEWSCF20020023_rev.0
4.59e−03	105C803SLUDGE−LLW_v.0	SWEWSCF20120017_rev.0
4.20e−03	ZSS001125−LLW_v.1	SWEWSCF20020018_rev.1
3.88e−03	FHLABSTC−LLW_v.1	SWEWSCF20080003_rev.1
3.68e−03	00523−LLW_v.3	SWEWSCF19960027_rev.4
3.67e−03	BAL001A−LLW_v.3	SWEWSCF20050007_rev.3
3.63e−03	HMWST00001−LLW_v.0	SWEWSCF20000006_rev.1
3.55e−03	109RBASIN−LLW_v.0	SWEWSCF20060013_rev.0
3.51e−03	105L2004−LLW_v.0	SWEWSCF20040067_rev.0
3.32e−03	SRNLJCW−LLW_v.1	SWEWSCF20060036_rev.1
3.19e−03	770URXCERCLA−LLW_v.0	SWEWSCF20100047_rev.0
3.12e−03	SDDSTCROOM−LLW_v.0	SWEWSCF20060043_rev.0
2.97e−03	MCU00004−LLW_v.0	SWEWSCF20110045_rev.0

2.74e−03	105L2004–LLW_v.1	SWEWSCF20040067_rev.1
2.64e−03	00427−LLW_v.1	SWEWSCF19950075_rev.2
2.40e−03	5RDPCBBULKCERCLA–LLW_v.0	SWEWSCF20100034_rev.1
2.40e−03	FHLABSTC–LLW_v.0	SWEWSCF20080003_rev.0
2.37e−03	FHLABSTC–LLW_v.3	SWEWSCF20080003_rev.3
2.26e−03	SRNLJCW–LLW_v.4	SWEWSCF20060036_rev.4
2.24e−03	728NDEBRIS–LLW_v.0	SWEWSCF20190004_rev.0
2.23e−03	221H0021–LLW_v.7	SWEWSCF19970052_rev.8
2.23e−03	HTK00002–LLW_v.3	SWEWSCF20030014_rev.3
2.16e−03	221FOXIDE–LLW_v.0	SWEWSCF20040032_rev.0
2.12e−03	105CDISAREA–LLW_v.0	SWEWSCF20110028_rev.1
2.11e−03	MCU00003–LLW_v.0	SWEWSCF20100052_rev.0
2.02e−03	MCU00004–LLW_v.1	SWEWSCF20110045_rev.1
1.87e−03	HTK00002–LLW_v.6	SWEWSCF20030014_rev.6
1.83e−03	FHW00001HTFJCW–LLW_v.7	SWEWSCF20120030_rev.3
1.73e−03	PAD02031–LLW_v.0	SWEWSCF20030048_rev.1
1.58e−03	772F023C–LLW_v.0	SWEWSCF19980063_rev.0
1.54e−03	772FHLGEN–LLW_v.2	SWEWSCF20140001_rev.2
1.48e−03	ASSAYBYPACKAGE–LLW_v.4	SWEWSCF20100009_rev.3
1.47e−03	FHW00003HTF–LLW_v.0	SWEWSCF20030035_rev.0
1.42e−03	00281−LLW_v.0	SWE–WSCF−1995−0036
1.42e−03	221H0021–LLW_v.6	SWEWSCF19970052_rev.7
1.40e−03	FHW00001HTFJCW–LLW_v.1	SWEWSCF20010039_rev.1
1.35e−03	772FHLGEN–LLW_v.1	SWEWSCF20140001_rev.1
1.28e−03	773−A VAR. CASK–LLW_v.7	SWEWSCF19980004_rev.7
1.22e−03	772F022B–LLW_v.4	SWEWSCF19980062_rev.6
1.14e−03	247F10032004–LLW_v.0	SWEWSCF20040015_rev.0
1.13e−03	221S00024–LLW_v.1	SWEWSCF20120005_rev.1
1.10e−03	512S01701–LLW_v.1	SWEWSCF20170008_rev.1
1.02e−03	HTK00002–LLW_v.5	SWEWSCF20030014_rev.5
9.96e−04	772F039Q–LLW_v.0	SWEWSCF20040004_rev.0
9.10e−04	221S00022−LLW_v.0	SWEWSCF20100050_rev.0
9.01e−04	221H0021–LLW_v.4	SWEWSCF19970052_rev.4
8.92e−04	100ROOM–LLW_v.0	SWEWSCF20030069_rev.0
8.60e−04	SRNLJCW–LLW_v.0	SWEWSCF20060036_rev.0
8.51e−04	HTK00002–LLW_v.7	SWEWSCF20030014_rev.7
8.24e−04	773AC–PROCESS–LLW_v.2	SWEWSCF19990022_rev.2
7.65e−04	105L2004–LLW_v.2	SWEWSCF20040067_rev.2
7.62e−04	FHW00001FTFJCW–LLW_v.7	SWEWSCF20120036_rev.3
6.95e−04	221H0021–LLW_v.0	SWEWSCF19970052_rev.0
6.84e−04	ETF−002−12−LLW_v.0	SWEWSCF19990035_rev.1
6.01e−04	HTK00002–LLW_v.4	SWEWSCF20030014_rev.4
5.66e−04	105X6025–LLW_v.0	SWEWSCF20030052_rev.0
5.57e−04	221H0021–LLW_v.1	SWEWSCF19970052_rev.1
5.25e−04	HBLTRUTOLLW01–LLW_v.0	SWEWSCF20070030_rev.1
4.84e−04	MCU00004–LLW_v.2	SWEWSCF20110045_rev.2
4.60e−04	ETF−001−125−LLW_v.7	SWEWSCF19990026_rev.9
4.53e−04	221FALINE–LLW_v.2	SWEWSCF20020039_rev.2
3.94e−04	221FCOMPOSITEJCW–LLW_v.1	SWEWSCF20060030_rev.1
3.69e−04	ASSAYB25–LLW_v.0	SWEWSCF20030032_rev.0
3.57e−04	00364−LLW_v.1	SWEWSCF19950045_rev.3

3.54e−04	221FALINE–LLW_v.3	SWEWSCF20020039_rev.3
3.15e−04	LTR60006CERCLA–LLW_v.0	SWEWSCF20120010_rev.0
2.87e−04	FHW00001HTFJCW–LLW_v.5	SWEWSCF20120030_rev.1
2.76e−04	ZSS007001−LLW_v.0	SWEWSCF20070012_rev.0
2.71e−04	211F813SLUDGE–LLW_v.0	SWEWSCF20070001_rev.1
2.68e−04	221H0036–LLW_v.5	SWEWSCF20030060_rev.5
2.58e−04	772FHLGEN–LLW_v.0	SWEWSCF20140001_rev.0
2.00e−04	221H0021–LLW_v.8	SWEWSCF19970052_rev.9
1.66e−04	BAL001A–LLW_v.1	SWEWSCF20050007_rev.1
1.50e−04	FHW00001HTFJCW–LLW_v.2	SWEWSCF20010039_rev.2
1.42e−04	ETF0032–LLW_v.1	SWEWSCF20090003_rev.2
1.39e−04	ETF−00003−1−LLW_v.3	SWEWSCF19940014_rev.5
1.31e−04	772F022B–LLW_v.5	SWEWSCF19980062_rev.7
1.18e−04	HAGW−0004A–LLW_v.0	SWEWSCF19980051_rev.1
9.60e−05	H3DECAY–LLW_v.0	SWEWSCF20070006_rev.0
8.78e−05	FAGW−0005−LLW_v.0	SWEWSCF19990005_rev.0
5.25e−05	FAREANONROUTINE–LLW_v.0	SWEWSCF20070035_rev.0
5.06e−05	105X2005–LLW_v.0	SWEWSCF19960033_rev.2
4.33e−05	211FTANKOSF–LLW_v.1	SWEWSCF20060059_rev.1
4.23e−05	211FPIPEOSF–LLW_v.0	SWEWSCF20060064_rev.1
3.28e−05	100TRITIUM–LLW_v.0	SWEWSCF20050045_rev.0
3.07e−05	FCXBOX157–LLW_v.0	SWEWSCF20110040_rev.0
2.79e−05	FAGW−0004−LLW_v.2	SWEWSCF19980073_rev.3
2.36e−05	FAGW−0004−LLW_v.1	SWEWSCF19980073_rev.2
1.83e−05	ASSAYBYPACKAGE–LLW_v.7	SWEWSCF20100009_rev.6
1.33e−05	BAL001A–LLW_v.0	SWEWSCF20050007_rev.0
1.25e−05	105L9005–LLW_v.0	SWEWSCF20070021_rev.0
1.07e−05	211FTANKOSF–LLW_v.0	SWEWSCF20060059_rev.0
8.61e−06	HBLBLACKBOX–LLW_v.0	SWEWSCF20030073_rev.0
7.88e−06	STP055LLW1–LLW_v.0	SWEWSCF19970054_rev.0
6.47e−06	FHG02002–LLW_v.0	SWEWSCF20020025_rev.1
5.99e−06	FCANBLBX06–LLW_v.0	SWEWSCF20060055_rev.0
5.38e−06	00689−LLW_v.1	SWEWSCF19970047_rev.1
4.83e−06	SOL99001CERCLA–LLW_v.0	SWEWSCF20010031_rev.0
4.13e−06	00685−LLW_v.1	SWEWSCF19970039_rev.1
3.44e−06	221F0006HD–LLW_v.0	SWEWSCF19970048_rev.1
3.18e−06	10502105−LLW_v.0	SWEWSCF20020037_rev.0
2.63e−06	221H0042–LLW_v.1	SWEWSCF20070019_rev.1
2.46e−06	221S−00011−LLW_v.1	SWEWSCF19980049_rev.2
1.94e−06	105X3512–LLW_v.0	SWEWSCF20020073_rev.0
1.75e−06	ETF−002−12−LLW_v.5	SWEWSCF19990035_rev.7
1.56e−06	BOX332005–LLW_v.0	SWEWSCF20050050_rev.0
1.50e−06	221FB25HOTSIDE–LLW_v.0	SWEWSCF19990134_rev.0
1.05e−06	221F0006SOIL–LLW_v.1	SWEWSCF19980088_rev.1
1.03e−06	105X6029–LLW_v.0	SWEWSCF20030065_rev.1
9.97e−07	211F813OSF–LLW_v.1	SWEWSCF20060061_rev.1
9.15e−07	105X3513–LLW_v.0	SWEWSCF20020077_rev.0
7.83e−07	221F0002–LLW_v.6	SWEWSCF19950094_rev.9
7.19e−07	CIF99017–LLW_v.0	SWEWSCF19990038_rev.0
6.59e−07	105X6026–LLW_v.0	SWEWSCF20030064_rev.1
5.96e−07	105X6030–LLW_v.0	SWEWSCF20040029_rev.0

```
5.08e-07    105L6034-LLW_v.0          SWEWSCF20050048_rev.0
4.81e-07    00686-LLW_v.1             SWEWSCF19970038_rev.1
4.72e-07    BGC00002-LLW_v.0          SWEWSCF20010005_rev.0
4.35e-07    FBI02001-LLW_v.0          SWEWSCF20020026_rev.0
4.25e-07    244H4005-LLW_v.0          SWEWSCF20010099_rev.0
3.79e-07    221HBL-LLW_v.5            SWEWSCF20090032_rev.5
3.60e-07    244H4006-LLW_v.0          SWEWSCF20020056_rev.0
3.29e-07    00666-LLW_v.4             SWEWSCF19970022_rev.4
3.12e-07    221S00016-LLW_v.1         SWEWSCF20010095_rev.1
2.39e-07    772F022B-LLW_v.6          SWEWSCF19980062_rev.9
unfiltered
```

```
This took:  5.592644453048706 secs
             0.0932107408841451 mins
```

Distribution:

sebastian.aleman@srnl.doe.gov
thomas.danielson@srnl.doe.gov
james.dyer@srnl.doe.gov
clint.gregory@srnl.doe.gov
maximilian.gorensek@srnl.doe.gov
luther.hamm@srnl.doe.gov
thong.hang@srnl.doe.gov
brady.lee@srnl.doe.gov
patricia.lee@srnl.doe.gov
dien.li@srnl.doe.gov
john.mayer@srnl.doe.gov
richard.minichan@srnl.doe.gov
stephanie.taylor@srnl.doe.gov
tad.whiteside@srnl.doe.gov
jennifer.wohlwend@srnl.doe.gov

heather.capogreco@srnl.doe.gov

Records Administration (EDWS)