In presenting this thesis in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the Library shall make it freely available for inspection. I further state that permission for extensive copying of my thesis for scholarly purposes may be granted by the Dean of the Graduate School, Dean of my academic division, or by the University Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my permission.

Signature _____

Date _____

Developments for the Stochastic Objective Decision Aide and Investigation into Respirable Fraction Parameter Distribution

by

Andrew Maas

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in the Nuclear Engineering Program

Idaho State University

Spring 2017

To the Graduate Faculty:

The members of the committee appointed to examine the thesis of ANDREW MAAS find it satisfactory and recommend that it be accepted.

_____

Dr. Chad Pope
Major Advisor

_____

Dr. Jay Kunze
Committee Member

_____

Dr. Seyed Mousavinezhad,
Graduate Faculty Representative

Dedication

This work is dedicated to my parents and my wife, whose support and encouragement over the years of my life has been invaluable.

Acknowledgements

The work performed in a master's thesis is performed by the author in fulfillment of the requirements of a master's degree. That is not to say, however, that this is done without the cooperation of others. I wish to acknowledge those who were helpful and/or instrumental in the success of this project.

This project would not have been possible without the financial support of Alan Levin, and the Nuclear Research Safety and Development (NSRD) office of the Department of Energy (DOE). I am grateful for the opportunity that was presented by the funding of this research. I am also grateful to Dr. Chad Pope, who selected me to work on this project on the ISU side, and who has been a constant source of support and feedback as work developed on the Stochastic Objective Decision Aide (SODA) application and the authorship of this thesis. His mentorship and willingness to share his expertise has been of great value to myself, and this thesis.

I am also grateful to my colleagues who worked with me on the SODA project. I wish to acknowledge Kushal Bhattari, the original code author of the SODA program, for his work both in the preceding year on the SODA project, and his continued work this past year. Kushal was very helpful and supportive in my time getting to know the SODA program, and a source of encouragement during the busier season of the project. I wish to also acknowledge Mary Tosten, who designed the User Defined Distribution (UDD) feature of the SODA program, and who led the charge on the parametric study component of the SODA project. I wish to acknowledge Jason Andrus of the INL, who provided some needed insight into the end use of SODA, some ideas for the future of SODA, and helpful feedback on SODA during development.

I wish to acknowledge my wife, Caitlin Maas, who has been a constant source of loving support throughout this entire process. Without her support, this success of this master's work would not have been possible. Her willingness to help, to listen, and to love are invaluable.

Finally, I wish to acknowledge my step-father, Michael Brashler. Mike is a software professional with many decades of experience, who has been willing to allow me to pick his brain. The insights and strategies which I gained from these interactions were valuable as developments were made in SODA.

About the Author

Andrew Maas was born in Portland, Oregon, and remained in this area throughout his younger years. After completion of High School, Andrew enrolled in community college, and set out on the long task of deciding what direction to take going into the future. After several starts and stops, Andrew decided to pursue a Bachelor of Science in Physics at Portland State University (PSU). While at PSU, Andrew worked for Klein Optical Instruments Inc, where he gained some experience in the working world, and recieved exposure to programming in the workplace. With the support of some classes at the university, Andrew became comfortable solving problems on the computer through programming. After reaching his Junior year in his study of Physics, Andrew decided to leave Klein Instruments behind, and seek out an undergraduate research project. Working with Dr. Peter Moeck, Andrew started working on a predictive method in Bicrystallography. This method involved the development and use of a MATLAB program, providing valuable experience which contributed to the success of the SODA project.

During the senior year of his work in PSU's physics program, Andrew became increasingly interested in the study of Nuclear Power. He felt strongly that the Integral Fast Reactor project of the 1980's offered the best hope for the energy future of the world, so Andrew got into contact with members of the Idaho State University Faculty in order to determine if this department was a good fit for graduate school. Ultimately, Andrew finished his BS in Physics, and started his master's work at ISU, where he met Dr. Pope and joined the SODA team.

Table of Contents

Table of Figures

Table of Acronyms

AED	Aerodynamic Equivalent Diameter
AMAD	Activity Median Aerodynamic Diameter
ANS	American Nuclear Society
ARF	Airborne Release Fraction
BR	Breathing Rate
CED	Committed Effective Dose
CSV	Comma Separated Variable
DBE	Design Basis Event
DCF	Dose Conversion Factor
DR	Damage Ratio
GUI	Graphical User Interface
LPF	Leak Path Factor
MAR	Material at Risk
NNDC	National Nuclear Data Center
OOP	Object Oriented Programming
PDF	Probability Density Function
RF	Respirable Fraction
SODA	Stochastic Objective Decision Aide
SSC	Structure System and Component
ST	Source Term
UDD	User Defined Distribution
USDOE	United States Department of Energy

Abstract

The work performed within this master's thesis is in support of the Stochastic Objective Decision Aide (SODA) project. Developments were made on the SODA program which include support for multi-material calculations, a Dose Conversion Factor (DCF) database, and implementation of program-wide parameter control and error trapping. Programming styles, such as Object Oriented Programming (OOP), were used to simplify the program and maximize performance. As the SODA program uses distribution multiplication to obtain a dose consequence distribution, it is important to develop physically correct distributions for the various input parameters. In support of this goal, a new Monte Carlo based methodology was developed to obtain the parameter distribution for Respirable Fraction (RF) using aerosol data. Formulation of this new methodology is developed within this thesis, and results and improvements are discussed. To demonstrate the use of this new methodology, the RF distribution for Pu metal droplets oxidized in air is developed using tabulated data for the particle size distribution.

1. Introduction

1.1. Background

US DOE non-reactor nuclear facilities use unmitigated hazard analysis to determine whether dose consequence to the public challenges regulatory guidelines from Design Basis Events (DBEs). If an unmitigated DBE challenges these guidelines, safety Structures Systems and Components (SSCs) will be selected to mitigate the consequence of the event, or reduce the frequency of the event. Traditional radioactive release models provide a single point estimate for dose consequence at a receptor site. To bound this figure, conservative values and assumptions are made to consider the worst-case scenario dose consequence. While this method does provide an upper limit to consequence, it fails to produce an expectation value, and does not quantify the uncertainty in the dose consequence. Failure to consider the uncertainty and expectation values can lead decision makers to unnecessarily select SSCs, or to fail to select SSCs when they may be warranted. Unnecessary selection of SSCs can be cost or mission prohibitive, while failing to select SSCs that are needed can result in a potentially excessive dose to the public.

To address the limitations of the single point approach, a computer code was developed that uses Monte Carlo methods to perform dose consequence calculations based upon distribution inputs. This code, the Stochastic Objective Decision Aide (SODA), provides a dose estimate distribution from a modeled release scenario. To support the long-term use of the SODA program, a proposal was made to continue development of SODA to include new features, general improvements, and to promote further understanding of the input parameter distributions through parametric study. It is not the objective of the SODA project to replace existing radioactive release codes, but rather to supplement these codes

with additional information. Supply of additional information about the dose consequence distribution will aid decision makers in their selections of SSCs.

1.2. Committed Effective Dose Calculations

The calculation methodology in SODA uses the traditional equations for source term and dose consequence, with the notable difference of performing these calculations with distribution inputs. The ability to use single point estimates is still retained. The committed effective dose is a function of plume dispersion, the breathing rate of the receptor, the source term, and the dose conversion factor. The equation is shown below [1]:

$$CED = \frac{\chi}{Q} * BR * ST * DCF$$

Where CED is the committed effective dose, chi over Q is the plume dispersion parameter, BR is breathing rate, ST is source term, and DCF is the dose conversion factor. Chi(χ) over Q is a measure of how rapidly an airborne plume will disperse from a given set of atmospheric parameters. Chi over Q has units of seconds per meter cubed (s/m^3). The plume dispersion rate is based upon a Gaussian plume model in SODA. The breathing rate is how much volume of air per unit time is being exchanged through the lungs by a person at the receptor site. Breathing rate has units of meters cubed per second (m^3/s). The source term, which will be described in more detail shortly, has units of Becquerels (Bq), also known as disintegrations per second. Finally, the dose conversion factor relates the activity absorbed by a person at the receptor site to the lifetime dose consequence. This has units of Sieverts per Becquerel (Sv/Bq). DCF values are estimated based on biological models which consider the transport of radionuclides in the body, and their associated biological removal mechanisms. Consideration of these inputs and units implies CED has units of Sieverts, and is a lifetime dose consequence estimate. Sieverts are straightforwardly

converted to rem, which is a more common unit in the US, by the following conversion factor:

$$1\ Sv = 100\ rem$$

The source term (ST) is determined by the "five-factor" formula:

$$ST = MAR * DR * ARF * RF * LPF$$

Where MAR is the material at risk, DR is the damage ratio, ARF is the airborne release fraction, RF is the respirable fraction, and LPF is the leak path factor. The MAR, measured in Bq, is the amount of radioactive material which could potentially be involved in an accident scenario. MAR is the only quantity with units in the five-factor formula, so the ST is in Bq, as expected. The remaining four factors are ratios between zero and one, which modify the MAR quantity to the quantity of interest in airborne release modeling. DR is the ratio of the material which is involved in a modeled accident scenario, to the total available MAR. A DR of one implies that 100% of the MAR was affected by the accident scenario, while a DF of 0.5 implies that only 50% is involved. ARF is the ratio of the material which was involved in the modelled accident scenario, to the amount which becomes airborne. The effect of these ratios compounding on the MAR is demonstrated in Figure 1. RF is the ratio of the amount of material which becomes airborne to the amount which readily deposits in the human lung, where it is in a position to impart its maximal lifetime dose. Finally, LPF is the amount of this respirable material which manages to escape containment, to be transported to the receptor site. Each factor, other than MAR, can be thought of as modifying MAR so as to correctly represent the amount of radioactive material which is available to potentially reach lung deposition at the receptor site. This amount is then modified in the CED equation to provide a dose estimate.

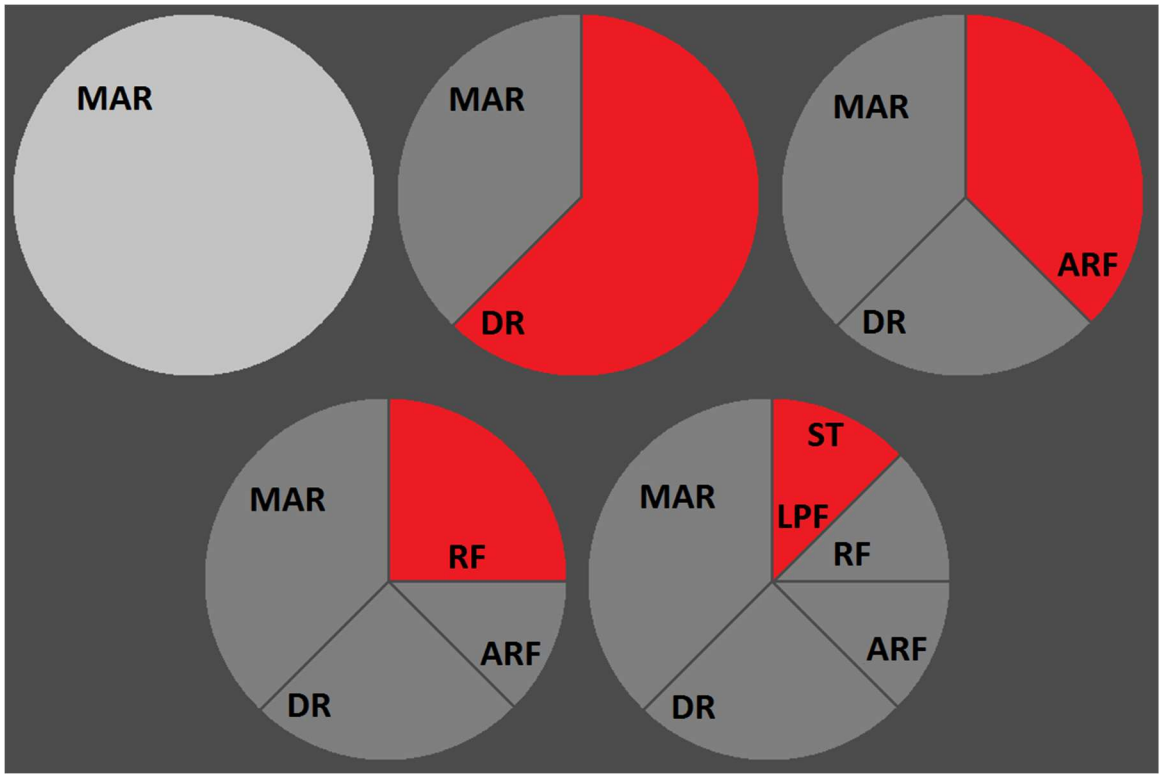


Figure 1. Visualization of the five-factor formula acting on the MAR to get to the ST.

An example calculation is demonstrated:

$$ST = MAR * DR * ARF * RF * LPF$$

$$ST = 3.7E10\ Bq * 0.3 * 0.1 * 0.5 * 0.25 = 1.3875E8\ Bq$$

Substituting into the CED Equation:

$$CED = \frac{\chi}{Q} * BR * ST * DCF$$

$$CED \quad 1E-4\ \frac{s}{m^3} * 1E-3\ \frac{m^3}{s} * 1.3875E8\ Bq * 1E-5\ \frac{Sv}{Bq} = 1.3875E-4\ Sv$$

So, in this example calculation, the lifetime dose consequence at the receptor site is ~0.139 mSv lifetime dose.

The SODA program performs these multiplications for both the ST and the CED equations to obtain a final result. In SODA, however, the option exists to express these inputs as distributions. Distribution multiplication is performed with Monte Carlo methods to obtain the CED distribution result.

1.3. Aerosol Terminology

When discussing airborne radioactive material releases, consider the respirable fraction of the plume, which is treated as an aerosol in most cases. Aerosols are made up of particles of varying sizes and shapes. The way these particles behave, however, is related not to their true size, but to their aerodynamic equivalent size, typically expressed as an aerodynamic equivalent diameter (AED) in reference to an equivalent spherical object. AEDs are easier to measure than actual sizes as well, due to the relationship between AED and particle behavior. In general, particles with large AEDs will have weak aerodynamic entrainment, and will tend to behave according to their inertia. In contrast, particles with very small AEDs will tend to be strongly aerodynamically entrained, and will behave as a gas. One way to characterize an aerosol is the median AED, often expressed as an activity median aerodynamic diameter (AMAD). This characterizes the average values of AED for the aerosol, and gives an idea of how this aerosol will behave. Examples include smoke, which has a very small AMAD, versus a dust cloud, which will have a larger AMAD (in general). A more detailed discussion of aerosols can be found in chapter 9.

1.4. Monte Carlo Methods

Monte Carlo methods involve the use of random numbers to perform stochastic (random) sampling of a problem to obtain a solution. The technique is known, more formally, as statistical sampling. Monte Carlo methods can be used to solve integrals, or

obtain the expectation value for a function. A classic example of how Monte Carlo works, is using statistical sampling to get an estimation of π. The area of a circle is given by π times the square of the radius. The area of a square is simply the square of the side length. Consider then, a circle inscribed within a square, where the radius of the circle is equal to one half the length of the side of the square, as shown in Figure 2.

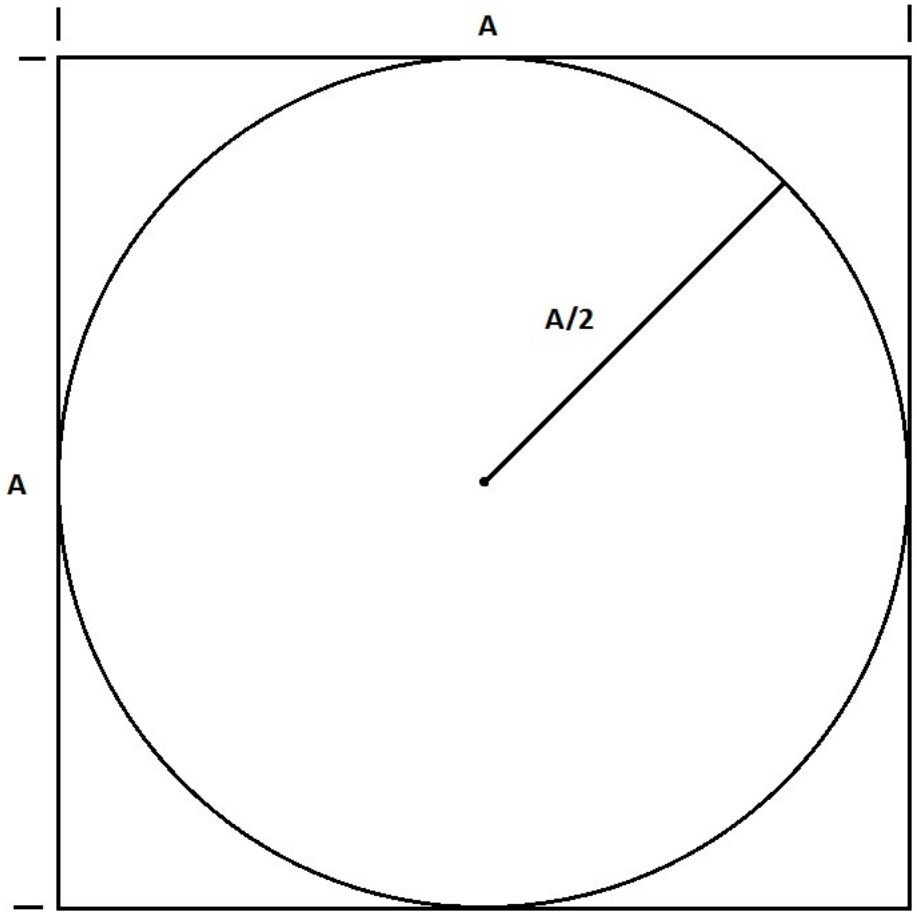


Figure 2. A circle of radius A/2 within a square of side length A, which share a common central point.

Considering these two shapes, it is obvious that the area of the square is A^2, while the area of the circle is $\pi*(A/2)^2$. Consider the ratio of these areas:

$$\frac{Area(Circle)}{Area(Square)} = \frac{\pi * (A/2)^2}{A^2} = \frac{\pi * A^2/4}{A^2} = \frac{\pi}{4}$$

Thus:

$$\pi = 4 * \frac{Area(Circle)}{Area(Square)}$$

So, π is demonstrated to be 4 times the ratio of the area of a circle with radius A/2 to the area of a square with side length A. Now, assume that the lower left corner of the square is the origin on the XY plane. If we chose two random numbers, x and y, which are equiprobable to be anywhere between zero and A, the result will be an ordered pair within the square. While it will not be proved here, it can be demonstrated that if an infinite number of such ordered pairs were to be sampled, then the ratio of the number of points that lie within the circle to the number of total points, will be equal to the ratio of the area of the circle to the area of the square. Since we cannot sample this problem an infinite amount of times, however, the result will be only an approximation to this ratio. So:

$$\pi \cong 4 * \frac{Number\ of\ Points\ within\ Circle}{Total\ Number\ of\ Points}$$

The above equation holds when a statistically significant number of samples are made. The accuracy of a Monte Carlo result relies on the string of random numbers being generated meeting randomness testing critera. In general, if the random number string is sufficiently random, the error in a Monte Carlo result is proportional to the inverse square root of the number of samples [2]. As a result, a very large number of samples is needed if very small errors are required.

Often, Monte Carlo methods are used to solve far more complex problems than the one shown above. The example given was a way to introduce the topic in a simplified

manner. For the circle within a square, the probability density functions (PDF) for both random variables were simple step functions. The probability for values less than zero, or greater than A, is zero, while the probability within the interval is uniform. This represents the simplest of cases. Typically, PDFs will be non-uniform, and encompass the physics of a problem. In SODA, the user defines the PDF for distribution inputs, and these PDFs are used to obtain the distribution result for CED. A random number, based on the PDF, is sampled, and this is multiplied in the equations shown in section 1.2 to get a sample for the CED. This can be extrapolated to many distribution inputs; a random number can be sampled for each PDF (for each input parameter) and these are multiplied to obtain a CED result. After many samples, a distribution for the CED can be inferred. This distribution result is the output of SODA.

The topic of Monte Carlo is far more complex and comprehensive than has been shown here. This section acts simply as an introduction to the topic for those with limited experience in this area. It is recommended that further study be made on the subject for curious readers. Reference [2], which has been cited in this chapter, covers the use of Monte Carlo for particle transport, an even more sophisticated use of Monte Carlo than SODA.

2. Source Code Control

2.1. SODA and Source Code Control

The need for source control was identified in the early days of the second year of the SODA project. When multiple individuals are working on a set of code, it is useful to have the ability to coordinate the efforts of each person, while allowing them to work simultaneously, and preserve working versions of the program as development commences. Source control has the added benefit of providing an easy way to keep a

development log as a piece of software matures, and acts as a backup copy. The Git source control software was selected upon recommendations from industry professionals, and GitLab chosen for hosting. Git is an industry standard, and its use is widespread. GitLab is the ISU IT recommendation for free online hosted private repositories. Instructions for how to set up Git and Gitlab can be found in Appendix E.

2.2. Source Code Control Expressions and Terms

When dealing with source code control, there are many terms which have specific meaning in this context. Not all will be covered here, but some basics will be outlined below:

- Clone: Make a full copy of an existing code repository on the machine calling for the clone.
- Commit: Accept the latest changes to the code in the local repository.
- Commit Message: A short message, sent with a commit, which outlines the changes made in the code version which is being committed.
- Push: Move the latest code changes to the remote repository; this is now the "current version."
- Pull: Download the latest code changes from the repository, without downloading the entire remote repository.
- Merge: A process by which divergent code states can be brought back into line with one another.
- Branch: A repository construct which starts as a duplicate of an existing one, but follows a different development track. A branch is intended to be merged back to the main track at a later date. These are used for simultaneous development of

several new features, or to keep a working version on the “master” branch, while new features are added on the new developments branch.

- Master: A branch name which is typically the main version of a code base (by convention); this is usually kept in working order, while features in progress are performed on other branches.
- Fork: A fork occurs when a program in a repository is taking a different development track. These are similar to a branch, but are notably different in that they are NOT intended to be merged back to the master at a later date.

An example of these terms can be made to explain some simple source code control activities. John performs a Clone on the repository; he wants all of the files on the repository to be on his computer. Jane performs a Commit, after finishing a new feature. She wants her work to be sent back to the repository. In good form, Jane writes a Commit Message detailing the work she performed on the new feature. John wants to take an existing piece of open source software, and make it his own. John performs a Fork on the repository, and takes that software on his desired course. Jane has finished a new set of features for her software project. Jane then Merges her branch back to the Master, so that her new feature set can be used in the main product.

For additional details on the terminology of source code control, and other specifics, consult the Git manual. [3]

2.3. Source Code Control Etiquette

Since one of the primary reasons to select source code control is to facilitate effective teamwork on a software project, it is essential for a user of source code control to understand how to use source code control effectively. It is expected, unless mutually

agreed upon by the entire team, that the master branch of the code should be kept in a working state with every commit. When a new set of features is in development, and the code will be "broken" for a time while working on these features, the best strategy is to check out a new branch, where work can be performed without perturbing the master. In many cases, multiple code branches may be checked out simultaneously, allowing each member of the team to work on their assigned feature set. When preparing to merge these branches back to the master, coordinate with other team members to decide the order of the merge. Typically, the largest set of changes should be merged first, since this merge will happen automatically with no issues. Subsequent branches will no longer have a deterministic path back to the master, so a manual merge will often be required. Make manual merges as easy as possible, as they can be quite time consuming.

Another consideration in etiquette is communication. Whenever multiple persons are working on the same checked out branch, or preparing to merge their branches, it is essential that both (or all) parties know what they are getting into ahead of time. Preparation will save a great deal of time when it comes time to perform a manual merge. In addition, it is very important to send clear, concise, and descriptive commit messages when committing the next set of changes. This will allow others to clearly see what is being worked on, in addition to creating a basis for a changelog in the future.

3. Object Oriented Programming

3.1. What is OOP

Object-Oriented Programming is a programming style in which "objects" are used to compartmentalize operations and data storage in a program. Objects, in the context of this work, are classes which contain both information (properties) and functions (methods). A

class object combines a data structure and a set of methods which act upon that data structure. This provides all of the benefits normally associated with a data structure, but has the additional benefit of simplifying the code of the program which instantiates the class. A simplified class in pseudocode can be seen below:

```
Class ExampleClass( )
Properties
        ExampleProperty
        ExampleArray[ ]
        ...
Methods
        ExampleMethod( )
        ...
        End
End
```

In the case of ExampleClass, there are properties and a method which are a part of the class. Each time an instance of ExampleClass is instantiated in a program, these properties will be available to that class instance, and the methods are available to act on the data stored in the properties. Generally speaking, an object-oriented approach to coding becomes increasingly beneficial as the amount of resulting simplification to the main program increases, or as the main program becomes increasingly complex. For example, making a class of two or three variables that has no intrinsic methods would be a weak use of OOP. On the other hand, a class that uses many variables, many methods, or both, is a strong use of OOP. In this latter scenario, a great degree of simplification can be made in the calling program.

It is considered common to create a new function in a program whenever a task is to be repeated many times to simplify the code. This allows the code author to avoid copying and pasting the same operations in many places within the code. OOP expands on this idea

and takes it to the next level. Rather than looking only for repeated operations, consider a situation in which many data elements need to stick together, or repeated tasks are being performed on that data. It is desirable when these scenarios are present in a code base, to simplify that code by creating an object.

3.2. OOP in MATLAB

MATLAB supports the use of OOP in program development. This is accomplished with the class construct. MATLAB makes using classes quite simple, as it is allowed for a class to inherit methods from other classes. When using MATLAB's graphic user interface development tool, GUIDE, it is common to adjust properties of graphics objects by using get and set methods. These same get and set methods can be inherited by a new class in MATLAB, allowing the programmer to access and change properties in the same way that they would when working with GUIDE objects like a button or text field. In addition to these features, MATLAB has high quality, comprehensive documentation on the MATLAB website. For anyone who is interested in using OOP in MATLAB, it is recommended to study the MATLAB documentation. [4]

3.3. How OOP is Used in SODA

In the SODA program, there are two classes which were developed and used. The first class is the SODA_Parameters class, which contains both properties and methods. This class acts as the global data structure, and all data which is needed in different places and times in SODA is stored here. This class was developed in conjunction with the Expanded MAR Function set, which is described in more detail in the following chapter. The second class is the UDDData class. This class is less complex than SODA_Parameters, and is only instantiated within the SODA_Parameters class. In much the same way as a class is used

to simplify the program that uses it, the UDDData class was used to simplify the SODA_Parameters class when the User Defined Distribution (UDD) feature was integrated into SODA. Using this inner class in SODA_Parameters made it possible to add 4 additional properties, rather than 24, to store UDD data. Unlike in SODA_Parameters, the UDDData class uses its own getter and setter methods. Methods were also added in SODA_Parameters to simplify the process of storing UDD data from SODA. Details of how these classes and methods function within the context of the SODA program are provided in later chapters.

The code for these classes, and their usage in SODA, can be found in Appendix A.

4. Expanded MAR Functionality

4.1. MAR Selection Tool

The first step in expanding MAR functionality in SODA was to develop the MAR Selection Tool. This tool was designed to fulfill the SODA objective of an expanded MAR Database, as well as supporting the later objective of multi-MAR CED calculation. To present this new functionality in an aesthetically pleasing way, the tool was designed to appear like the periodic table. The periodic table used for selecting data is common enough; the National Nuclear Data Center (NNDC) periodic table selector was used as a design reference. This NNDC selector is shown in Figure 3.

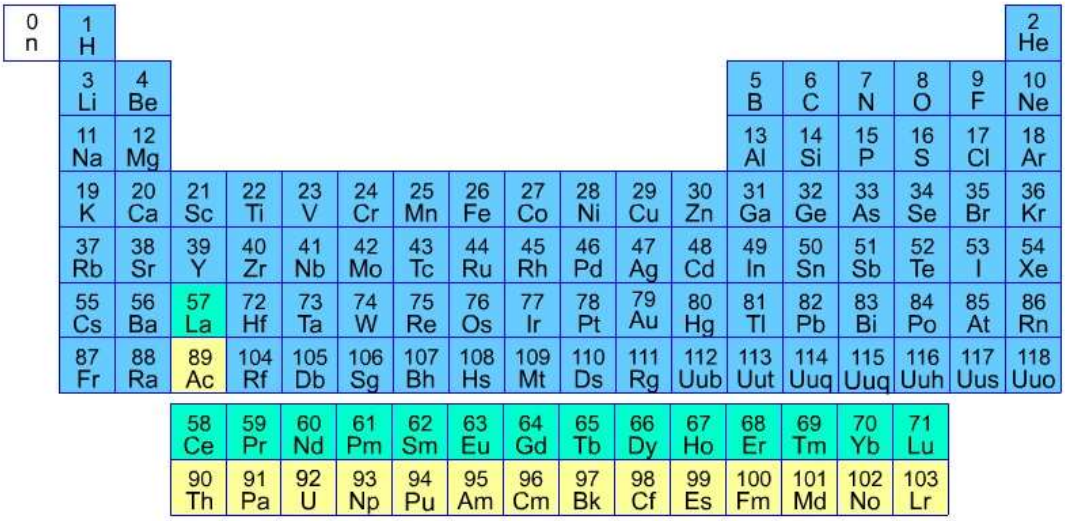


Figure 3. NNDC Periodic Table Selector, which is used as a design reference for the SODA MAR Selection tool. (NNDC)

Using the NNDC design reference for the element selection, other needed features were identified and added. Since DCF values are specific to isotopes, not elements, isotope selection fields were added as well. It would not be feasible to have buttons for each isotope, so selection of an element will cause the isotope selection fields to populate with the available data for this element. To make things easier to identify for the user, elements which have no available data are grayed out. The result of this general layout is shown in Figure 4.

The general usage model for the MAR Selection tool, is to first access the tool from SODA, and then select the radionuclide of interest. To do this, the relevant element is selected, followed by the relevant isotope. Finally, the quantity of the material is entered, and this information is exported to SODA. After adapting the tool to accommodate multiple material selections, radio buttons were added in the upper right corner of the display. After selecting another MAR, the element and/or isotope buttons can be selected, as was the case

in the single isotope method. Clicking the Export to SODA button sends the DCF and MAR quantity to SODA for all selected MARs. Before saving the data to the main class object in SODA, checks are performed to ensure that the data is valid.

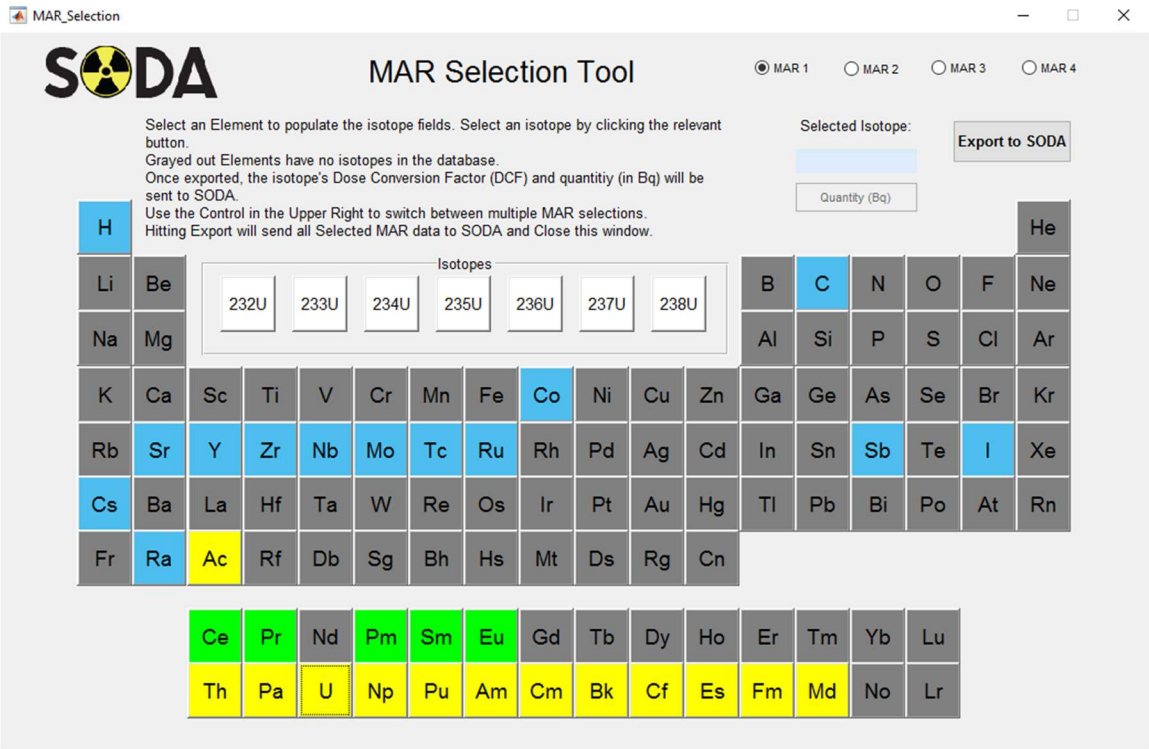


Figure 4. The Final MAR Selection Tool, with available Uranium isotopes loaded.

The final result is an aesthetically pleasing and easy to use tool. The code for this tool was designed to be flexible and easy to follow. More details on the coding strategy can be found in chapter 8. Part of the adaptability of the tool is how it grays out selections. Each time that the tool is opened, the MAR Database is checked for data, and any element which has no data is grayed out. This means that when the MAR Database is changed, this will be reflected in the MAR Selection Tool the next time it is opened.

4.2. MAR Database

Behind the MAR Selection tool is a database file, which holds the DCF data which is exported to the main SODA GUI. The database file itself is a comma separated variable (CSV) file, which allows it to be easily modified and viewed in either a text editor or Microsoft Excel (or an equivalent spreadsheet program). The csv file is fixed format, meaning that the format in the file cannot be changed, else when SODA tries to look at the data, there will be errors. The fixed format supports up to 7 isotopes for each element, inline with the number of isotope buttons which fit on the MAR Selection GUI. The first row of the file is used for identifiers, and each subsequent row corresponds to an element. The first column of an element row is the atomic number of that element. The next 7 columns are for atomic masses of isotopes which have data in the database. The following 7 columns are for DCF data, corresponding to the atomic mass which was entered into the first block of columns. An excerpt from the file, viewed in excel, is shown in Figure 5 to illustrate.

88	223	224	225	226	228			8.70E-06	3.40E-06	7.70E-06	9.50E-06	1.60E-05		
89	225	227						8.50E-06	5.50E-04					
90	227	228	229	230	232	234		1.00E-05	4.00E-05	2.40E-04	1.00E-04	1.10E-04	7.70E-09	
91	230	231	232	233				7.60E-07	1.40E-04	1.00E-08	3.90E-09			
92	232	233	234	235	236	237	238	3.70E-05	9.60E-06	9.40E-06	8.50E-06	8.70E-06	1.90E-09	8.00E-06
93	234	235	236	237	238	239		5.50E-10	6.30E-10	8.00E-06	5.00E-05	3.50E-09	1.00E-09	
94	238	239	240	241	242	243	244	1.10E-04	1.20E-04	1.20E-04	2.30E-06	1.10E-04	8.60E-11	1.10E-04
95	240	241	243					4.30E-10	9.60E-05	9.60E-05				
96	243	244	245	246	247	248	250	6.90E-05	5.70E-05	9.90E-05	9.80E-05	9.00E-05	3.60E-04	2.10E-03
97	245	246	247	249	250			2.10E-09	3.30E-10	6.90E-05	1.60E-07	1.00E-09		
98	248	249	250	251	252	253	254	8.80E-06	7.00E-05	3.40E-05	7.10E-05	2.00E-05	1.30E-06	4.10E-05
99	251	253	254					2.10E-09	2.70E-06	8.60E-06				
100	253	257						4.00E-07	7.10E-06					
101	258							5.90E-06						

Figure 5. Example section of the MAR Database file.

Coupled with the check in the MAR Selection Tool on each run, the easy to edit nature of this file makes it straightforward to modify the MAR database as a user sees fit. New DCF figures, which may be released in the future, can be added without modification

to the source code. Furthermore, the user may select a set of DCFs that corresponds to their regulatory requirements if the International Comission on Radiological Protection (ICRP) document 119 values which are in the database are not suitable. This will help the SODA program to adapt to changes in the future, without having to delve into source code and recompile.

The current set of materials which are included in the MAR Database were selected from a few criteria. A set of commonly used isotopes for the type of hazard analysis that SODA is expected to be used for is preloaded into the database. Some additional actinides and activation products were added that may be potentially useful. While many elements do not have data, and many isotopes are not present, they can be easily added if a customer needs them.

4.3. Multi-MAR CED Computation

After expanding the MAR Database, and providing a tool to make it easy to access said database within the GUI environment of SODA, multi-material calculation capability was added to SODA. The ability to accommodate multi-MAR CED calculations added some complexity to the program, which was counteracted by a simplifying approach. As mentioned in chapter 3, OOP is a good technique to apply when attempting to simplify complex programs. Input parameters needed to be remembered and recalled for each material. A data structure, by itself, could simplify the data handling, but OOP methods were desirable as well. A class was designed to meet these needs. More detailed discussions of OOP can be found in chapters 3 and 8.

To enable selection of different materials, radio buttons were added to both the MAR Selection tool and the main SODA GUI. Clicking these radio buttons causes the program

to change to the selected material, and the information associated with that material is loaded. Since different MARs may have different chemical and physical properties, specification of ARF and RF is made material specific, in addition to the obvious (MAR and DCF). Finally, a new run button was added which performs a CED calculation for all selected materials.

In performing a multi-MAR CED calculation, some assumptions must be made. First off, as SODA is not an atmospheric code, it was outside of the scope of the project to account for deposition processes which may change the relative concentration of a plume while it moves towards the receptor site. Due to this, it is assumed that the relative concentration of each radionuclide species in the plume remains the same from release to receptor. Next, it is assumed that all radionuclide species in the plume will be well mixed by the time they reach the receptor site. Given the distances involved, and the typical atmospheric conditions, this is a reasonable assumption in many cases. When atmospheric and/or weather factors cause this assumption to break down, a code which can correctly account for these phenomena should be employed. The results from such a code can be used to apply a correction factor to the leak path factor to get a good result with SODA.

The way the multi-MAR CED is calculated, given these assumptions, is to calculate the CED distribution for each species, and then sum these resulting distributions. Given this method, a multi-MAR CED result can be somewhat variable. In the case that one MAR is dominant, from either a high DCF, a high quantity, or both, the multi-MAR CED result will be nearly identical to the CED distribution for this dominant species. In other, more interesting cases, the results can be unique, and harder to predict.

5. Log-Normal Distribution Option

The Log-Normal Distribution option was originally added by Kushal Bhattari to the SODA program, to add additional distribution options. [5] After Kushal added this feature to SODA, some changes were made to make the feature more user friendly. The standard values used to specify a log-normal distribution in MATLAB are the normal mean and the normal standard deviation. These are also known as the location parameter (μ) and the scale parameter (σ). An example of various log-normal distributions, and their associated location and scale parameters, can be found in Figure 6. Note that all three have the same location parameter, but differing peak locations.

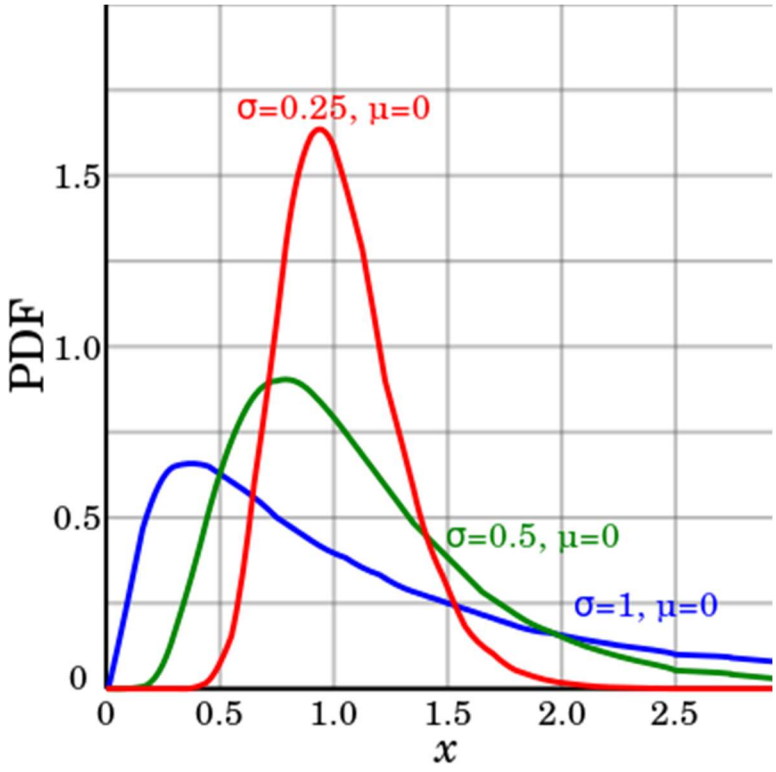


Figure 6. An example of three log-normal distributions. (Wikipedia)

When considering the location of a normal distribution, it is common to consider the location the highest point of the peak, also known as the mode. In the case of a normal

distribution, the mean, median, and mode of the distribution tend to lie at the same location. This is not true in a log-normal distribution. The mode value, the maximum value in the distribution, is not at the same location as the mean or median. This can make the log-normal distribution difficult to visualize, especially when expressed in terms of the location and scale parameter. Consider Figure 6; note that all three distributions shown feature a location parameter of zero. The modes of these distributions, however, are not in the same place. One goal of SODA is to provide a low barrier of entry for decision makers to perform distribution calculations. In support of this goal, it was decided to alter the method of specification for log-normal distributions in order to make them more intuitive.

In SODA, specification of a log-normal distribution is made by inputting a mode value and a scale parameter. The scale parameter, when combined with a mode specification, provides the expected response on the distribution. A larger scale parameter stretches the distribution, while a smaller scale parameter concentrates it, which is how the standard deviation effects a normal distribution. On the other hand, the location parameter does not provide a meaningful relationship between the location of the peak and the value. By specifying the mode, the user can locate the peak of the distribution in a straightforward manner. The user is still able to supply other inputs for the log-normal distribution, by considering the following relationships:

$$Mode = e^{\mu-\sigma^2}$$

$$\ln(Mode) = \mu - \sigma^2$$

$$\ln(Mode) + \sigma^2 = \mu$$

$$Mode = Median * e^{-\sigma^2}$$

$$\frac{Mode}{e^{-\sigma^2}} = Median$$

Where μ is the location parameter, and σ is the scale parameter. Thus, for a known scale parameter, the relationship between mode, median, and location parameter is straightforward. Mode is the default selection since it is the most intuitive for new users. Advanced users who wish to specify the distribution with another parameter will be able to use the above equations to determine the necessary input. An example of log-normal PDF specification in SODA is shown in Figure 7. Note that the mode was specified as 0.2, which is clearly reflected in the location of the peak.

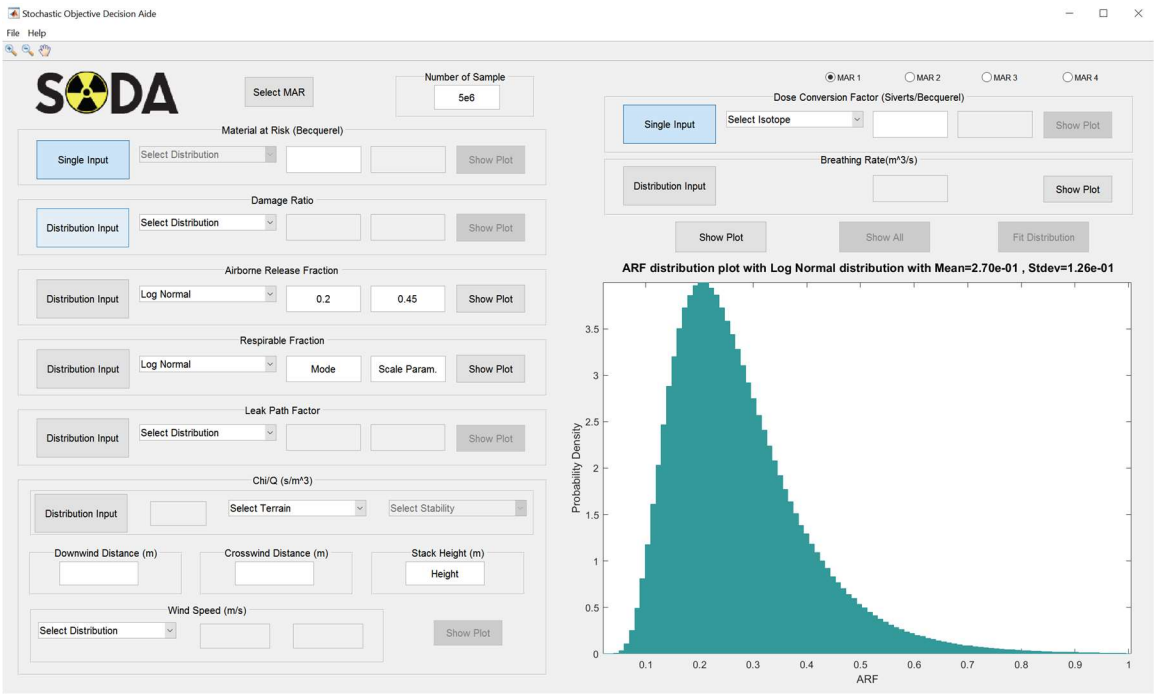


Figure 7. The SODA program in action, displaying a log-normal PDF for ARF, with a mode of 0.2, and a scale parameter of 0.45.

6. Parameter Control

6.1. Introduction

In support of the objective to make the SODA program easy to use, comprehensive program-wide parameter control was added. This feature set allows the software to trap errors without crashing the program, as well as provide feedback to the user as to what caused the error or the problem. This type of feedback can include notifications of invalid inputs, missing inputs, and more. Providing useful and easy to understand feedback to the user is essential to lowering the barrier of entry to a new piece of software.

6.2. Implementation

The parameter control system in SODA was setup using functions which address parameter checking at different levels. The innermost level function is used to determine what category an input belongs in. For SODA, it matters whether inputs are numeric or non-numeric, whether the numeric input is positive or negative, whether the input is finite or not, whether inputs are between zero and one, and finally, whether the input is smaller than 10^{-3}. Thus, the innermost function analyses an input field and returns a flag, depending on the checked parameters that were mentioned previously.

The next level function in the parameter control scheme is called upon when checking is required. This function is aware of the type of input being analyzed, and it checks the return from the innermost function against a lookup table to determine whether the input is valid for the parameter that was specified. Thus, for a single input parameter check, the calling function can pass arguments to the second level parameter checking function, and get a result back stating whether the input was valid. If the input is invalid, a message is displayed to the user giving them appropriate feedback.

The final and outermost level function for parameter control addresses the need to simplify the parameter checking code when it is called in routine functions. When SODA is being used to obtain a CED result, an entire MAR selection worth of inputs must be checked. It would be very cumbersome to call the second level function many times to achieve this. Thus, the third level function automates checking for all available inputs to check whether a CED calculation will proceed without error. Since the second layer function provides the feedback to the user, the third layer function merely needs to automate checking each input, and the other behaviors are performed automatically.

7. User Defined Distribution Integration

7.1. Module Communication

The user defined distribution feature was conceptualized and created in its initial state by Mary Tosten. [6] Upon receipt of the initial feature, it was necessary to establish inter-module communication between SODA and the UDD GUI. The same approximate technique was used for this communication as was used for the MAR Selection GUI. This involves passing the class object from SODA to the UDD GUI. After the user specifies the distribution, it is saved to a temporary variable, and sent back to SODA. Once returned to SODA, analysis methods are applied to determine the validity of the distribution. If the distribution passes the analysis, it is saved to the appropriate variable in the object by calling the appropriate methods.

7.2. Sampling

At the time the UDD feature was added to SODA, it was essential that the integration was performed quickly. To achieve this, a sampling strategy was needed that was both flexible and simple, so that it could be quickly developed, and integrated into the SODA

program in a straightforward way. The algorithm that was implemented is able to operate on UDDs with any allowable number of intervals, and any configuration that can get past the checks that take place before saving a UDD. The downside to the algorithm used, is execution speed. Ideally, this process, which uses many loops, would have incorporated multi-threading to expedite the calculations. Optimization, however, was a lower priority item; as a result, the UDD sampling algorithm remains somewhat slow. Options for improvement in this area are discussed in the future work section of this paper (Chapter 10).

After developing a sampling algorithm, the UDD feature was ready to be added to the show plot button functions for both parameters and CEDs. Since the algorithm was compact, the code was copied and pasted into each function it was used in. At this point, the UDD feature was able to be used throughout SODA. More discussion of this process can be found in chapter 8.

8. Discussion of SODA Commit Log

View Appendix B to see the ordered list of raw commit log changes. The following section gives commentary on the development process of SODA while following along with the raw log. This approach provides insight into the thought processes employed in development, while also putting into perspective the amount of work which was performed in support of the SODA project. Next to each commit message header, the major feature being addressed in the commit is provided for perspective.

Commit Message 1: (MAR Expansion)

After setting up the Git repository, hosted by GitLab, the first feature to be added to the SODA program was the expansion of MAR selection. This part came in two phases,

the first being a MAR selection tool, the second part being the ability to perform multiple MAR calculations in the SODA framework. In the first development Commit to the repository, the files for the MAR Selection GUI figure and code were added.

It was decided to go with a multi-file approach at this stage, to allow for more in-depth tools to be employed. One of the priorities in a program like SODA is user friendliness. A MAR selection tool could have been as simple as a drop-down list in the SODA main GUI, but this would be cumbersome to use, and not very aesthetically pleasing. Instead, a new GUI was adopted, taking on a common style for these types of look up databases of material data, a periodic table. Rather than a long extensive drop-down, the user can click an element, and get a list of available isotopes. Furthermore, this allows for easy visual feedback for the user, to determine what data is available.

Also, in the first commit, some visual improvements were made in the SODA main GUI. It is important that software tools look as good and as professional as possible, within the scope of the work. An easy way to add a measure of professionalism to a GUI-driven piece of software, is to make sure that UI elements have equal spacing, and are aligned on a common grid. Accordingly, changes of this nature were made to SODA. In MATLAB GUIDE, each object on a GUI has a property list, viewed with the property inspector. These give the default properties for a given object including color, displayed text, font size, location etc. In GUIDE, the mouse can be used to drag objects about the GUI, and they can be simply placed in this fashion. If quality is desired, the property inspector can be used to manually line up objects with great precision. The location parameters in the property inspector can be edited numerically. Once an object in a “row” has a finalized location, the Y location can be copied to all other objects on this row to perfectly line them up on the

row. Similarly, the X coordinate can be used to align the left edge of objects in a column. A good example of this is in the MAR selection tool, where there are a great many buttons. If manual alignment was not used, this tool would look unpolished and unprofessional; an aesthetic which should be avoided in software which is to be distributed to a customer.

Finally, the strategy in this tool development should be discussed. When developing a tool with considerable functionality, there are two main issues to consider for a GUI: Layout and Functions. For this tool, layout is the first consideration. The reason for this choice, is that a MATLAB GUI uses event driven code to execute its functions. For example, a function can be called when a button is clicked, or when a new selection is made in a drop-down list, or a new selection is made in an array of radio buttons. GUIDE creates these event functions for each object. By setting up the layout first, a coding framework is already placed in the m file (MATLAB code file), giving a set of events to develop the code behavior around.

Commit Message 2: (MAR Expansion)

After getting the initial layout of the buttons set up in the MAR Selection GUI, some strategies were needed to simplify the code development of the back end of the tool. In a GUI with hundreds of objects present, it is very important to develop sensible naming conventions for the objects on the GUI. For the MAR Selection tool, there are over 100 element buttons which make up the periodic table portion of the tool, as well as 7 isotope buttons, several text fields, and etc. For the element buttons, the naming convention chosen was to name the buttons after the element that they represent. For example, the Hydrogen button has the tag ID “element1” while the Uranium button has the tag ID “element92” which allows the software developer to easily reference each of these buttons inside the

code. Similarly, the isotope selections have the tag ID “isotopeX” where X is the number of the button in standard left to right ordering. The other advantage to naming the objects in this manner is the way the code interacts with this naming scheme. For example, for an element button, it is easy to get the atomic number of the element from taking a known part of the tag string for the button which called a given function. This trick will be discussed in more detail in a later section, but the idea can be applied in many cases.

Commit Message 3: (MAR Expansion)

At this point, the code begins to take advantage of the highly general layout and naming schemes that were employed in the layout phase of the development. There are 112 element buttons in the MAR Selection tool, but each and every one of them does exactly the same thing. It simply fills in the isotope boxes with available isotope data for that element. Given the commonality in function between each element button, it does not make sense to write specific code for each button. Instead, a general function is used, which performs this process for every element button. In order for this single function to work for all buttons though, a way for the function to know which button called it is required.

The function which does all this work is titled GetAvailIso, which is short for get available isotope data. In order for the function to know which button called it, the function takes an input argument, which is the “handle” to the object which called it. For any button press callback function in MATLAB GUIDE, the arguments to the callback include hObject, which is a handle to the calling object, and handles, which is the list of handles to all objects in the GUI. In order to take advantage of this, each call to GetAvailIso includes passing the hObject handle from the calling object to the function. At this point, our convenient naming scheme begins to reveal its effectiveness. The function takes the handle

to the calling object, and extracts its tag parameter. Since all of the tags start with "element," the function is able to get the number after that word from the object. At this point, the function knows the atomic number for the element that data is needed for. Due to the way this code is written, the call to the function for all element buttons is exactly the same. This makes the programming for the behavior of the GUI very easy, as a single line is copied and pasted into each callback for each element button. No case specific code is required. A similar strategy is also used for the isotope buttons.

At this point in development, the code was not in place to actually look up the data. This is an important point; behavioral functions, which help to define how the code will execute, can be laid out without being fully fleshed out. Looking at what data needs to be passed, what tasks will be repeated, and which tasks will not be repeated, will help to identify the framework. Lay out the code appropriately; this is the chance to create a solid framework. Once a solid framework is in place, and the behavior of the code is understood, further development of the "details" is made considerably easier. This style of development can be thought of as an organism analogy. This first stage is laying out the skeleton. The skeleton is the base upon which the rest of the functional systems are placed. Next, the vital organs must be included. In a program, these are your functions which do the heavy lifting, meaning that they perform the core functions needed in that program. After these core function, things get fleshed out, that is, secondary functions and quality of life improvements are made. Finally, the skin is laid on, which can be thought of as the final polish in the program. This is where minor errors are corrected, final aesthetics are set, and the code is turned over to QA. At this point, for the material selection aspect of the MAR expansion, the skeleton is in place. Thinking more about what skeletal development was

needed for the multi material case would have been beneficial, and would have saved time later.

Commit Message 4: (MAR Expansion)

At this point in development, there were functions in place to parse which element needs data, and a framework for how that data will be displayed. Now, a database for this data is needed, so that the code is able to do a lookup, determine what data is available, and send that data to the isotope boxes. The data that needs to be kept for the material lookup, is isotope mass, and the corresponding dose conversion factor (DCF.) For ease of use, a comma separated variable file (CSV) was chosen for this database. In order for the code to understand the formatting in the file, the CSV file is constructed so that each row is an element. The code can then use element number plus one as the index variable for the row. The first column gives the element number, mainly for ease of use when editing the database. The next 7 columns are then isotope numbers, with the following 7 columns being the corresponding DCF for a given isotope mass number. The limit of 7 was chosen mainly because there is only room for 7 isotope buttons in the layout of the MAR selection tool.

Initially, DCF values for adults, with a medium absorption rate, were used in the database. Later (Commit 6) this was changed to the highest adult DCF value, in order to be appropriately conservative. The format of the MAR database does allow individual users to input the DCF values that they wish to use. For example, if a user wished to compare the dose consequence in rapid and slow absorption regimes, they could adjust the DCF values in the MAR database accordingly, and rerun SODA. The program itself is blind to changes in these values, and will simply pull up whatever is in the file.

Finally, a feature was added to help make clear what data is available in the MAR selection tool. On startup, the program has its default colors on the element buttons, blue for standard elements, green for Lanthanides, and yellow for Actinides. During the startup routine, however, the code checks the MAR Database file, looking for available data. For any element that has no isotope data, the corresponding element button is disabled, and grayed out. This gives a clear visual cue to the user, allowing them to easily differentiate between elements which have data, and elements that do not have data.

Commit Message 5: (MAR Expansion)

At this point, integration into the SODA main program was performed. As a simple, stand-in way to get MAR selection to appear, a button which reads "Select MAR" was added to the SODA GUI, above the MAR inputs of SODA. This location was meant to be temporary, but after many people viewed and worked with it in this format, it was decided that it would remain in the interim location. In order to get the information from MAR selection to SODA, initially, it was decided to have the code search for the SODA GUI, and manually add the selected values to the relevant text boxes upon selection of the export button in MAR Selection. This strategy was reasonable at this point in time, where there was still only single MAR calculation available, but it was later abandoned with the implementation of multi-MAR.

After reaching this functional level, suggestions were sought from the SODA team. It was agreed team-wide that the new tool was excellent aesthetically, and some minor improvements were suggested. These were incorporated shortly thereafter. The big point of discussion that proceeded at this point was how to get it working with multi-MAR.

Commit Message 6: (MAR Expansion)

Commit 6 was simply the correction to the MAR Database, to use the most conservative DCF values. This is discussed in more detail in the Commit 4 section. No code development occurred at this commit.

Commit Message 7: (MAR Expansion)

In order to get the MAR Selection feature working for multiple isotope selections, some changes needed to be made to the GUI to accommodate selection of a given MAR. It was decided to use grouped, mutually exclusive, radio buttons for this selection. In the top right hand corner of both MAR Selection, and SODA, these radio buttons were added. They are identified by number; there was a group discussion on making them change dynamically with isotope selection, but this was abandoned in favor of showing this information in the DCF selection dropdown.

At this time, some other improvements were made to both MAR Selection and the main SODA GUI. On the MAR Selection side, instructions were added to some of the empty space on the form. This gives a new user quick tips on working this feature. On the SODA end, the displayed graph area was adjusted, as well as the form size, to ensure that none of the text around the plot was clipped. This was an issue, as some of the lower text would be cut off by the bottom of the form before this change.

Commit Message 8: (MAR Expansion)

While it may seem unnecessary to comment on the addition of comments to the code, these are actually invaluable to the ability of the code to be developed further at a later date. When a new developer of a code settles down, and begins to work, there is a learning curve. The presence of comments, or lack thereof, can make a huge difference on

how steep this learning curve is. Well commented code is far more readable, as the new developer can typically get some insight into what is going on in the code in a section. This snapshot into the mind of the original author makes it easier to infer what the intended function and behavior of a block of code is. Experienced code authors that have worked on long term projects typically find it is also helpful for themselves. When code was written over 6 months ago, it can even be difficult for the original author to remember what they were doing. On the whole, having well commented code is very important, and this practice should be held in any major software development work.

On the SODA side, some additional improvements and fixes were made. First and foremost, a correction was made to the truncation of what can be called "zero to one parameters." These include ARF, RF, LPF, and DR. Since these quantities are meant to be a ratio of a total, the only values that make any physical sense for them to be is between zero and one. A value of one implies that the entire starting quantity falls into the category, while a zero implies that none of the starting quantity falls into the category. For example, a DR of one implies that all MAR was affected by an incident, while a DR of zero implies that none of the MAR was affected. In the original release of SODA, the truncation was taken between zero and infinity, rather than from zero to one. This problem would only affect the answer when a distribution input was used where the distribution had a non-zero probability to have a number greater than one. This was corrected in this commit. Secondly, the title of the plot was repositioned to correctly center it over the axes, as a follow up to the adjustments made in the previous commit.

The last area of additions in this commit were the creation of the SODA_Parameters class for the SODA program. At this stage, this was not yet added to the main code base,

but was being prepared for integration. During the process of attempting to figure out how to manage the data flow from MAR Selection to SODA and back, a discussion was made with a software professional of over 30 years. His advice was simple: use object oriented programming (OOP). After some research, it was discovered that MATLAB does indeed support OOP, and has some tricks to make it easy to get started. This idea behind this concept was to lower the number of global variables, and reduce the number of individual pieces of information that needed to be passed between files. As an added bonus, classes can also have integrated methods, which are functions that act on data contained within the class. To make things very simple, a handle class was selected, and the get and set methods normally used in MATLAB were inherited. This allows the developer to use the same get and set commands used on GUI objects on the SODA_Parameters class.

With the introduction of OOP to SODA, the only variable that needs to be passed between files is the global instance of the SODA_Parameters class. This also allows the SODA program to have access to all data selected in the MAR Selection tool intrinsically, rather than relying on data sent over from MAR Selection directly. The persistance of this data is also mandatory for the multiple MAR option in SODA, as the GUI gains the ability to "remember" a setting made in a MAR which is no longer selected. At this stage, the idea behind SODA_Parameters is to carry around any data which needs persistance in the object, keeping the count of global variables to an absolute minimum.

Commit Message 9: (MAR Expansion)

The focus of this commit was to fully integrate, and take advantage of, the addition of the class object to the SODA project. With the multi-MAR radio button setup already in place, the task here was to let this selection function behave as it was intended to, and to

save all of the needed data between the selections to the class object. In this commit, the behavior of MAR Selection was changed so that any data which is selected by the user is saved to the class object, rather than being simply transferred to a text box on the SODA main GUI. With the class object holding all of the data being specified in MAR Selection, there is only a single variable that gets passed from file to file, this being the global instance of the class object. The other advantage for MAR Selection, is that all the data which must be checked for integrity is now within the class object. This means that class methods were developed to validate the correctness of the data entry, before returning the information to SODA. This was an early step in the process to help eliminate hard errors in the code, through the use of input parameter control. From this point forward, any information that needs to be passed around or retained for later calculations, is saved to the class object.

Commit Message 10: (MAR Expansion)

In this commit, the additions needed to fully integrate the multi-MAR functionality were extended to SODA main. The Function ChangeMARState was added, which takes the program through a sequence of steps designed to save the data entered in the currently selected MAR, and then recall the data in the new MAR selection, and put this data into the appropriate text boxes. This function grew throughout later development, and additional functions were added to help break up the sequence of tasks taking place. For example, a function was added to perform all of the saving steps, before the changes take place. This allows the change function to be focused on the input of new information, while the saving function has all the steps to save the existing data to the class object. It is usually best to break up tasks in a sensible way, to improve the readability of the code, the flexibility of being able to easily perform repetitive tasks, and in some cases, to speed up

the program. The speed difference often being in how MATLAB handles garbage collection and limited scope variables. This can be corrected with manual garbage collection code, but I digress; this is a topic which will be covered in more depth later in the commit log.

With the ability to select multiple MARs, switch between them, and calculate a result for any individual MAR in place, the next step was to perform a multi-MAR CED calculation. In the first iteration of this technique, the code was not particularly robust, but sufficient to begin to test the feature. The initial technique which was used combined all of the CED results arrays into one very large result array. Fundamentally, this initial technique used a model in which an exposed person is only ever exposed to one MAR. Obviously, this is a flawed model, since in many, if not most, cases, the multiple materials that are affected in a release will propagate together. Since it is not the mission of SODA to perform dry and wet deposition calculations, or particularly complex atmospheric calculations, the model in which materials are mixed makes much more sense. If a user of the code wishes to analyze a scenario in which it is expected that two of the three materials will drop out before reaching a receptor site, then a calculation using just the material which is expected to make it to the receptor site can be performed. Other codes have been created which include the more complex atmospheric modeling which is needed to determine the ratio of a given material which drops out of the plume en route, and those results can be sensibly applied in SODA when the decision maker is ready to take a look at the distribution of expected doses at a given receptor site.

In the new modeling regime, the individual CED distributions are summed, rather than being appended to one another. This models a perfect blend of the released material,

and assumes that the blend is what is absorbed by a receptor. At this point, the multi-MAR calculation feature was limping, and code adding additional robustness was added in time.

Commit Message 11: (MAR Expansion)

Commit 11 was discussed in the preceding section, with regards to the multi-MAR modeling scheme in the sum final result for a multi-MAR CED calculation.

Commit Message 12: (MAR Expansion)

The Fit Distribution feature in SODA, developed by Kushal Bhattari, was designed to take data from the MATLAB workspace and use it to perform a curve fit using the Bayesian Information Criteria (BIC) method. Under the multi-MAR calculation regime, the required information was not saved to the workspace, so the curve fitting did not work. This simple to fix, just a couple lines of code were added to save the final multi-MAR CED result to the workspace variable where the fitting function expected it was sufficient. This was a classic case of the new developer not understanding the full intricacy of the code, and then learning and adjusting.

Commit Message 13: (MAR Expansion)

This step in the changes to SODA revolved around the changing of MAR state, and the handling of less than optimal conditions in the GUI. When a distribution input is selected in SODA, the textboxes that are used to input those parameter values which define the distribution initially populate with the type of information expected. For example, in a normal distribution, the text boxes display mean and standard deviation. Normally, if there is any text entered into a numeric input field, the program should report an error, and let the user correct the typo. However, this means that the default input that populates the the

boxes would also throw the error! To fix this issue, the checking routine for the input fields gained exceptions, to allow default text to not throw an error. In the case that a default entry is found, it is simply treated as a blank entry, and no entry is saved to the class object. In the event that an error message is given for a blank field, such as when performing a CED calculation, the same error is given as if the field was blank in the new regime. Additionally, code was added to bring back the default messages on a MAR State Change, so that the tips would be displayed again for the user to aid them in easily specifying their input parameters.

Commit Message 14: (MAR Expansion)

In Single Input mode for DCF, there were a couple of isotopes that could be selected to get their preset DCF. Since the MAR Selection tool replaces this feature with something more comprehensive, the old presets were removed. In their place, Select Isotope is placed into the drop-down. Selecting this drop-down item will load the MAR Selection tool. After an isotope is selected, it will be displayed in the drop-down menu, unless distribution input mode is selected. In order to make these features more accessible, DCF was set to default to single input mode. There is no loss of functionality to the distribution input mode, but these features would conflict with the options present in distribution input mode, so they are only present in Single Input mode.

The Run All routine, which computes individual CED arrays for each MAR, and then sums them to get the sum total CED, was made more robust. Try, catch statements (a common construct in programming to catch errors before they crash the program) were added to each call of the Get Results function, so that if there is an error, the remaining MAR calculations can still be performed. In the event that an error is caught, the user is

informed that the calculation failed for that material, and that the final answer would not incorporate that data. In order for the calculation to still be able to finish successfully, an appropriately sized array of zeros is passed to the sum CED method in the class object.

Message boxes and Error dialogues were changed so that all had the "modal" property. Doing so causes the program to not allow any interaction with other GUI elements until the message is dealt with by the user. This helps to prevent execution of code waiting on a response, or an unexpected selection of axes to draw to. In some rare cases, the final plot would appear in the message box, rather than in the primary axes. To further aid in eliminating this issue, explicit calls to the desired graphics object were added, rather than allowing MATLAB to assign the plot based on its standard logic when a set of axes are not specified.

Commit Message 15: (MAR Expansion)

This commit was a simple bug fix, due to an oversight when developing the Change MAR State function and other associated functions. In Single Input mode, junk was being saved and/or entered into the second text box for an input block, which is not used in Single Input mode. The bug fix added the conditional statements needed to avoid this behavior.

Commit Message 16: (Log-Normal Distribution)

The content of commit 16 was created by Kushal Bhattari. This change added some new features to the plotting function, displayed new parameters (such as a 95% CI value) and added the Log-normal distribution option to the drop-down menu.

Commit Message 17: (Log-Normal Distribution)

This set of changes dovetailed from the previous commit. The Log-normal distribution option was added to all relevant places in the code. The ability to show the PDF plot for a selected Log-normal distribution was added, and Log-normal was added as an option for DCF. Some changes were also made to bring the multi-MAR calculation and plotting into line with the new setup added by Kushal for a single material run. This included additions to the SumCED method in the class object, to get the 95% CI and median values for the total CED distribution.

Commit Message 18: (Merge)

At this point, the AMaas branch, on which development had been taking place up to this point, was merged back to the master branch. Master should always be a working version of the code, both in debug, and in the installed version. This merge was intended to have occurred as soon as the multi-MAR functionality was added and verified to work, but there were delays in testing the installed version. Once the operability of the installed version was verified, the branch was ready to be merged. Further development would take place on the master branch, so each commit needed to be a working version from this point. To allow for partial work to take place while maintaining a working version, functions can be outlined and added to parts of the code without their full functionality. This tactic was used many times to outline work to be performed, while the functions would be fleshed out at a later commit. The branching feature in Git had been intended to allow for each member of the SODA team to work in parallel, but due to project scheduling and individual factors, this never occurred.

Commit Message 19: (Log-Normal Distribution)

This was another simple bug fix that was corrected as soon as it was discovered. The Log-normal distribution, when selected, was not filling its tips into the text fields in every case.

Commit Message 20: (Log-Normal Distribution)

The Log-normal distribution option was removed for MAR and DCF. The Log-normal distribution is not located in the same manner as some other distributions. When a user inputs information from the MAR Selection tool, this information needs to be reflected in the distribution being sampled. With the location parameter input of a Log-normal distribution not having a direct correlation to a supplied value, there is a potential for an inexperienced user to model a situation which does not strongly resemble the one they are attempting to model. This issue is somewhat alleviated when the input parameter scheme for Log-normal was changed in a later commit, but this option was never added again to MAR or DCF. The necessary code was also added to the GetResults() function so that a Log-normal distribution input could be used in a CED calculation.

Commit Message 21: (Parameter Control)

Additional parameter control was started at this point. It is desired, in a user-friendly program, to provide feedback to a user when there is a problem, and to trap errors so that they do not crash the program. Since the implementation of this parameter control on a wide scale was a sizable proposition, some dummy code was put in place to first start defining the way in which parameter checks would be performed. This feature set was not completed until much later in the project, and the first pass was actually abandoned in the next commit.

Commit Message 22: (Log-Normal Distribution)

In this commit, there were two notable changes. First off, the input parameters for the Log-Normal distribution were modified. As mentioned in the details for commit message 20, the Log-normal distribution default parameters, Location Parameter (Normal Mean) and Scale Parameter (Normal Standard Deviation), are not intuitive to use when trying to generate a distribution. Worse, if a user has a known most probable value (such as a measured value) for a parameter, there is not a straightforward way to create a distribution centered on this value. In order to make the Log-normal distribution feature fit into the goal of SODA to be easy to use, the input parameters were changed, so that instead of inputting the Location Parameter, the user can instead enter the Mode, which is the most probable value. This makes locating the distribution peak extremely straightforward, and allows the user to easily generate a distribution with the most probable value equal to their known value. For power users, the math needed to convert this input to a median (or back to the location parameter) is straightforward, so long as the scale parameter is known. Since the scale parameter scales the distribution in a straightforward manner, it was not deemed necessary to change this input parameter.

The other major changes were performance improvements to the GetResults() function, and other supporting code. One of the big problems with a program like SODA, in terms of performance issues, is that large sample counts will both slow down the calculation time (which is to be expected) and greatly increase the RAM usage. Since SODA can treat each input parameter as a distribution, when the sample count is large, there are many distributions being created and saved in RAM. Each of these results arrays can be Gigabytes in size when the sample count is sufficiently high. A complicating factor, as mentioned previously, is how MATLAB handles garbage collection. Since the

GetResults() function is very large, and contains all the steps to calculate the CED distribution, all of these distribution result arrays end up being saved in memory simultaneously. An easy way to reduce the peak RAM usage presented itself after this realization was made. New code was added to perform some additional garbage collection. For example, once the Source Term distribution is calculated, all of the distributions that were used to calculate it no longer need to be saved in memory. In any case where this type of wasted memory allocation existed, code was added to release this memory. In addition to reducing the peak RAM usage for a given sample count, this also sped up the code by ~30% in large sample count CED calculations. These changes were especially important for the multi-MAR calculations, since even more data has to be kept around until the final summed CED distribution is computed.

Commit Message 23: (Parameter Control)

More work was done on formulating the parameter control strategy. This feature was a lengthy process to implement, and progress is mentioned throughout the latter half of the commit log. There was also a major bug fix. SODA allows the use of scientific notation to enter values; many common values, such as sample count and MAR quantity, are large enough that this feature is needed. There was an issue in the error checking for MAR quantity in the MAR Selection tool. To avoid non-numeric values, the first iteration checked for any non-numeric characters in the text field. The problem with this method is that the ‘e’ or ‘E’ used to denote the exponent of the scientific notation was triggering the error, rather than taking the input. To change this error checking to accommodate the scientific notation entry, the new method uses the str2double routine in MATLAB. This routine will change any compatible string into a double precision number, and return a

value of ‘NaN’ if the entry is not a number. Making the check for NaN, rather than the presence of any characters, fixed the issue while still maintaining the desired behavior in the code.

Commit Message 24: (General)

In this commit, the behavior of the Show Plot, and Show All routines was altered. In order to prevent double calling of functions, or other error inducing scenarios in the code, the buttons which trigger function calls are disabled when the CED calculation begins. In the former version, all buttons were turned off in the beginning of the Show Plot / Show All routines, and then all buttons were turned on. The problem with this behavior, is that turning on all buttons can cause buttons which should be turned off to be turned on. In order to fix this issue, the code first checks whether or not a button is active at the time Show Plot / Show All is pressed. If the button is on, the code sets a flag, and then turns off the button. If the button is off, the flag is set accordingly, and the button stays off. At the end of the routine, the code checks the flag, and turns on the button only if it was an active button when the Show Plot / Show All button was pressed.

Commit Message 25: (UDD)

At this point, the base version of the User Defined Distribution (UDD) GUI was added to the code repository. Mary Tosten performed the work on the UDD GUI up to this point.

Commit Message 26: (UDD)

For this commit, the UDD GUI was prepared for integration into SODA. The GUI was rescaled to better conform to the aesthetic of SODA, and some bug fixes were made.

Commit Message 27: (UDD)

In this commit, parameter control was added to the UDD GUI. The input fields where the number of bins, and the width of a bin, now perform checks to determine if the input to the text box is acceptable. If the input is not within desired parameter limits, the user is informed of the correct parameter limits. A similar scheme to control inputs to the manual mode was added.

Commit Message 28: (UDD)

Considering the UDD feature, it was decided to eliminate the single bin option, as the only normalized distribution over a single interval for a zero to one parameter is the uniform distribution of height one. Error message boxes were altered to have the modal type specification, which requires mandatory response from the user before continuing interactions with the main GUI. Some of the boxes already had this, as this strategy was used in SODA originally, and this change brought the new code up to compliance.

Originally, the UDD GUI featured a standard bar graph. This graph was somewhat misleading as to the nature of the PDF being generated by the user. By featuring the bar over a single value, and not connecting to adjacent bars, this can give the impression that the PDF is only non-zero at the specified values. This, of course, is non-physical for any process being represented in the SODA program. Given the nature of these calculations, what is needed is a PDF which spans the range of data, and a probability defined to sensible sub-intervals. To make the visual cues more in line with this type of PDF, the bar graph was changed to a histogram style, where the bars cover the entire interval, and stop at the edge of the graph, or the start of the next interval. This way, there is visual communication

to the user that they are defining the probability on the subinterval between the limits, rather than a probability of a single value.

Finally, an issue was corrected which could cause a major error in the UDD GUI. The integrated MATLAB function ginput was used for the clicked input method for the UDD. This function, however, is able to return any position on the GUI, rather than being confined to a set of axes. As a result, it is possible to select buttons and text boxes while in ginput mode. The resulting problem is that the button which starts ginput can be clicked while in ginput mode, causing a nested ginput routine to be started. The GUI could not recover from this condition, so the button was disabled while ginput is active.

Commit Message 29: (UDD)

In this commit, the first steps towards intercommunication between SODA and the UDD feature were implemented. As has become standard when integrating new modules into SODA, the passing of the SODA_Parameters class object was added, and the ability to call the UDD feature from the SODA main GUI added. The steps and technique used for the intercommunication between the MAR_Selection GUI and SODA was used as a template for these inclusions to the UDD tool.

Commit Message 30: (UDD)

At this commit, an object-oriented approach was made to managing the data for the user defined distribution. Adding the new parameters to the SODA_Parameter class would have greatly increased the number of parameters specified in the class, so a new class was made for the purpose of storing and recalling the UDD data. The way MATLAB Classes interact when one is instantiated within another class was not how they behaved with the outer class. To deal with this, methods were added to both SODA_Parameters and

UDD_Data which when combined, act as getter and setter methods for the UDD_Data objects. Once these features were integrated, saving routines were added so that UDD's can be saved in SODA_Parameters.

Commit Message 31: (General)

At this point, a needed feature was added to SODA. When the multi-MAR feature set was initally implemented in SODA, the ability to change the numerical inputs was added so that different material properties could be accounted for. However, this was insufficient as it did not allow for different distributions to be selected on different materials. Work on the UDD feature brought this deficiency to the forefront and it was decided to add the capability to select different distributions on different selected materials. In addition, the ability to have a distribution selected for one parameter, on one material, no longer precludes having a single value specified for the same parameter on different materials.

Commit Message 32: (UDD)

In this commit an algorithm was developed which allows one set of code to sample any allowed UDD randomly. This is added initially to allow the UDD to be plotted in the same way that SODA can plot other distributions independently. This also formed the prep work for implementation into the main GetResults() function.

Commit Message 33: (UDD)

For this commit, the UDD option was implemented for all parameters where the UDD is allowed. At this point, it became possible to use a UDD as a selection when computing a CED. The algorithm used to sample the UDD is somewhat inefficient with

respect to the integrated MATLAB methods used for the built-in distributions. This was considered acceptable, as the UDD feature is designed to let a user model something quickly, rather than to get a precise result. The inefficiency of the UDD sampling is not too noticeable if smaller sample counts are used. The reason for this behavior is the many nested for-loops that are used. It is possible that this could be multi-threaded to improve runtimes, but this was never implemented. The advantage of the method that was used here is that the code is fairly simple, and works on any allowed UDD. This saved development time, which was crucial at this stage.

Commit Message 34: (UDD)

In this commit some older code, in the MAR_Selection GUI, was modified. It was discovered that the MAR_Selection GUI would override a UDD selection if it was made. This was due to how MAR_Selection saved data selections, and how these were applied.

UDDs gained some parameter control in this commit, to prevent non-physical UDD's from being specified for a parameter. For example, a damage ratio may not be less than zero, nor may it be greater than one. With the changes made, if a UDD has a non-zero probability of generating a value outside this range, it is not saved, and the user is prompted to try again. This change in behavior of the code prevents SODA from reporting a result based off of non-physical assumptions in the input.

Some work was also started for program-wide parameter control at this stage. When running an individual parameter distribution, the code checks that the values of the input parameters meet expectations, and prevents execution if they do not. The user is informed as to the nature of the problem so they can correct it.

Commit Message 35: (Parameter Control)

At this stage parameter control was implemented program wide. SODA is now able to report issues in inputs to the user, and prevent crashes of the program by preventing a function call to the main GetResults() function unless the inputs all pass the tests laid out in the parameter control. Consistent parameter control and user feedback is important in a user-friendly program.

The strategy for parameter control was to use layered functions, each of which performs a given role in the parameter checking. The innermost function simply reports back what category of data is stored in the item in question. Catagories which are used include whether the input is zero, between zero and one, numeric or non-numeric, etc. The next layer is a function which compares the results of the innermost function to what is permissible for the argument being analyzed. For example, if the argument is the mean value for Damage Ratio, the value needs to lay in the interval (0,1] in order to be valid. If the innermost function returns that the value in question is numeric and on the interval, this function gives the calling routine the okay to proceed. If not, it gives a return code which tells the calling routine that the input is invalid.

To further simplify the parameter control implementation, automated routines were added for individual parameter input checking and full MAR checking. These work together to simplify the code that needed to be added in the other parts of the program.

Commit Message 36: (Parameter Control)

In this commit some bugs had to be corrected. The parameter control functions needed to have an additional case added which covered the uniform distribution. Fortuantely, due to how the code was designed, only a case statement in the inner function

needed to be added. Another issue which was identified allowed a failed UDD specification to save an old result rather than clearing the old result and having the user retry. A crash in the UDD GUI was also corrected.

Commit Message 37: (General)

Over the course of the previous commits, many new additions to the program were made. Some of these changes altered how information is stored in SODA, and what information needs to be kept in order to preserve a session between uses of SODA. Due to these changes, the original save function for SODA was insufficient to work with the existing system. To change this, the SODA_Parameters object was saved to the file, and a recall system put in place to bring all of the stored information back into the object upon a load.

Commit Message 38: (General)

The changes in the previous commit were not without error. Saving the entire SODA_Parameters object was not a feasible choice, as the information stored in the object can become very large when calculations have taken place with high sample counts. To deal with this, the large entries in SODA_Parameters get deleted after they are used. Since SODA has no way to call up this information again after generating and plotting it, this has no ill effect on other features. In addition to allowing the save feature to work without error, this change has the added benefit of reducing the program's static memory usage.

Commit Message 39: (General)

In this commit another old feature was updated. The old image saving feature in SODA used the 4:3 aspect ratio, and clipped some of the plot title text. To correct this, a

16:9 aspect ratio was adopted, and the output image scaled so that nothing was clipped. The zoom and drag features were checked as well and no errors or issues were found. At this point, the coding part of the project seemed nearly over, although a couple outstanding issues remained.

Commit Message 40: (General)

Upon integrating the updated help file into SODA, it was discovered that cross platform code for file recall was not in place. Mac™ and Windows™ systems need different MATLAB code to open the help file PDF (file format) from within SODA. For Mac™, a command is sent directly to the terminal, while Windows™ utilizes a function. Previously, this code had two commented out versions, with one being activated based on whether a windows or mac installer was being created. In the new version, MATLAB checks for the OS version, and calls the appropriate code. Some redundancy was added as well, to account for possible selection of non-default file locations.

Commit Message 41: (General)

Incorporating a critique given by Alan Levin of DOE, the SODA can-shaped icon used in the program was modified. The DOE logo was to be removed and replaced by an INL Logo. Lacking the original template for the can, GIMP software was used to edit out the DOE logo, and add the INL logo. After changing the can, the files were changed so that the updated can would be used for the splash screen and the icon in the installer.

Commit Message 42: (UDD)

When adding the UDD sampling code to the GetResults() function, an error was introduced. One of the for-loop indexing variables had the same name as a

distribution/single input flag for GetResults(). This caused that variable to always be treated as a single input anytime a UDD was specified. Changing the indexing variable name solved this issue. It is notable that without the fix, selection of a UDD to the parameter which was defaulted to single input resulted in a crash, since the input was "User" rather than a number.

This last commit took place after the official completion date of SODA, and marked the end of code development.

9. Respirable Fraction Distribution Calculation

9.1. Definition of Respirable Fraction

When considering a methodology for the determination of the distribution of the respirable fraction of a material, it is logical to begin with the definition of the respirable fraction. In other words, the respirable fraction is the fraction of radioactivity in an airborne release that is within particles of "respirable sizes." For the sake of discussion, consider respirable sizes as particles which lie between an upper and lower limit of size. These limits are selected based upon the effectiveness of particles at depositing themselves in the deep lung, where they will be absorbed into the bloodstream.

In a discussion of respirable aerosols, it is useful to consider three classes of particles. The first and largest class of particles are those that possess sufficient inertia to overcome aerodynamic entrainment in the air stream being directed towards the lung. These large particles will collide with the walls of the mouth and trachea, where the body can easily expel them. Although these large particles are able to impart dose while they are present in the body, their residence times are low. When calculating CEDs, the dose is a lifetime figure. As a result, when considering an aerosol of radioactive material, the material which

is absorbed into the blood stream is able to impart much more dose than the material which is rapidly removed through mucus processes, or merely inhaled and breathed out again. Due to this, the particles which are not absorbed into the deep lung are ignored. This assumption is good for long half-life radionuclides, while it is questionable for short lived ones. The next class of particle to be considered is the smallest category. These tiny particles are so well aerodynamically entrained that they will follow the flow of air throughout the trachea and lung. These small particles essentially act as a gas, and will not be absorbed in most cases [7]. Many conservative calculations will treat these very small particles as part of the respirable fraction, while other models will provide a non-zero lower limit. A more detailed discussion of respirable particles can be found later in this chapter.

The final class of particles, and the one of greatest interest to this discussion, is the intermediate size range. These particles are sufficiently aerodynamically entrained to avoid deposition in the mouth and trachea, but weakly entrained enough to still leave the gas and deposit in the lung. Once deposited in the lung, and exposed to the bloodstream, the radionuclides in question will impart their maximum dose to the body. Thus, when considering lifetime doses, these particles are the ones of interest.

Since measurements of particle size distributions are not made with respect to radioactivity, it is important to formulate a definition of respirable fraction which does not rely on radioactivity. If it is assumed that the radioactive components of the aerosol are uniformly distributed, then an assumption can be made that the ratio of radioactivity in the respirable range to the total radioactivity is equal to the ratio of the mass in the respirable range to the total mass. Cumulative mass fractions with respect to particle size are common

in aerosol analysis, and relationships can be drawn between particle size distributions and mass distributions as well. Mass ratios can, therefore, be used to calculate RF:

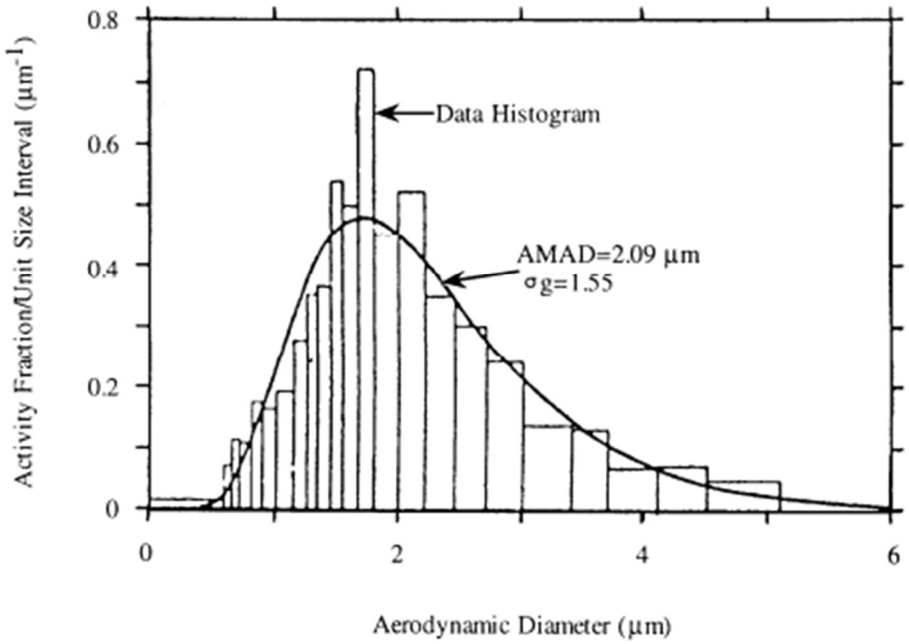
$$RF = \frac{\int_{LL}^{UL} M(D)dD}{\int_{0}^{\infty} M(D)dD}$$

Where RF is respirable fraction, UL is the upper limit on respirable particle size, LL is the lower limit on respirable particle size, and M(D) is the distribution of mass with respect to particle size, with the independent variable D representing Aerodynamic Equivalent Diameter (AED) of the particles.

9.2. Computational Technique

9.2.1. Introduction

The definition provided in 9.1 for RF acts as a basis for the calculational technique employed in this study. The objective of this RF study, however, is not to determine RF values, but to determine how RF is distributed. Monte Carlo methods can be applied to this analysis, and used to determine the distribution of RF. Put simply, this technique models the effect of uncertainty in a measured particle size distribution on the calculated value of RF. To simplify the model, the curve fitting parameters of the input data set are modelled as uncertain, rather than the data itself. This is a necessary compromise when working with limited information. To clarify this distinction, consider Figure 8, which is reproduced from reference [8].



Aerosol data with fitted log-normal distribution function for a sample collected with the spiral duct aerosol centrifuge of plutonium oxide fume aerosol from a burning plutonium droplet 400µm in dia. with the probability density of the distribution plotted vs. the aerodynamic diameter and showing the activity median aerodynamic diameter (AMAD).

Figure 8. Raw data used for RF distribution determination. Figure is reproduced from reference [8].

Ideally, to use the uncertainty in measurement to determine the RF distribution, the error in the height of each bar in Figure 8 would be varied according to its corresponding PDF. This is typically taken to be a normal distribution, where the standard deviation would be reported along with the measurement. Each Monte Carlo sample would randomly select the height of each bar, according to their corresponding PDF, generate a new log-normal curve fit, and then take the RF integral on the result. By performing a large number of samples in this way, a distribution for RF could be developed.

In the case of this study, the exact heights of each bar in Figure 8 and the corresponding error in the measurements is not available. To make do with what is available, PDFs are estimated for the curve fitting parameters of the log-normal fit to the measured data. This allows the calculation to treat the uncertainty in the measurements as being analogous to uncertainty in the curve fitting parameters of the log-normal fit. This is supported by the fact that varying the data according to its associated uncertainty would result in a variance in the curve fitting parameters for each individual case. This is modeled by varying these parameters directly, rather than developing the PDF for the curve fitting parameters by directly varying the data, and then sampling the resulting PDFs. This will limit the accuracy of the resulting distribution, but allows this study to demonstrate the results of this computational method.

9.2.2. Selecting PDFs

To perform the computations with available data, PDFs must be selected for the two curve fitting parameters. Consider again Figure 8, which shows data from DOE HDBK 3010-94 Appendix A, Page A-88. [8] It shows a log-normal fitted plot of particle size with an AMAD of 2.09 μm, and a geometric standard deviation of 1.55. By the following relationships, the scale parameter can be determined:

$$Log\ Normal\ Median = e^{\mu}$$

$$Log\ Normal\ Geometric\ Standard\ Deviation = e^{\sigma}$$

Where μ is the log-normal location parameter, and σ is the log-normal scale parameter. For a geometric standard deviation of 1.55, a scale parameter of 0.4382 is used. These values for the median and scale parameter are the most probable values for the curve fitting parameters. Given this, the values should be used as the mean of a normal

distribution, or as the mode of a log-normal distribution. It should be noted that the actual PDF, if determined by the methodology outlined in the preceding section, would likely not look exactly like either of these distributions. The use of a normal or log-normal distribution of medians and scale parameters is an estimate on the uncertainty shape of these parameters, made due to lack of information. When comparing the normal and log-normal distributions, the normal distribution was selected in this study. The reason for this, is that the log-normal distribution has a very long tail, providing a non-negligible probability of values for the curve fitting parameters that is much larger than the accepted value. This is non-physical, and thus, the normal distribution was selected.

9.2.3. Generating the RF String

After selecting or calculating PDFs for the curve fitting parameters, the Monte Carlo calculation of the RF distribution can proceed. To generate a single sample for RF, initially 2 random numbers must be generated; the first uses the PDF for median, and the second uses the PDF for scale parameter. Using these randomly generated curve fitting parameters, a log-normal distribution is generated. Next, the log-normal distribution that was generated is used as a PDF, and a string of numbers is generated. These numbers are AMADs for particles in the distribution. To calculate an RF for this set of particles, a conversion to mass must be made. Using a spherical approximation for the particles, the result is:

$$RF = \frac{\int_{LL_m}^{UL_m} \rho * \frac{4}{3} * \pi * (\frac{D}{2})^3 S(D) dD}{\int_0^{\infty} \rho * \frac{4}{3} * \pi * (\frac{D}{2})^3 S(D) dD}$$

Where ρ is the mass density of the particles, S(D) is the distribution of particles with respect to their size, UL_m and LL_m implies limits of integration in mass space. The other variables are as they were previously defined. Simplifying:

$$RF \quad \frac{\int_{LL_m}^{UL_m} (\frac{D}{2})^3 S(D) dD}{\int_0^{\infty} (\frac{D}{2})^3 S(D) dD}$$

From the formulation above, apply the (D/2)^3 weighting factor to the number string. The string is now a list of particle masses. The integrations are performed numerically, resulting in an RF sample. Repeat this process untill the sample count is reached to obtain the result. A flowchart of this process can be found in Figure 9; blue denotes inputs and outputs, while the encircled orange steps are repeated for each sample.

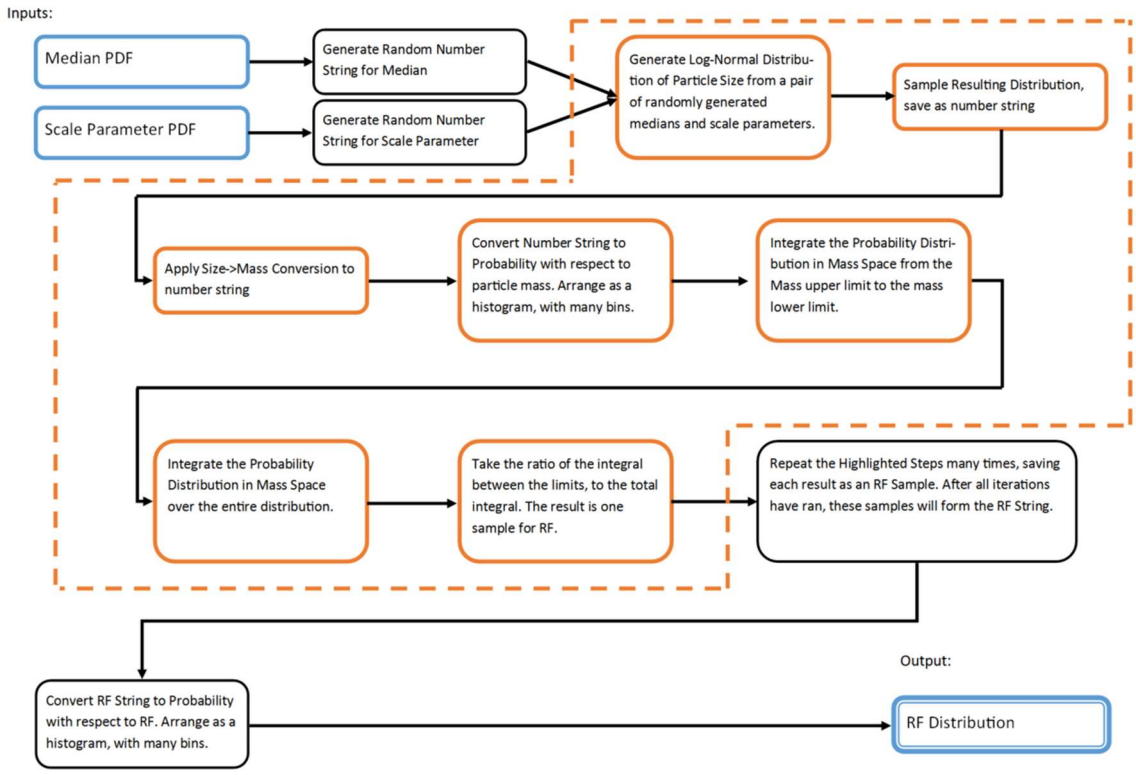


Figure 9. Flowchart detailing the process used to compute the RF Distribution.

9.2.4. Selection of Limits

9.2.4.1. Particle Deposition and Regimes

Particle deposition in the human respiratory tract can occur through five different processes. These processes include sedimentation, inertial impaction, diffusion, interception, and electrostatic deposition. The diffusion process has some significance for particles of sizes less than 0.5 μm. For particles above this size threshold, the processes which are most important for inhaled aerosols are sedimentation and inertial impaction; interception and electrostatic deposition do occur, but they are dominated by the other processes. These two primary processes are regulated by the aerodynamic equivalent diameter (AED) of the particles in question [7].

When considering the inhalation of particles, and the fate of these particles, the size classes must be considered in detail. The largest of particles, those greater than ~30 μm AED, will be deposited in the airways of the head. Depending on whether the individual is breathing through the nose, or through the mouth, head airways are defined as the nasal passages down to the larynx, or the mouth down to the larynx. The nose is a more effective particulate filter than the mouth, so nose breathing will lead to less particle penetration into the airways than will occur in mouth breathing. Since there is a preferential shift in all people from nose breathing to mouth breathing under exertion, there is a coupling between breathing rate and lung deposition. [7] This will be ignored in this study, however, and conservative assumptions made instead.

The next size class of particles are those which are between 10 and 30 μm AED. These particles will penetrate further into the airway system, passing the head, but not

reaching the alveolar region of the lung. Particles at 10 μm AED have ~1% penetration rates into the alveolar region of the lung. This can be considered the hard upper limit for alveolar penetration. Thus, particles below 10 μm AED can be considered as having a chance of reaching the alveolar region of the lung. [7]

In the particle size range below 10 μm AED, the rate of alveolar penetration is a strong function of particle size. At the high end of the range, at 10 μm AED, only around 1% of particles penetrate to the alveolar region. This figure increases as the particle size goes down to 2 μm AED, where it is maximal. Note, that 2 μm AED is the particle size that achieves maximal penetration and deposition, not a particle size which achieves 100% penetration and deposition. After the 2 μm AED point, the percentage of particle deposition reduces back down to around 10% at 0.5 μm AED. Below 0.5 μm AED, the diffusion mechanism becomes appreciable, once again raising the deposition probability somewhat. A large percentage of particles below 2 μm AED are simply exhaled again. [7]

9.2.4.2. The "True" Model of RF

After consideration of the information presented in section 9.2.4.1, it becomes clear that the true definition of RF would rely upon the deposition rates with respect to particle size. In the definition provided in the earlier sections, it is assumed that all particles between the upper and lower limits are completely absorbed into the bloodstream:

$$RF = \frac{\int_{LL}^{UL} M(D)dD}{\int_{0}^{\infty} M(D)dD}$$

Where RF is respirable fraction, UL is the upper limit on respirable particle size, LL is the lower limit on respirable particle size, and M(D) is the distribution of mass with

respect to particle size, with the independent variable D representing the AMAD of the particles. Modifying this equation to take into account the size dependent deposition probability:

$$RF = \frac{\int_{LL}^{UL} P(D)M(D)dD}{\int_{0}^{\infty} M(D)dD}$$

Where P(D) is the probability of particle deposition with respect to particle size in AMAD. The other variables are as they were previously defined. This second equation, if all the quantities were well defined, would calculate the true RF for an aerosol. The upper and lower limits, in this case, would be chosen to bound the non-zero region of P(D) between 0 and positive infinity.

9.2.4.3. Selection of Upper Limit

As discussed in earlier sections, the upper limit on penetration to the alveolar region of the lung is at ~10 μm AED. Due to this, the upper limit for respirable particles is typically chosen to be 10 μm AED. This choice, however, is very conservative, as particle deposition in the alveolar region at that size is only around 1%. [7] In the simplified RF model, which is typically used, it is assumed that all particles between the upper and lower bounds achieve 100% penetration and deposition.

9.2.4.4. Selection of Lower Limit

For the reference case calculation, which is used to verify the model, a lower limit of 0 was chosen. This matches the limits which were used in the reference material, allowing direct comparison to be made. Ideally, the P(D) function would be known, allowing the lower limit to remain zero while accounting for the actual deposition rate. Since this is not

the case, other limits must be chosen. The correct choice of lower limit in the simplified model will depend upon the data set in question. If a data set includes a high probability of particles being present at sizes less than 0.5 μm AED, a lower limit of zero may be chosen to account for diffusion of the small particles. This choice, however, will overpredict RF in the simplified model. For distributions which do not contain appreciable particle densities at sizes below 0.5 μm AED, the choice of lower limit is somewhat more complicated. The lower limit choice of 0.5 μm AED could be made, but once again, this will overpredict RF. Choice of a lower limit at or above 2 μm AED would cause an underprediction in most cases. For this study, which offers demonstration estimates on the RF distribution for PuO_x, a compromise of 1 μm AED was chosen for the lower bound. Given the greatly conservative assumption that all particles between the upper and lower bounds deposit in the alveolar region, this calculated RF is sufficently conservative.

9.3. Results

The technique detailed earlier in the chapter was made into a MATLAB program which allowed for simple specification of curve fitting parameter PDFs, and the calculation of an RF distribution. To save time, some code was reproduced from SODA, while other code was written from scratch to perform the Monte Carlo sampling of RF. The first distribution to be computed was the reference case calculation. This used the reference particle size limits of 0 to 10 μm AED, with normal distributions as PDFs for the curve fitting parameters on the log-normal curve fit of the data. To show that the model would make the same predictions to reference, very low uncertainties were chosen to emulate a single point calculation. The results are shown in Figure 10.

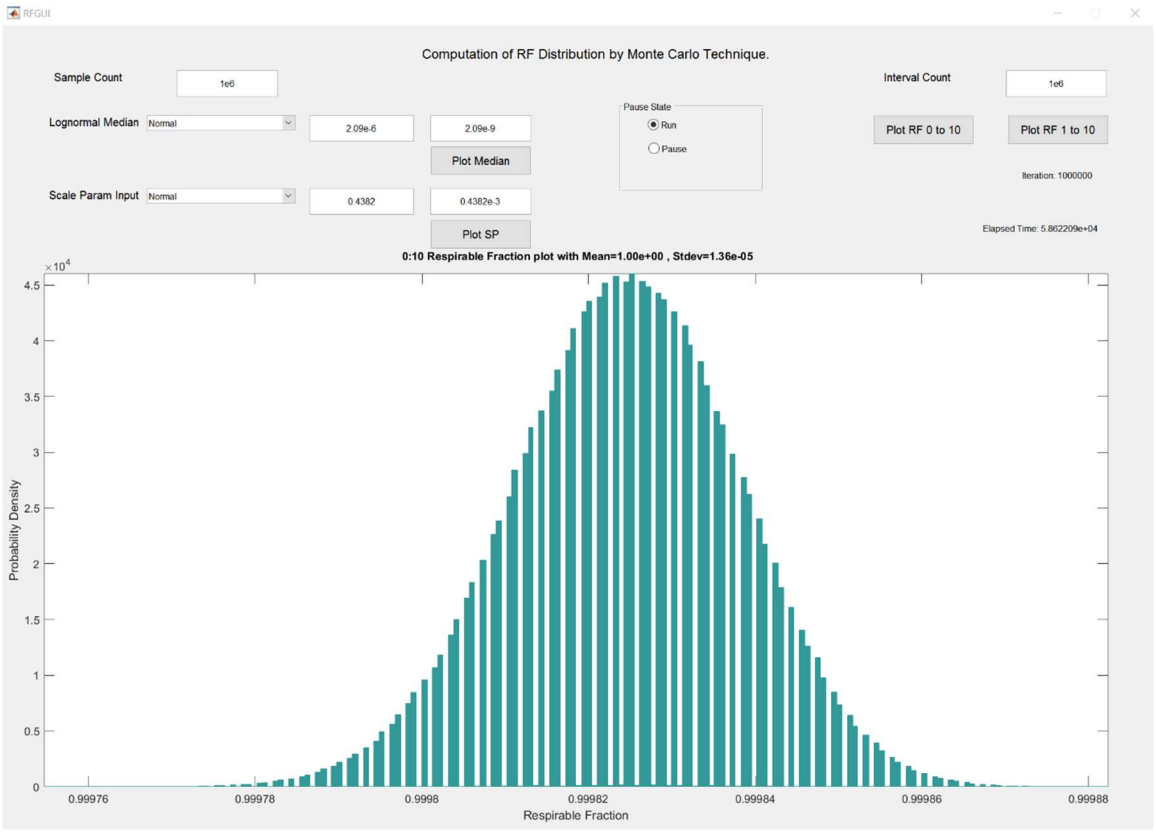


Figure 10. RF Limits from 0 to 10µm, showing the result approaches one as the uncertainty approaches zero (0.1%).

The reference calculation shows that for very small uncertainties, and the choice of the same upper and lower limits, the result is the same (to within 3 decimal places). This result supports the calculation methodology, as the same result can be obtained when the uncertainties in the curve fitting parameters approach zero. The gaps in the distribution are present due to the bar graph mesh being too small. The X axis of this plot only has an extent of 0.00012. The use of more samples would fill the plot, but more samples were not taken due to the extensive time required to obtain these results. After obtaining this result, the uncertainties were increased in order of magnitude steps, and the chosen lower limits applied. These results are shown in Figures 11 to 13.

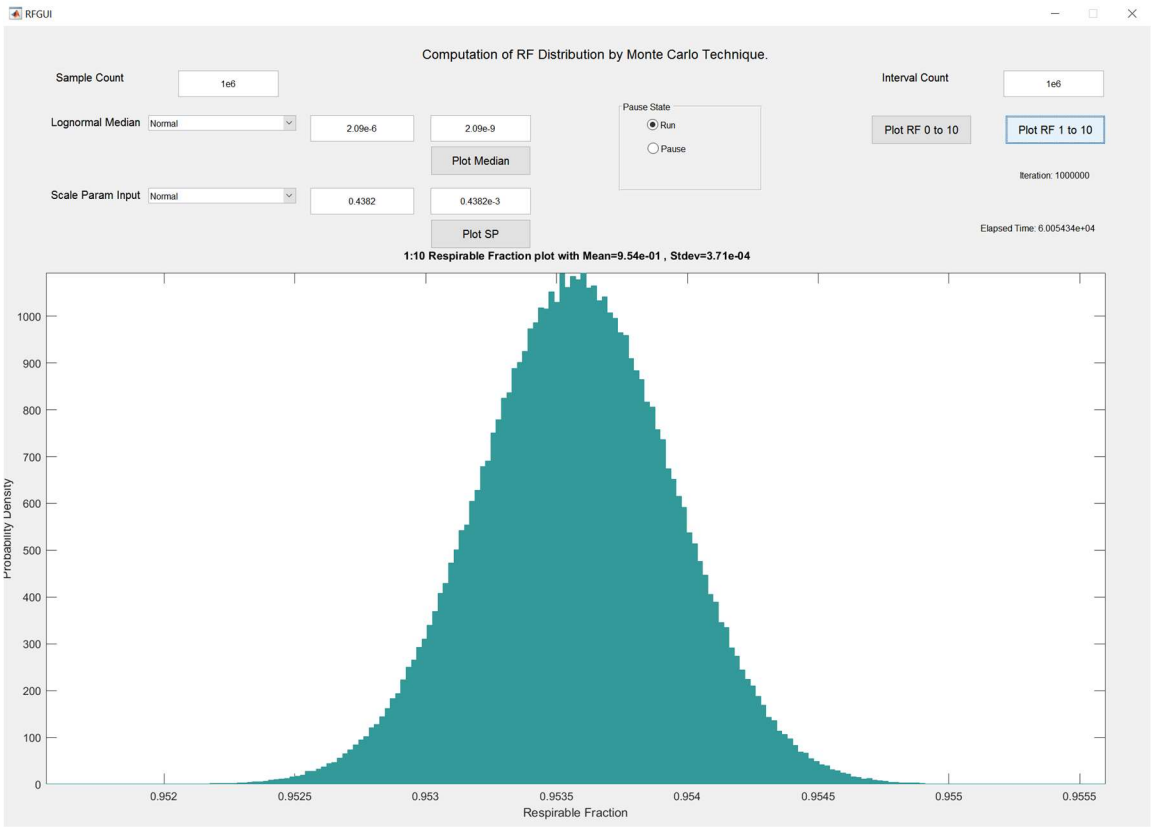


Figure 11. RF Limits from 1 to 10um, same uncertainties as in the reference case.

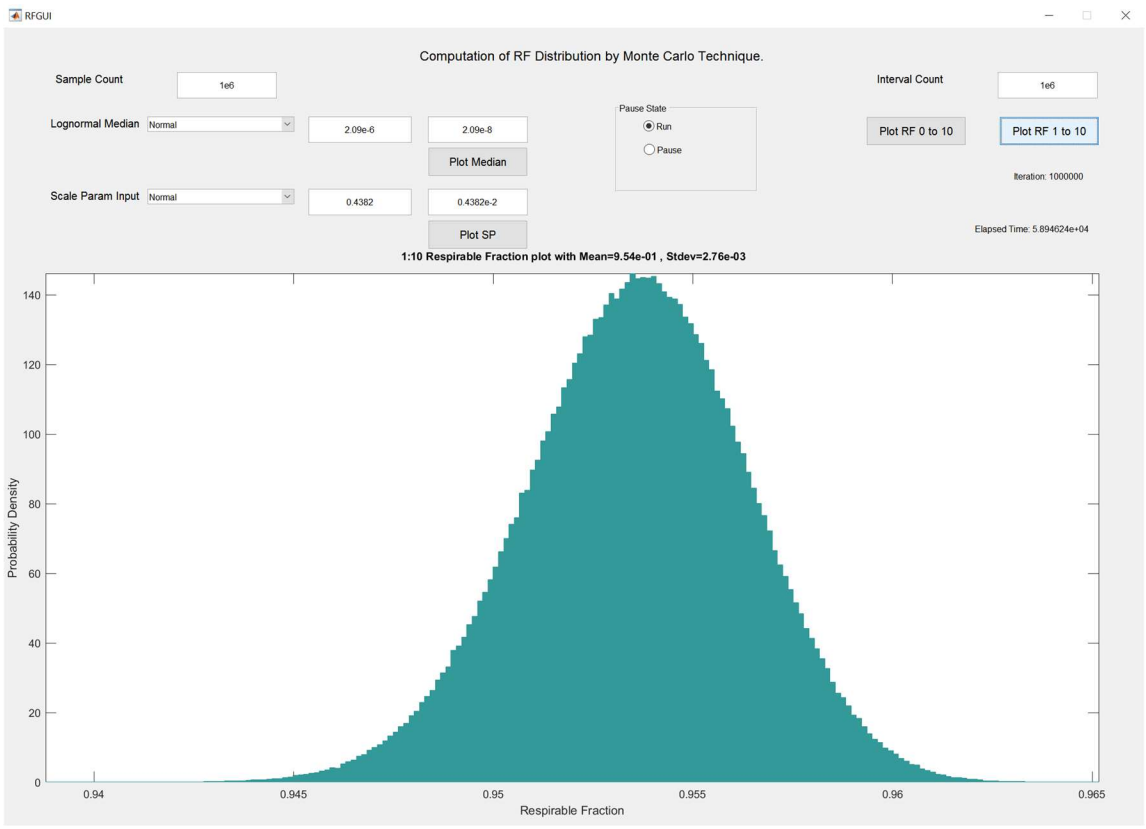


Figure 12. Same as previous figure, with uncertainties relaxed to 1% on both parameters.

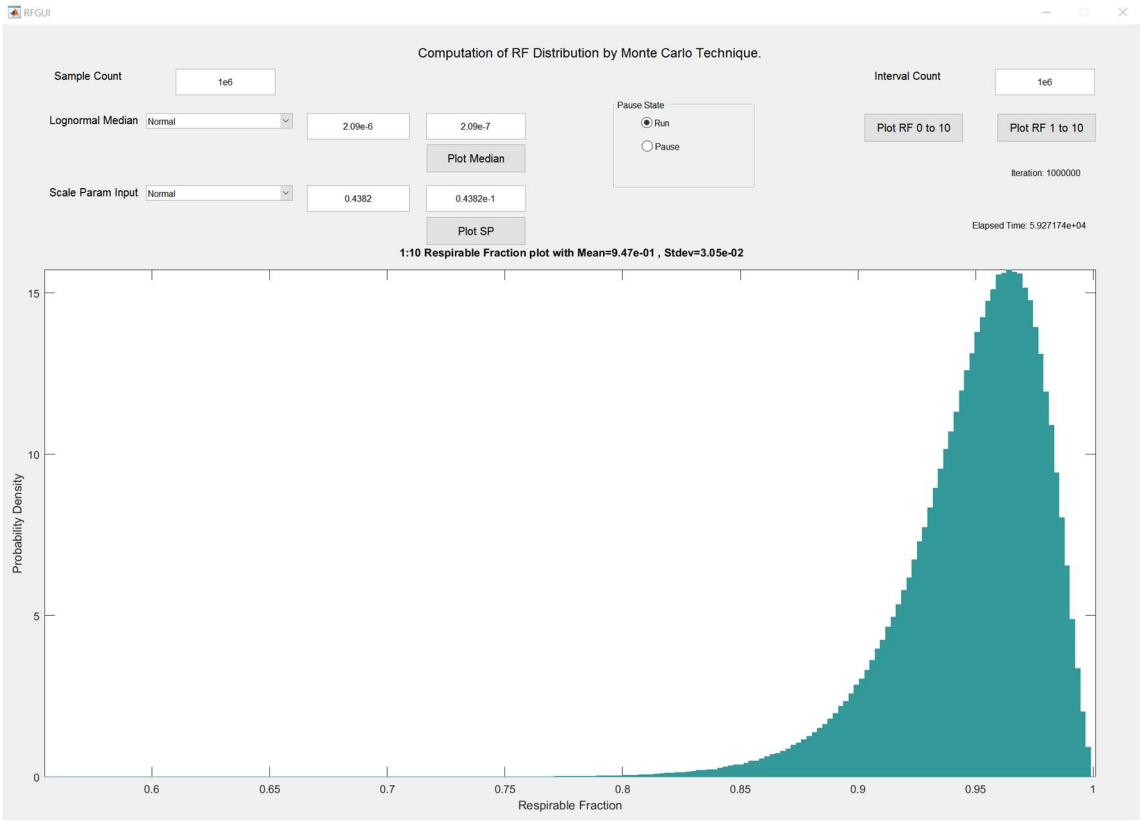


Figure 13. Uncertainties in both parameters relaxed to 10% as an extreme case.

The large uncertainty case shown in Figure 13, shows an interesting trend. When the uncertainties were small, the resulting RF distribution took the same shape as the PDFs for the curve fitting parameters. When the uncertainty is large, the resulting RF distribution looks like a log-normal distribution, but with the shape inverted in the X axis. After noticing the trend, the scale parameter and median uncertainties were varied independently to determine the effect on distribution shape, and sensitivity, to each parameter. The results of this are shown in Figures 14 and 15.

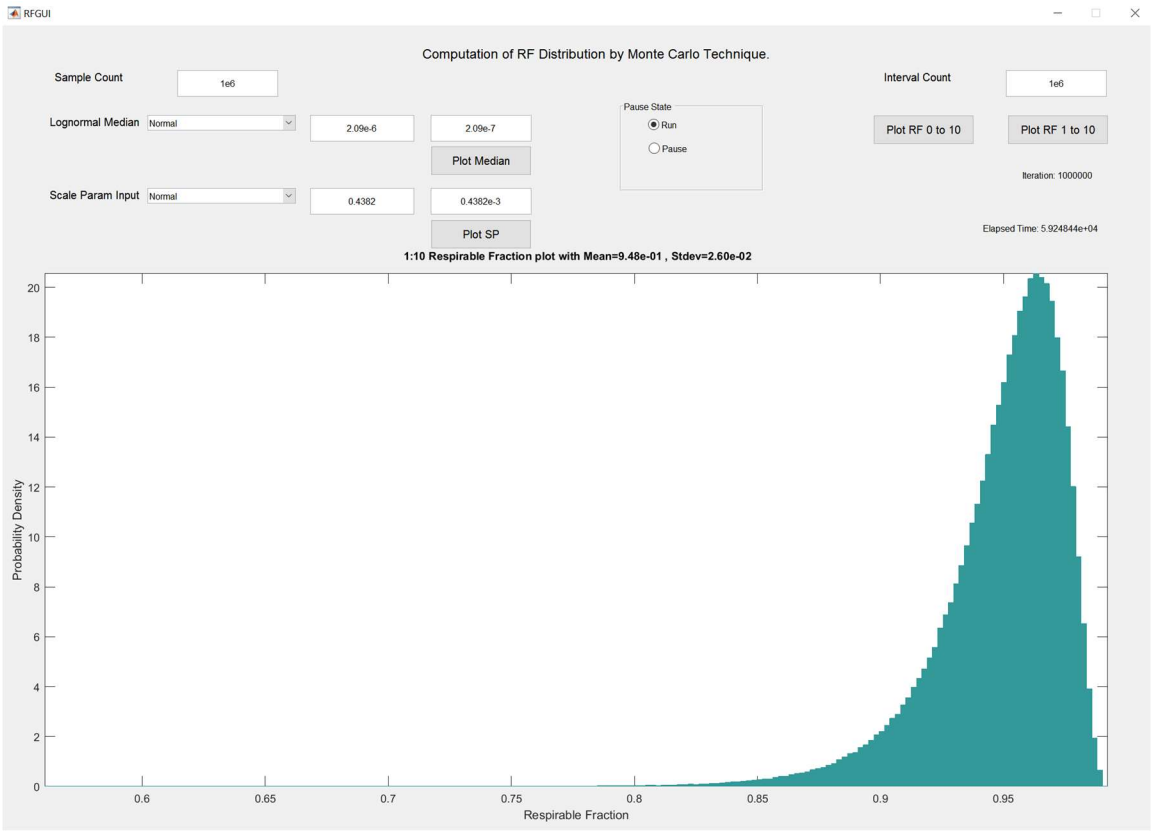


Figure 14. Parametric study of sensitivity of median and scale parameter to uncertainty with 10% uncertainty in median, and 0.1% uncertainty in scale parameter.

In this first case, the median uncertainty is placed at 10%, while the scale parameter is kept at 0.1% uncertainty. The distinctive shape, observed when both parameters were uncertain, remains. When compared to the case where both parameters have 10% uncertainty, the only difference is that the peak is more concentrated when the scale parameter has low uncertainty (lower standard deviation in the low scale parameter uncertainty case). After this, the opposite case was run, and the results are presented in Figure 15.

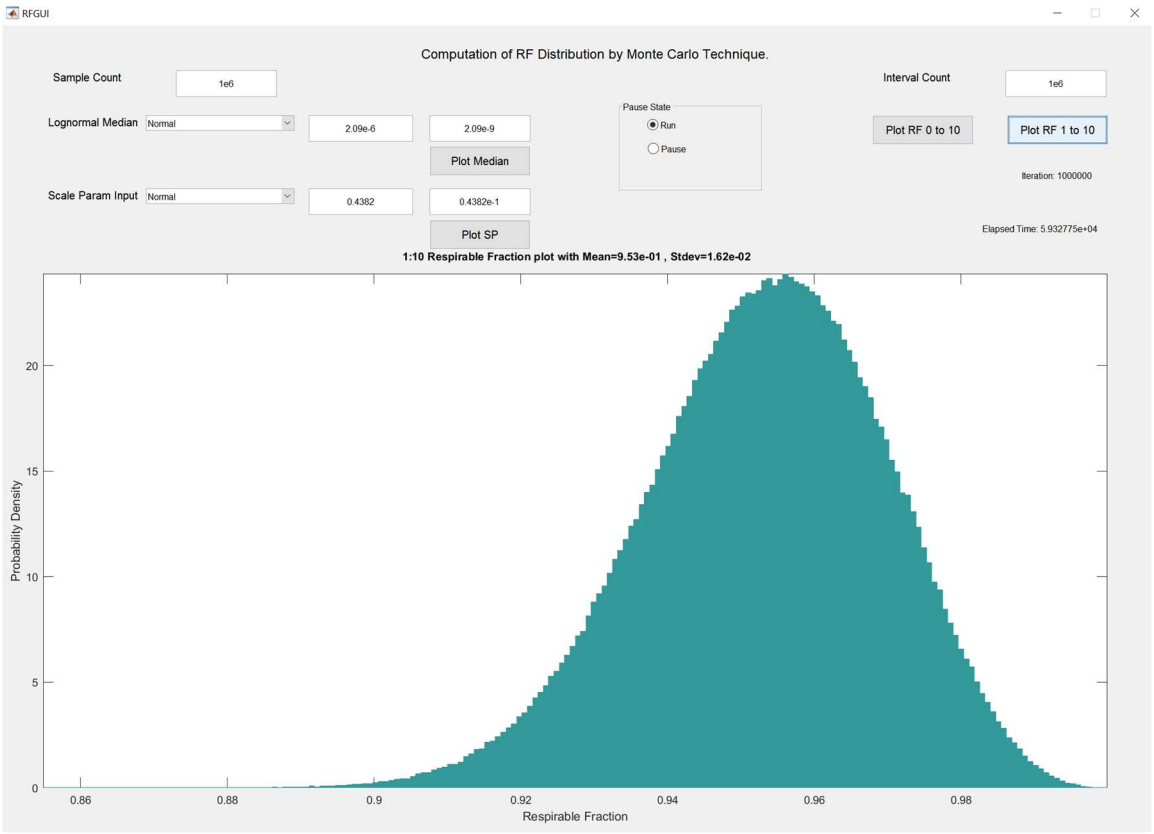


Figure 15. Parametric study of sensitivity of median and scale parameter to uncertainty with 0.1% uncertainty in median, and 10% uncertainty in scale parameter.

In this last case, with 10% uncertainty in the scale parameter, but only 0.l% in the median, the result is revealing. The case with 10% uncertainty in both parameters, as would be expected, gives the highest standard deviation in the final result. A 10% uncertainty in median, with low uncertainty in scale parameter results in the middling standard deviation, with the 10% uncertainty in scale parameter with low uncertainty in median yielding the smallest standard deviation. Furthermore, the reversed log-normal shape that was observed in the high uncertainty case is retained when the median has large uncertainty, but is only mildly present when the scale parameter has the high uncertainty. These results would suggest that median is the more sensitive parameter in this case.

9.4. Code/Algorithm Performance

The coding strategy that was employed allowed for rapid development, and the ability to obtain results utilizing the methodology that was detailed early in this chapter. In the first iteration of the program, the RF distribution was computed using an entirely single-threaded approach, which used nested for-loops to obtain samples. This method was very inefficient, typically placing only around 30% load on the CPU (The PC in question runs on an Intel Core i7 6700k @ 4.5Ghz, a quad core processor with hyperthreading, 4 physical cores, 8 virtual cores) when using 1E6 samples. There was an interdependence between CPU utilization and sample count, since the integrated MATLAB functions being used had some multithreading intrinsically. Thus, when lower sample counts were used, higher CPU loading was experienced. In this early implementation, computation of an RF distribution with 1E6 samples took 58 hours to complete. This amount of time was not ideal, so performance improvements were made to reduce this execution time. Since modern high performance CPUs incorporate multiple processor cores on a single die, it is important to design demanding software in a way that allows the code to execute on all available processor cores. This practice is called multithreading.

Using the MATLAB parfor construct, which allows a for-loop to be executed on multiple threads, or workers in MATLAB language, the run time was significantly reduced. Said simply, this change causes the for loop to be broken up into segments, each of which runs on a separate thread. These separate threads are each processed by a CPU core, allowing the entire CPU (all cores) to work on the problem. Using 7 workers on the same CPU as mentioned before resulted in 100% CPU loading, and a reduction in runtime down

to ~16.5 hours for 1E6 samples. This reduction made generating results far less time consuming.

In summary, for 1E6 samples the code is performing a one million sample calculation one million times. This is nessesary to get a sufficient sample count for the RF string, while minimizing Monte Carlo error in each RF calculation. Ultimately, the program in this mode must generate one trillion random numbers, in addition to performing two million numerical integrations. Thus, this methodology would not have been feasible to perform on a home PC even a decade ago.

10. Recommended Future Work

10.1. SODA

10.1.1. Code Optimizations

An area of potential future improvement of the SODA program is optimization of the algorithms employed in performing the Monte Carlo calculations. There are some areas which could be further vectorized, and potentially multi-threaded. Vectorization allows code to take advantage of advanced instruction sets in the CPU to perform computations more efficiently. MATLAB's integrated functions make good use of vectorization, while some SODA functions could be improved in this area. Notably, the user defined distribution sampling algorithm is very limited in performance by its liberal use of for-loop constructs. Another area which could be improved is in the multi-MAR calculations. For example, if four MARs need to have their CED distributions calculated, each of these could use a different thread. To implement something like this, however, the memory usage must be better understood and modelled. In some cases, not enough memory may be present to accommodate multiple CED calculations simultaneously. Features like this would be

highly beneficial in high computing power setups, although very high sample counts are, generally speaking, not needed for a tool like SODA. Considering that even the average PC these days has at least 4 cores, it would be beneficial to take advantage of multicore calculation schemes.

10.1.2. Machine Specific Sample Count Limits

One issue that SODA struggles with, is memory usage. When performing high sample count CED calculations with many distribution inputs, the memory usage can scale up to rather high values relative to the typical installed memory size in a home PC. (See Section on SODA Ram usage for additional details) The recommended future improvement to deal with this issue is to have the software interrogate the machine to discover how much available memory is present. Based upon this value, the software would set a limit on sample count based on the desired operation and available memory to ensure that the user never asks for a calculation which requires more memory than the machine has available. The relationship between sample count and peak memory usage is linear for a given number of MARs and distribution inputs. This would allow such a relationship to be easily identified and implemented, using a similar strategy to that which was employed in the parameter control portion of SODA.

10.1.3. Evaluation Guideline Specification

A feature which was considered in the past, and is still recommended for SODA, is the ability for the user to specify the evaluation guideline. It is not the purpose of SODA to give the user “the answer” to their question. If, instead, the user is able to program their desired metric to define a pass or fail, this allows SODA to give this type of feedback in a way that does not presume to know the needs of the user. For example, a user could specify

that the pass condition is a mean CED of less than 25 rem. In this case, SODA identifies a pass or fail based on whether or not the mean CED is less than 25 rem. Alternatively, the user could specify that the 95% CI value be less than 25 rem (Such as for the Maximally Exposed Hypothetical Individual, or MEHI, case). Regardless of the metric used, it would be useful for SODA to provide this type of immediate feedback.

10.1.4. Scripting

For a tool like SODA, which can be very effective in performing parametric study, it would be useful to add scripting capability. For example, if a user wished to run 50 calculations for CED studying variance in the distribution of DCF, it would be convenient to specify the step and range of values, then hit a button and have it all done automatically. To do this, the code would need some automation routines, as well as a save feature, which automatically saved the relevant values for each CED distribution and the corresponding plot. The scripting itself could be entered graphically, in keeping with the rest of the program, or by a text file similar to a batch file.

10.1.5. Development of Additional Parameter Distributions

The previous section of this thesis details a methodology by which the parameter distribution of respirable fraction can be quantified. By looking at the physics behind such a value, the distribution of this value can be identified, and applied to the calculations performed in a program like SODA. It is the recommendation of this author that additional work be performed in this area. The initial proof of concept used information on Pu => PuO_x oxidation aerosol product to get an RF distribution. Similar evaluations should be performed on other high importance materials so that their RF distributions can be

obtained. Eventually, these can be included into SODA, and used as the correct PDF for these stochastic calculations.

In addition to further development of RF distributions, it is important to develop distributions for other parameters which are similarly generated by examination of the raw data and physics governing the processes which affect the parameter in question. For example, an ARF distribution might consider data from materials which are made airborne by various physical and chemical processes, and develop a distribution based on the relative probability of these mechanisms being applied to the material at risk, and the uncertainties in each process measurement. A strategy for DCF is detailed in the following chapter.

As more parameters get their correct distributions identified, the ability for SODA to model true population dose distributions improves. This is the ultimate goal of a tool like SODA, and it is hoped that the presence of such a tool will encourage additional work in this area. Additional discussion on the future of SODA can be found in the concluding chapter.

10.2. RF Methodology

10.2.1. Computational Efficiency

The initial algorithm employed to compute the RF distribution was highly inefficient at using available CPU resources. The change to multithreading on the outermost loop did improve CPU utilization, but there would certainly be room to improve this further. Incorporation of multithreading or vectorization in the inner loops would considerably increase performance as well. It is also worth noting that, in the event that the data was available, directly sampling and integrating the interval-wise data that was taken

for the PuO_x particle distribution would have been more efficient, since the log-normal distributions would not need to be created and sampled for each iteration.

10.2.2. Further Exploration

With the development of a methodology to determine RF parameter distributions, further work in this area is advised. Using the simplified RF definition, use of other sets of mass distribution data or particle size distribution data could be used, and more RF distribution data could be uncovered. This would allow more generalizations to be made, which could help to advise SODA users as they specify their RF distributions as they perform CED calculations.

10.2.3. Advanced Model

In section 9.2.4.2 of this thesis, a model of RF was discussed which would yield true figures, rather than estimates, conservative or otherwise:

$$RF = \frac{\int_{LL}^{UL} P(D)M(D)dD}{\int_{0}^{\infty} M(D)dD}$$

Acquisition of the population average figures for the P(D) function, or the distribution of this function, would allow for proper quantification of RF, accounting for the deposition probability accurately. This would result in RF figures which were not based off of highly conservative assumptions, and further the goal of understanding population dose distributions using the SODA program.

11. Concluding Discussions

11.1. DCF Investigation and Abandonment

The first choice of the author for parametric study in SODA was the Dose Conversion Factor (DCF). The idea was to quantify the uncertainty in the DCF, and get an idea of how the DCF is distributed from this uncertainty. The inhalation DCF values, which are compiled in ICRP 119, are based on the respiratory tract model of ICRP 66. This is a very complex model in which intake processes, and biological processes, are considered. The purpose of this is to model how radionuclides propagate through the body, and consider how much dose will be imparted along this path. Biological half-lives give us an idea of how long it takes the body to expel these radionuclides. If, for example, the radionuclide (RN) in question is a calcium replacer, the RN will tend to attack the bones. From here, they are in a position to impart considerable dose to bone marrow, which is a tissue that is particularly sensitive to radiation [9].

Upon careful consideration of ICRP 66, and the models contained therein, it became quickly apparent that the investigation of DCF, and the quantification of the uncertainty, is a larger project than a Master's student would want to take on [9]. Consideration, even of the uncertainties involved in the measurements of the biology, would prove challenging. The course of action that should be taken, to really build a distribution for this parameter, would be the following:

First, a range of values must be determined with regards to the biological processes which are measured in order to model the systems represented by the ICRP inhalation tract model. Once this range is identified, the causes for the variance must be determined, so that when considering a population, an idea of how these values will be distributed in said

population can be established. If, hypothetically, 45% of people in a population are expected to be “highly vulnerable” to RN X in an inhalation scenario, 25% are expected to have “low vulnerability,” and the balance is “moderately vulnerable,” then a picture of how the rate of this given biological process varies in a population can be established. Such a distribution would need to be defined for each biological process that is involved in the transport of a given RN throughout the body. Once these parameters, and their population uncertainties are quantified, then these distributions can be propagated into the inhalation tract model.

From this point, the uncertainties in the model itself, must be identified. Simplifying assumptions must be considered, and how they may affect the uncertainty in population outliers. This step, it would seem, would be the easier of the two, though would still represent a good amount of work. Some groups have already worked on uncertainty quantification in the ICRP model, so some ground is already broken in that regard. Once both the biological uncertainties are quantified, and the model uncertainties are quantified, this total uncertainty can be propagated throughout the entire DCF calculation process, and a distribution of DCF values can be established for a representative population.

11.2. SODA Potential

The SODA program brings to the table a new set of capabilities. In the future, when parameter distributions can be identified for representative populations, it will become possible to model the dose distribution of a population. At the moment, using estimated uncertainties, a dose distribution can be approximated, already providing more information than is obtained from a single point calculation. All of this said, the real potential of SODA is the ability to identify actual population dose distributions for a modelled accident

scenario. As mentioned in the future work section, development of distributions for all modelled parameters is of great importance to realizing the full potential of this software.

11.3. RF Distribution Calculation

The methodology detailed in chapter 9 of this thesis is a new way to approach the determination of parameter distributions in support of modeling such as that used in SODA. By generalizing the methodology, correct distributions for RF can be computed for any aerosol that has been analyzed to determine its mass or particle size distributions. Eventually, these RF distributions can be tabulated in a manner similar to DCF value tabulation in SODA. Further development of actual distributions for these values, and their integration into SODA, pushes the goal of the SODA program to be able to model actual population dose distributions. From information in reference [7], it is clear there is, at a very detailed level of description, some interdependence between breathing rate and respirable fraction. Perhaps in the future a method will be developed to account for this interdependence. In any case, a method now exists to develop RF distributions, and a methodology is in place to develop distributions for other parameters.

11.4. Final Remarks

A great deal of progress was made on the SODA program over the course of the 2015-2016 FY of the project. Each objective for SODA improvement which was desired for this stage of the project was achieved, leaving the customer happy with the results. Effective strategies were employed in SODA development, which have been documented for the benefit of those who will work on similar projects in the future. The SODA application itself is now available for distribution through the webpage of Dr. Chad Pope. [10] This webpage contains the online and offline installers, as well as the help file. In addition to all

that was gained for the SODA program, development of a methodology to quantify parameter distributions for SODA was made. Thanks to modern computing power, this methodology opens many doors for future developments in distribution oriented dose calculations.

References

[1] DOE-STD-1027-92 HAZARD CATEGORIZATION AND ACCIDENT ANALYSIS TECHNIQUES FOR COMPLIANCE WITH DOE ORDER 5480.23, NUCLEAR SAFETY ANALYSIS REPORTS. December 1992. USDOE.

[2] Alireza Haghighat. Monte Carlo Methods for Particle Transport. CRC Press. 2015. Bocca Raton FL.

[3] Scott Chacon, Ben Straub. Pro Git, Second Ed. November 2014. Apress.

[4] Object Oriented Programming in MATLAB (and nested pages). Mathworks. Retrieved from: https://www.mathworks.com/discovery/object-oriented-programming.html

[5] Kushal Bhattari. SODA Application Design and Development. 2016. Idaho State University.

[6] Mary Tosten. Parametric Study of Plume Dispersion for Stochastic Objective Decision Aide. 2016. Idaho State University.

[7] Hazard Prevention and Control in the Work Environment: Airborne Dust. 1999. World Health Organization.

[8] DOE HDBK-3010-94 AIRBORNE RELEASE FRACTIONS/RATES AND RESPIRABLE FRACTIONS FOR NONREACTOR NUCLEAR FACILITIES. December 1994. USDOE.

[9] ICRP Publication 66: Human Respiratory Tract Model for Radiological Protection. Annals of the ICRP, Volume 24 Nos 1-3. 1994. International Commission on Radiological Protection.

[10] Stochastic Objective Decision Aide (SODA) Download Page. Idaho State University. Retrived from: http://www.isu.edu/ne/soda/

Appendix A: SODA Source Code

File 1, Project1.m: Main Code file for SODA.

```
function varargout = Project1(varargin)
% TO understand this code and follow the work it is essential that
% you open project figure file in Matlab GUIDE and get the tag name of
each object.
% Tag name is esssentially the functions in project.m file
% PROJECT1 MATLAB code for Project1.fig
%      PROJECT1, by itself, creates a new PROJECT1 or raises the
existing
%      singleton*.
%
%      H = PROJECT1 returns the handle to a new PROJECT1 or the handle
to
%      the existing singleton*.
%
%      PROJECT1('CALLBACK',hObject,~,handles,...) calls the local
%      function named CALLBACK in PROJECT1.M with the given input
arguments.
%
%      PROJECT1('Property','Value',...) creates a new PROJECT1 or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before Project1_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Project1_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Project1

% Last Modified by GUIDE v2.5 23-Jan-2016 13:55:46

% Begin initialization code - DO NOT EDIT
% This portion is generated by Matlab GUide. It is to make sure figure
file
% worl

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @Project1_OpeningFcn, ...
```

```
    'gui_OutputFcn',  @Project1_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
end

% --- Executes just before Project1 is made visible.
% This function is loaded in the begineing of the Soda first run.
% IN this function i set the default settings for all the push button
and
% turn on and off push button and toggle button.
function Project1_OpeningFcn(hObject, ~, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Project1 (see VARARGIN)

% Choose default command line output for Project1
handles.output = hObject;
axes(handles.logo_axes);
imshow('sodaLogo1.png');
% Update handles structure
guidata(hObject, handles);

%disable show plot button for all parameter when program starts
set(handles.fit_dist,'Enable','off')
set(handles.mar_pushbutton,'Enable','off');
set(handles.dr_pushbutton,'Enable','off');
set(handles.arf_pushbutton,'Enable','off');
set(handles.rf_pushbutton,'Enable','off');
set(handles.lpf_pushbutton,'Enable','off');
set(handles.br_pushbutton,'Enable','on');
set(handles.dcf_pushbutton,'Enable','off');
set(handles.cq_pushbutton,'Enable','off');
set(handles.runall_pushbutton, 'Enable','off');

%disable text1 all parameter when program starts
set(handles.mar_text1,'Enable','on');
set(handles.mar_popup_dist,'Enable','off');
```

```
set(handles.dr_text1,'Enable','off');
set(handles.arf_text1,'Enable','off');
set(handles.rf_text1,'Enable','off');
set(handles.lpf_text1,'Enable','off');
set(handles.br_text1,'Enable','off');
set(handles.dcf_text1,'Enable','off');
set(handles.cq_text1,'Enable','off');

%disable text2 for all parameter when program starts
set(handles.mar_text2,'Enable','off');
set(handles.dr_text2,'Enable','off');
set(handles.arf_text2,'Enable','off');
set(handles.rf_text2,'Enable','off');
set(handles.lpf_text2,'Enable','off');
set(handles.dcf_text2,'Enable','off');

%disable chi/q section distribution input and show plot button
set(handles.windspeed_text1,'Enable','off')
set(handles.windspeed_text2,'Enable','off')
set(handles.cq_pushbutton,'Enable','off')

set(handles.stability_popup,'Enable','off','String',{'Select
Stability'});
%set MAR toggle button pressed (to single input) when program starts

set(handles.mar_togglebutton,'Value',1);
set(handles.mar_togglebutton,'String','Single Input');
set(handles.dcf_togglebutton,'string','Single Input');
set(handles.dcf_togglebutton,'Value',1);
set(handles.dcf_popup_dist,'String',{'Select Isotope'}...
    ,'Value',1,'Enable','on');
set(handles.dcf_text1,'Enable','on');
set(handles.dcf_text1,'String','');
set(handles.dcf_text2,'String','');
set(handles.dcf_text2,'Enable','off') ; %
set(handles.dcf_pushbutton,'Enable','off');  %

global Parameters
global CurrentMAR
CurrentMAR = 1;
Parameters = SODA_Parameters();
end


% --- Outputs from this function are returned to the command line.
function varargout = Project1_OutputFcn(hObject, ~, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% ~  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
end


function num_sample_text_Callback(hObject, ~, handles)
% hObject    handle to num_sample_text (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end
% Hints: get(hObject,'String') returns contents of num_sample_text as
text
%        str2double(get(hObject,'String')) returns contents of
num_sample_text as a double


% --- Executes during object creation, after setting all properties.
function num_sample_text_CreateFcn(hObject, ~, handles)
% hObject    handle to num_sample_text (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on button press in rf_pushbutton.
%This function is exceuted when user presses RF show plot button
function rf_pushbutton_Callback(hObject, ~, handles)
% hObject    handle to rf_pushbutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%when user press RF Show plot button these command are exceuted.
% grab number from number of samples box.
% grab  number from text1 and text2 box.
% grab what kind of  distribution user has selected.
% if normal distribution is selected than generate random normal
% distribution with given paramter and plot histogram in is axes.
global Parameters;
global CurrentMAR;
```

```
sample = get(handles.num_sample_text,'String');%grab number from number
of sample box
samplesize = str2double(sample);
if strcmp(sample,'') == 1 || samplesize < 0
    errordlg('Please enter number of samples','Sample Number','modal');
    return;
end
col = get(handles.rf_pushbutton,'backg');
set(handles.rf_pushbutton,'str','RUNNING...','backg',[.2 .6 .6]);
pause(eps);
num1 = str2double(get(handles.rf_text1,'String')); %grab number from
number of text box 1
num2 = str2double(get(handles.rf_text2,'String')); %grab number from
number of text box 2
% Find what kind of  distribution user has selected.
contents = get(handles.rf_popup_dist,'String');
popupmenuvalue = contents{get(handles.rf_popup_dist,'Value')};
cla(handles.axes1,'reset');
switch popupmenuvalue
    case 'Normal'
        result = InputIsValid(handles.rf_text1, 'RF', ''); %check for
valid input, report to user
        result2 = InputIsValid(handles.rf_text2, 'RF', 'Sig'); %if
input is invalid.
        if result && result2
            pd = makedist('Normal','mu',num1,'sigma',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','rfxi', xi);
            assignin('base','rffi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('RF');
            str = sprintf('\\fontsize{12} RF distribution plot with
Normal distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
```

```
            if ~result && ~result2
                errordlg('Problem in rf_text1, rf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in rf_text1, invalid input.','Invalid
Input','modal');
            else
                errordlg('Problem in rf_text2, invalid input.','Invalid
Input','modal');
            end
        end
    case 'Log Normal'
        result = InputIsValid(handles.rf_text1, 'RF', '');
        result2 = InputIsValid(handles.rf_text2, 'RF', 'Sig');
        if result && result2
            pd =
makedist('Lognormal','mu',log(num1)+num2^2,'sigma',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','drxi', xi);
            assignin('base','drfi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            axis tight;
            ylabel('Probability Density');
            xlabel('RF');
            str = sprintf('\\fontsize{12} RF distribution plot with Log
Normal distribution with Mean=%0.2e , Stdev=%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in rf_text1, rf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in rf_text1, invalid input.','Invalid
Input','modal');
            else
```

```
                errordlg('Problem in rf_text2, invalid input.','Invalid
Input','modal');
            end
        end
    case 'Beta'
        result = InputIsValid(handles.rf_text1, 'RF', 'ab');
        result2 = InputIsValid(handles.rf_text2, 'RF', 'ab');
        if result && result2
            pd = makedist('Beta','a',num1,'b',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','rfxi', xi);
            assignin('base','rffi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('RF');
            str = sprintf('\\fontsize{12} RF distribution plot with
Beta distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in rf_text1, rf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in rf_text1, invalid input.','Invalid
Input','modal');
            else
                errordlg('Problem in rf_text2, invalid input.','Invalid
Input','modal');
            end
        end
    case 'Uniform'
        result = InputIsValid(handles.rf_text1, 'RF', '');
        result2 = InputIsValid(handles.rf_text2, 'RF', 'LL');
        if result && result2
            if num1 < num2;
```

```
                % In unifrom distribution upper limt must be greater
than lower
                % limit, if not show the error message
                errordlg('Upper Limit is less than lower limt','Uniform
Distribution','modal')
                set(handles.rf_pushbutton,'str','Show
Plot','backg',col);
                return;
            else
                pd = makedist('Uniform','Upper',num1,'Lower',num2);
                t = truncate(pd,0,1);
                n = random(t,samplesize,1);
                axes(handles.axes1)
                nbins = max(min(length(n)./10,100),50);
                xi = linspace(min(n),max(n),nbins);
                dx = mean(diff(xi));
                fi = histc(n,xi-dx);
                fi = fi./sum(fi)./dx;
                assignin('base','rfxi', xi);
                assignin('base','rffi2', fi);
                bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6
.6], 'BarWidth',1);
                axis tight;
                % %             hist(n,50);

                ylabel('Probability Density');
                xlabel('RF');
                str = sprintf('\\fontsize{12} RF distribution plot with
Uniform distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                    mean(n),std(n));
                title(str,'Units', 'normalized', ...
                    'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
            end
        else
            if ~result && ~result2
                errordlg('Problem in rf_text1, rf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in rf_text1, invalid input.','Invalid
Input','modal');
            else
                errordlg('Problem in rf_text2, invalid input.','Invalid
Input','modal');
            end
        end
    case 'Exponential'
        result = InputIsValid(handles.rf_text1, 'RF', '');
        if result
```

```
            pd = makedist('Exponential','mu',num1);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','rfxi', xi);
            assignin('base','rffi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            % %         hist(n,50);
            ylabel('Probability Density');
            xlabel('RF');
            str = sprintf('\\fontsize{12} RF distribution plot with
Exponential distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            errordlg('Problem in rf_text1, invalid input.','Invalid
Input','modal');
        end
    case 'User Defined'
        [Parameters,X,Y] = Parameters.GetUDD(CurrentMAR,'RF'); %Get UDD
data from object
        n = zeros(1,samplesize);
        for e = 1:samplesize;  %Sample the saved probability
distribution
            num_rand=rand;
            ter = size(X);
            for i = 1:ter(2)
                iSum = 0;
                for j = 1:i
                    iSum = iSum + Y(j);
                end
                if num_rand < iSum
                    if i == 1
                        n(e) = rand*(X(i+1)-X(i))+X(i);
                    else
                        n(e) = rand*(X(i)-X(i-1))+X(i);
                    end
                    break;
                end
            end
```

```
        end
        axes(handles.axes1)
        nbins = max(min(length(n)./10,100),50);
        xi = linspace(min(n),max(n),nbins);
        dx = mean(diff(xi));
        fi = histc(n,xi-dx);
        fi = fi./sum(fi)./dx;
        assignin('base','drxi', xi);
        assignin('base','drfi2', fi);
        bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
        axis tight;
        %         hist(n,50);
        ylabel('Probability Density');
        xlabel('RF');
        str = sprintf('\\fontsize{12} RF distribution plot with User
Defined Distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
            mean(n),std(n));
        title(str,'Units', 'normalized', ...
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')
end
set(handles.rf_pushbutton,'str','Show Plot','backg',col);
end



function rf_text2_Callback(hObject, ~, handles)
% hObject    handle to rf_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end
% Hints: get(hObject,'String') returns contents of rf_text2 as text
%        str2double(get(hObject,'String')) returns contents of rf_text2
as a double


% --- Executes during object creation, after setting all properties.
function rf_text2_CreateFcn(hObject, ~, handles)
% hObject    handle to rf_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end
end


function rf_text1_Callback(hObject, ~, handles)
% hObject    handle to rf_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end
% Hints: get(hObject,'String') returns contents of rf_text1 as text
%        str2double(get(hObject,'String')) returns contents of rf_text1
as a double


% --- Executes during object creation, after setting all properties.
function rf_text1_CreateFcn(hObject, ~, handles)
% hObject    handle to rf_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on selection change in rf_popup_dist.
%This function is executed when user selects from a drop down menu a
list
% of distribution and ask for respective parameter in the text box.
function rf_popup_dist_Callback(hObject, ~, handles)
% hObject    handle to rf_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% This function is executes when a Respirable fraction popup menu
distribution
% is choosen.

%find the string the user have selected.
global Parameters;
global CurrentMAR;
contents = cellstr(get(hObject,'String'));
rfpopchoice = contents{get(hObject,'Value')};

switch rfpopchoice
```

```
    case 'Normal'
        %if normal is selected enable input text box and also display
parameter
        %required in those text box.
        set(handles.rf_text1,'Enable','inactive')
        set(handles.rf_text2,'Enable','inactive')
        set(handles.rf_pushbutton,'Enable','on')
        set(handles.rf_text1,'String','Mean');
        set(handles.rf_text2,'String','Std Deviation');
        set(handles.rf_text1,'TooltipString','')
        set(handles.rf_text2,'TooltipString','')
    case 'Beta'
        %if Beta is selected enable input text box and also display
parameter
        %required in those text box.
        set(handles.rf_text1,'Enable','inactive')
        set(handles.rf_text2,'Enable','inactive')
        set(handles.rf_pushbutton,'Enable','on')
        set(handles.rf_text1,'String','a');
        set(handles.rf_text2,'String','b');
        set(handles.rf_text1,'TooltipString','shape parameter')
        set(handles.rf_text2,'TooltipString','shape parameter')
    case 'Uniform'
        %if Uniform is selected enable input text box and also display
parameter
        %required in those text box.
        set(handles.rf_text1,'Enable','inactive')
        set(handles.rf_text2,'Enable','inactive')
        set(handles.rf_pushbutton,'Enable','on')
        set(handles.rf_text1,'String','Upper Limit');
        set(handles.rf_text2,'String','Lower Limit');
        set(handles.rf_text1,'TooltipString','')
        set(handles.rf_text2,'TooltipString','')
    case 'Exponential'
        %if Exponential is selected enable input text box and also
display parameter
        %required in those text box.
        set(handles.rf_text1,'Enable','inactive')
        set(handles.rf_text2,'Enable','off')
        set(handles.rf_pushbutton,'Enable','on')
        set(handles.rf_text1,'String','Mean');
        set(handles.rf_text2,'String','');
        set(handles.rf_text1,'TooltipString','')
        set(handles.rf_text2,'TooltipString','')
    case 'Select Distribution'
        %if Select distribution is selected disable input text box and
also
        %disable show plot button.
        set(handles.rf_text1,'String','');
```

```
        set(handles.rf_text2,'String','');
        set(handles.rf_text1,'Enable','off')
        set(handles.rf_text2,'Enable','off')
        set(handles.rf_pushbutton,'Enable','off')
        set(handles.rf_text1,'TooltipString','')
        set(handles.rf_text2,'TooltipString','')
    case 'Log Normal'
        %if Log Normal is selected enable input text box and also
display parameter
        %required in those text box.
        set(handles.rf_text1,'Enable','inactive')  %
        set(handles.rf_text2,'Enable','inactive')  %
        set(handles.rf_pushbutton,'Enable','on')  %
        set(handles.rf_text1,'String',{'Mode'});
        set(handles.rf_text2,'String',{'Scale Param.'});
        set(handles.rf_text1,'TooltipString','')
        set(handles.rf_text2,'TooltipString','')
    case 'User Defined'
        %if Select distribution is selected disable input text box and
also
        %enable show plot button.
        set(handles.rf_text1,'Enable','off')
        set(handles.rf_text2,'Enable','off')
        set(handles.rf_pushbutton,'Enable','on')
        set(handles.rf_text1,'String','User');
        set(handles.rf_text2,'String','Defined');
        set(handles.rf_text1,'TooltipString','')
        set(handles.rf_text2,'TooltipString','')
        Parameters = UserDefined(Parameters);
        [Parameters, msg, flag] = Parameters.CheckUDD('RF'); %Check UDD
for correctness
        if flag == 1                                            %If
problem, do not save, and reset.
            msgbox(msg);
            set(Parameters,'UDtempX',0);
            set(Parameters,'UDtempY',0);
            set(hObject,'Value', 1);
            rf_popup_dist_Callback(hObject, '', handles);
        else
            Parameters = Parameters.SaveUDD(CurrentMAR,'RF');
        end
end
end
% Hints: contents = cellstr(get(hObject,'String')) returns
rf_popup_dist contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
rf_popup_dist
```

```
% --- Executes during object creation, after setting all properties.
function rf_popup_dist_CreateFcn(hObject, ~, handles)
% hObject    handle to rf_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on selection change in dr_popup_dist.
%This function is executed when user selects from a drop down menu a
list
% of distribution and ask for respective parameter in the text box.
function dr_popup_dist_Callback(hObject, ~, handles)
% hObject    handle to dr_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Parameters;
global CurrentMAR;
contents = cellstr(get(hObject,'String'));
drpopchoice = contents{get(hObject,'Value')};
switch drpopchoice
    case 'Normal'
        set(handles.dr_text1,'Enable','inactive')
        set(handles.dr_text2,'Enable','inactive')
        set(handles.dr_pushbutton,'Enable','on')
        set(handles.dr_text1,'String','Mean');
        set(handles.dr_text2,'String','Std Deviation');
        set(handles.dr_text1,'TooltipString','')
        set(handles.dr_text2,'TooltipString','')
    case 'Beta'
        set(handles.dr_text1,'Enable','inactive')
        set(handles.dr_text2,'Enable','inactive')
        set(handles.dr_pushbutton,'Enable','on')
        set(handles.dr_text1,'String','a');
        set(handles.dr_text2,'String','b');
        set(handles.dr_text1,'TooltipString','shape parameter')
        set(handles.dr_text2,'TooltipString','shape parameter')
    case 'Uniform'
        set(handles.dr_text1,'Enable','inactive')
        set(handles.dr_text2,'Enable','inactive')
        set(handles.dr_pushbutton,'Enable','on')
        set(handles.dr_text1,'String','Upper Limit');
```

```
            set(handles.dr_text2,'String','Lower Limit');
            set(handles.dr_text1,'TooltipString','')
            set(handles.dr_text2,'TooltipString','')
        case 'Exponential'
            set(handles.dr_text1,'Enable','inactive')
            set(handles.dr_text2,'Enable','off')
            set(handles.dr_pushbutton,'Enable','on')
            set(handles.dr_text1,'String','Mean');
            set(handles.dr_text2,'String','');
            set(handles.dr_text1,'TooltipString','')
            set(handles.dr_text2,'TooltipString','')
        case 'Select Distribution'
            set(handles.dr_text1,'String','');
            set(handles.dr_text2,'String','');
            set(handles.dr_text1,'Enable','off')
            set(handles.dr_text2,'Enable','off')
            set(handles.dr_pushbutton,'Enable','off')
            set(handles.dr_text1,'TooltipString','')
            set(handles.dr_text2,'TooltipString','')
        case 'Log Normal'
            set(handles.dr_text1,'Enable','inactive')
            set(handles.dr_text2,'Enable','inactive')
            set(handles.dr_pushbutton,'Enable','on')
            set(handles.dr_text1,'String',{'Mode'});
            set(handles.dr_text2,'String',{'Scale Param.'});
            set(handles.dr_text1,'TooltipString','')
            set(handles.dr_text2,'TooltipString','')
        case 'User Defined'
            set(handles.dr_text1,'Enable','off')
            set(handles.dr_text2,'Enable','off')
            set(handles.dr_pushbutton,'Enable','on')
            set(handles.dr_text1,'String','User');
            set(handles.dr_text2,'String','Defined');
            set(handles.dr_text1,'TooltipString','')
            set(handles.dr_text2,'TooltipString','')
            Parameters = UserDefined(Parameters);
            [Parameters, msg, flag] = Parameters.CheckUDD('DR');
            if flag == 1
                msgbox(msg);
                set(Parameters,'UDtempX',0);
                set(Parameters,'UDtempY',0);
                set(hObject,'Value', 1);
                dr_popup_dist_Callback(hObject, '', handles);
            else
                Parameters = Parameters.SaveUDD(CurrentMAR,'DR');
            end
    end
end
```

```
% Hints: contents = cellstr(get(hObject,'String')) returns
dr_popup_dist contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
dr_popup_dist


% --- Executes during object creation, after setting all properties.
function dr_popup_dist_CreateFcn(hObject, ~, handles)
% hObject    handle to dr_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function dr_text1_Callback(hObject, ~, handles)
% hObject    handle to dr_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of dr_text1 as text
%        str2double(get(hObject,'String')) returns contents of dr_text1
as a double

end
% --- Executes during object creation, after setting all properties.
function dr_text1_CreateFcn(hObject, ~, handles)
% hObject    handle to dr_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function dr_text2_Callback(hObject, ~, handles)
```

```
% hObject    handle to dr_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of dr_text2 as text
%        str2double(get(hObject,'String')) returns contents of dr_text2
as a double

end
% --- Executes during object creation, after setting all properties.
function dr_text2_CreateFcn(hObject, ~, handles)
% hObject    handle to dr_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end



% --- Executes on button press in dr_pushbutton.
%This function is executed when user press DR show plot button
function dr_pushbutton_Callback(hObject, ~, handles)
% hObject    handle to dr_pushbutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Parameters;
global CurrentMAR;
sample = get(handles.num_sample_text,'String');
samplesize = str2double(sample);
if strcmp(sample,'') == 1 || samplesize < 0
    errordlg('Please enter number of samples','Sample Number','modal');
    return;
end
col = get(handles.dr_pushbutton,'backg');
set(handles.dr_pushbutton,'str','RUNNING...','backg',[.2 .6 .6]);
pause(eps);
num1 = str2double(get(handles.dr_text1,'String'));
num2 = str2double(get(handles.dr_text2,'String'));
contents = get(handles.dr_popup_dist,'String');
popupmenuvalue = contents{get(handles.dr_popup_dist,'Value')};
cla(handles.axes1,'reset');
switch popupmenuvalue
    case 'Normal'
```

```
        result = InputIsValid(handles.dr_text1, 'DR', '');
        result2 = InputIsValid(handles.dr_text2, 'DR', 'Sig');
        if result && result2
            pd = makedist('Normal','mu',num1,'sigma',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','drxi', xi);
            assignin('base','drfi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            axis tight;
            ylabel('Probability Density');
            xlabel('DR');
            str = sprintf('\\fontsize{12} DR distribution plot with
Normal distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in dr_text1, dr_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in dr_text1, invalid input.','Invalid
Input','modal');
            else
                errordlg('Problem in dr_text2, invalid input.','Invalid
Input','modal');
            end
        end
    case 'Log Normal'
        result = InputIsValid(handles.dr_text1, 'DR', '');
        result2 = InputIsValid(handles.dr_text2, 'DR', 'Sig');
        if result && result2
            pd =
makedist('Lognormal','mu',log(num1)+num2^2,'sigma',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
```

```
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','drxi', xi);
            assignin('base','drfi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            axis tight;
            ylabel('Probability Density');
            xlabel('DR');
            str = sprintf('\\fontsize{12} DR distribution plot with Log
Normal distribution with Mean=%0.2e , Stdev=%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in dr_text1, dr_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in dr_text1, invalid input.','Invalid
Input','modal');
            else
                errordlg('Problem in dr_text2, invalid input.','Invalid
Input','modal');
            end
        end

    case 'Beta'
        result = InputIsValid(handles.dr_text1, 'DR', 'ab');
        result2 = InputIsValid(handles.dr_text2, 'DR', 'ab');
        if result && result2
            pd = makedist('Beta','a',num1,'b',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','drxi', xi);
            assignin('base','drfi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
```

```
            axis tight;
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('DR');
            str = sprintf('\\fontsize{12} DR distribution plot with
Beta distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in dr_text1, dr_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in dr_text1, invalid input.','Invalid
Input','modal');
            else
                errordlg('Problem in dr_text2, invalid input.','Invalid
Input','modal');
            end
        end
    case 'Uniform'
        result = InputIsValid(handles.dr_text1, 'DR', '');
        result2 = InputIsValid(handles.dr_text2, 'DR', 'LL');
        if result && result2
            if num1 < num2;
                % In unifrom distribution upper limt must be greater
than lower
                % limit, if not show the error message
                errordlg('Upper Limit is less than lower limt','Uniform
Distribution','modal')
                set(handles.dr_pushbutton,'str','Show
Plot','backg',col);
                return;
            else
                pd = makedist('Uniform','Upper',num1,'Lower',num2);
                t = truncate(pd,0,1);
                n = random(t,samplesize,1);
                axes(handles.axes1)
                nbins = max(min(length(n)./10,100),50);
                xi = linspace(min(n),max(n),nbins);
                dx = mean(diff(xi));
                fi = histc(n,xi-dx);
                fi = fi./sum(fi)./dx;
                assignin('base','drxi', xi);
                assignin('base','drfi2', fi);
                bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6
.6], 'BarWidth',1);
```

```
                axis tight;
                %             hist(n,50);

                ylabel('Probability Density');
                xlabel('DR');
                str = sprintf('\\fontsize{12} DR distribution plot with
Uniform distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                    mean(n),std(n));
                title(str,'Units', 'normalized', ...
                    'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
            end
        else
            if ~result && ~result2
                errordlg('Problem in dr_text1, dr_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in dr_text1, invalid input.','Invalid
Input','modal');
            else
                errordlg('Problem in dr_text2, invalid input.','Invalid
Input','modal');
            end
        end
    case 'Exponential'
        result = InputIsValid(handles.dr_text1, 'DR', '');
        if result
            pd = makedist('Exponential','mu',num1);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','drxi', xi);
            assignin('base','drfi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('DR');
            str = sprintf('\\fontsize{12} DR distribution plot with
Exponential distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
```

```
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            errordlg('Problem in dr_text1, invalid input.','Invalid
Input','modal');
        end
    case 'User Defined'
        [Parameters,X,Y] = Parameters.GetUDD(CurrentMAR,'DR');
        n = zeros(1,samplesize);
        for e = 1:samplesize;
            num_rand=rand;
            ter = size(X);
            for i = 1:ter(2)
                iSum = 0;
                for j = 1:i
                    iSum = iSum + Y(j);
                end
                if num_rand < iSum
                    if i == 1
                        n(e) = rand*(X(i+1)-X(i))+X(i);
                    else
                        n(e) = rand*(X(i)-X(i-1))+X(i);
                    end
                    break;
                end
            end
        end
        axes(handles.axes1)
        nbins = max(min(length(n)./10,100),50);
        xi = linspace(min(n),max(n),nbins);
        dx = mean(diff(xi));
        fi = histc(n,xi-dx);
        fi = fi./sum(fi)./dx;
        assignin('base','drxi', xi);
        assignin('base','drfi2', fi);
        bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
        axis tight;
        %         hist(n,50);
        ylabel('Probability Density');
        xlabel('DR');
        str = sprintf('\\fontsize{12} DR distribution plot with User
Defined Distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
            mean(n),std(n));
        title(str,'Units', 'normalized', ...
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')
end
set(handles.dr_pushbutton,'str','Show Plot','backg',col);
end
```

```
% --- Executes on button press in run_pushbutton.
%This function is executed when user press Run
% THis is the main function where the CED is computed
function run_pushbutton_Callback(hObject, ~, handles)
% hObject    handle to run_pushbutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
tic
global Parameters

SaveMARSpecificData(handles);  % Save current inputs on the selected
mar to the class object.
if CheckSamples(handles) %Check for sample count, report if problem
    col = get(handles.run_pushbutton,'backg');
    %disable show plot button for all parameter when running ced plot
    set(handles.fit_dist,'Enable','off')
    if strcmp(get(handles.mar_pushbutton,'Enable'),'on')  %Disable
buttons that are active, and save their original state.
        set(handles.mar_pushbutton,'Enable','off');
        marbutton = 1;
    else
        marbutton = 0;
    end
    if strcmp(get(handles.dr_pushbutton,'Enable'),'on')
        set(handles.dr_pushbutton,'Enable','off');
        drbutton = 1;
    else
        drbutton = 0;
    end
    if strcmp(get(handles.arf_pushbutton,'Enable'),'on')
        set(handles.arf_pushbutton,'Enable','off');
        arfbutton = 1;
    else
        arfbutton = 0;
    end
    if strcmp(get(handles.rf_pushbutton,'Enable'),'on')
        set(handles.rf_pushbutton,'Enable','off');
        rfbutton = 1;
    else
        rfbutton = 0;
    end
    if strcmp(get(handles.lpf_pushbutton,'Enable'),'on')
        set(handles.lpf_pushbutton,'Enable','off');
        lpfbutton = 1;
    else
        lpfbutton = 0;
    end
    if strcmp(get(handles.br_pushbutton,'Enable'),'on')
```

```
    set(handles.br_pushbutton,'Enable','off');
    brbutton = 1;
else
    brbutton = 0;
end
if strcmp(get(handles.dcf_pushbutton,'Enable'),'on')
    set(handles.dcf_pushbutton,'Enable','off');
    dcfbutton = 1;
else
    dcfbutton = 0;
end
if strcmp(get(handles.cq_pushbutton,'Enable'),'on')
    set(handles.cq_pushbutton,'Enable','off');
    cqbutton = 1;
else
    cqbutton = 0;
end

%disable text1 all parameter
if strcmp(get(handles.mar_text1,'Enable'),'on')
    set(handles.mar_text1,'Enable','off');
    martext1 = 1;
else
    martext1 = 0;
end
if strcmp(get(handles.dr_text1,'Enable'),'on')
    set(handles.dr_text1,'Enable','off');
    drtext1 = 1;
else
    drtext1 = 0;
end
if strcmp(get(handles.arf_text1,'Enable'),'on')
    set(handles.arf_text1,'Enable','off');
    arftext1 = 1;
else
    arftext1 = 0;
end
if strcmp(get(handles.rf_text1,'Enable'),'on')
    set(handles.rf_text1,'Enable','off');
    rftext1 = 1;
else
    rftext1 = 0;
end
if strcmp(get(handles.lpf_text1,'Enable'),'on')
    set(handles.lpf_text1,'Enable','off');
    lpftext1 = 1;
else
    lpftext1 = 0;
end
```

```
if strcmp(get(handles.dcf_text1,'Enable'),'on')
    set(handles.dcf_text1,'Enable','off');
    dcftext1 = 1;
else
    dcftext1 = 0;
end
if strcmp(get(handles.cq_text1,'Enable'),'on')
    set(handles.cq_text1,'Enable','off');
    cqtext1 = 1;
else
    cqtext1 = 0;
end

%disable text2 for all parameter
if strcmp(get(handles.mar_text2,'Enable'),'on')
    set(handles.mar_text2,'Enable','off');
    martext2 = 1;
else
    martext2 = 0;
end
if strcmp(get(handles.dr_text2,'Enable'),'on')
    set(handles.dr_text2,'Enable','off');
    drtext2 = 1;
else
    drtext2 = 0;
end
if strcmp(get(handles.arf_text2,'Enable'),'on')
    set(handles.arf_text2,'Enable','off');
    arftext2 = 1;
else
    arftext2 = 0;
end
if strcmp(get(handles.rf_text2,'Enable'),'on')
    set(handles.rf_text2,'Enable','off');
    rftext2 = 1;
else
    rftext2 = 0;
end
if strcmp(get(handles.lpf_text2,'Enable'),'on')
    set(handles.lpf_text2,'Enable','off');
    lpftext2 = 1;
else
    lpftext2 = 0;
end
if strcmp(get(handles.dcf_text2,'Enable'),'on')
    set(handles.dcf_text2,'Enable','off');
    dcftext2 = 1;
else
    dcftext2 = 0;
```

```
    end


    %Disable radio buttons before run
    set(handles.radioMAR1,'Enable','off')
    set(handles.radioMAR2,'Enable','off')
    set(handles.radioMAR3,'Enable','off')
    set(handles.radioMAR4,'Enable','off')

    Diditwork = 1;
    cla(handles.axes1,'reset');
    if MARxisValid(handles)  %Try catch, in case of error due to
invalid input selection.
        try
            GetResults(handles); %Run to get results
        catch
            msgbox('Failed to Create Plot, incomplete or invalid
inputs.');
            Diditwork = 0;
        end
        if strcmp(get(handles.num_sample_text, 'str'),'') == 1
            Diditwork = 0;
        end
    else
        Diditwork = 0;
        errordlg('Invalid input detected. Calculation not
performed.','Invalid Input','modal');
    end
    if Diditwork
        I = GetCurrentMAR();
        switch I        %Plot the correct result, then release the
memory.
            case 1

bar(handles.axes1,Parameters.XResult1,Parameters.YResult1,'FaceColor',[
.2 .6 .6],'EdgeColor',[.2 .6 .6], 'BarWidth',1);

bar(handles.axes1,Parameters.XResult1,Parameters.YResult1,'FaceColor','
m','EdgeColor','m','BarWidth', 1);
                Parameters.CED1 = 0;
            case 2

bar(handles.axes1,Parameters.XResult2,Parameters.YResult2,'FaceColor',[
.2 .6 .6],'EdgeColor',[.2 .6 .6], 'BarWidth',1);

bar(handles.axes1,Parameters.XResult2,Parameters.YResult2,'FaceColor','
m','EdgeColor','m','BarWidth', 1);
                Parameters.CED2 = 0;
            case 3
```

```
bar(handles.axes1,Parameters.XResult3,Parameters.YResult3,'FaceColor',[
.2 .6 .6],'EdgeColor',[.2 .6 .6], 'BarWidth',1);

bar(handles.axes1,Parameters.XResult3,Parameters.YResult3,'FaceColor','
m','EdgeColor','m','BarWidth', 1);
                Parameters.CED3 = 0;
            case 4

bar(handles.axes1,Parameters.XResult4,Parameters.YResult4,'FaceColor',[
.2 .6 .6],'EdgeColor',[.2 .6 .6], 'BarWidth',1);

bar(handles.axes1,Parameters.XResult4,Parameters.YResult4,'FaceColor','
m','EdgeColor','m','BarWidth', 1);
                Parameters.CED4 = 0;
            case 0
                msgbox('MAR State Exclusivity Error; SODA will
close.','Fatal Error')
                delete(handles.Soda_Main);
        end
        str = sprintf('\\fontsize{11}CED, Mean = %0.3e rem, Median=
%0.3e rem, 95 Percentile = %0.3e
rem',Parameters.AvgCED(I),Parameters.MedCED(I),Parameters.Ninty_fifth(I
));
        title(handles.axes1,str,'Units', 'normalized', ...
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')
        xlabel(handles.axes1,'Commited Effective Dose (rem)')
        ylabel(handles.axes1,'Probability Density')
        legend(handles.axes1,'Random Generated','Location','NE')
        axis tight;
        grid on;
        set(handles.fit_dist,'Enable','on');
    end

    set(handles.run_pushbutton,'str','Show Plot','backg',col);
    %Enable show plot button for all parameter when running ced plot

    if marbutton == 1       %Reenable those buttons which were active
before pressing show plot.
        set(handles.mar_pushbutton,'Enable','on');
    end
    if drbutton == 1
        set(handles.dr_pushbutton,'Enable','on');
    end
    if arfbutton == 1
        set(handles.arf_pushbutton,'Enable','on');
    end
    if rfbutton == 1
        set(handles.rf_pushbutton,'Enable','on');
```

```
end
if lpfbutton == 1
    set(handles.lpf_pushbutton,'Enable','on');
end
if dcfbutton == 1
    set(handles.dcf_pushbutton,'Enable','on');
end
if cqbutton == 1
    set(handles.cq_pushbutton,'Enable','on');
end

%Enable text1 all parameter
if martext1 == 1
    set(handles.mar_text1,'Enable','on');
end
if drtext1 == 1
    set(handles.dr_text1,'Enable','on');
end
if arftext1 == 1
    set(handles.arf_text1,'Enable','on');
end
if rftext1 == 1
    set(handles.rf_text1,'Enable','on');
end
if lpftext1 == 1
    set(handles.lpf_text1,'Enable','on');
end
if dcftext1 == 1
    set(handles.dcf_text1,'Enable','on');
end
if cqtext1 == 1
    set(handles.cq_text1,'Enable','on');
end

%Enable text2 for all parameter
if martext2 == 1
    set(handles.mar_text2,'Enable','on');
end
if drtext2 == 1
    set(handles.dr_text2,'Enable','on');
end
if arftext2 == 1
    set(handles.arf_text2,'Enable','on');
end
if rftext2 == 1
    set(handles.rf_text2,'Enable','on');
end
if lpftext2 == 1
    set(handles.lpf_text2,'Enable','on');
```

```
    end
    if dcftext2 == 1
        set(handles.dcf_text2,'Enable','on');
    end

    %Enable radio buttons after run
    set(handles.radioMAR1,'Enable','on')
    set(handles.radioMAR2,'Enable','on')
    set(handles.radioMAR3,'Enable','on')
    set(handles.radioMAR4,'Enable','on')
else
    return
end
toc;
end

% --- Executes on button press in runall_pushbutton.
function runall_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to runall_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global Parameters
global CurrentMAR

result = SaveMARSpecificData(handles);  %Similar to the show plot
routine.
if CheckSamples(handles)
    col = get(handles.run_pushbutton,'backg');
    if result == 0;
        if strcmp(get(handles.num_sample_text, 'str'),'') == 0
            %disable show plot button for all parameter when running
ced plot
            set(handles.fit_dist,'Enable','off')
            if strcmp(get(handles.mar_pushbutton,'Enable'),'on')
                set(handles.mar_pushbutton,'Enable','off');
                marbutton = 1;
            else
                marbutton = 0;
            end
            if strcmp(get(handles.dr_pushbutton,'Enable'),'on')
                set(handles.dr_pushbutton,'Enable','off');
                drbutton = 1;
            else
                drbutton = 0;
            end
            if strcmp(get(handles.arf_pushbutton,'Enable'),'on')
                set(handles.arf_pushbutton,'Enable','off');
                arfbutton = 1;
```

```
else
    arfbutton = 0;
end
if strcmp(get(handles.rf_pushbutton,'Enable'),'on')
    set(handles.rf_pushbutton,'Enable','off');
    rfbutton = 1;
else
    rfbutton = 0;
end
if strcmp(get(handles.lpf_pushbutton,'Enable'),'on')
    set(handles.lpf_pushbutton,'Enable','off');
    lpfbutton = 1;
else
    lpfbutton = 0;
end
if strcmp(get(handles.br_pushbutton,'Enable'),'on')
    set(handles.br_pushbutton,'Enable','off');
    brbutton = 1;
else
    brbutton = 0;
end
if strcmp(get(handles.dcf_pushbutton,'Enable'),'on')
    set(handles.dcf_pushbutton,'Enable','off');
    dcfbutton = 1;
else
    dcfbutton = 0;
end
if strcmp(get(handles.cq_pushbutton,'Enable'),'on')
    set(handles.cq_pushbutton,'Enable','off');
    cqbutton = 1;
else
    cqbutton = 0;
end

%disable text1 all parameter
if strcmp(get(handles.mar_text1,'Enable'),'on')
    set(handles.mar_text1,'Enable','off');
    martext1 = 1;
else
    martext1 = 0;
end
if strcmp(get(handles.dr_text1,'Enable'),'on')
    set(handles.dr_text1,'Enable','off');
    drtext1 = 1;
else
    drtext1 = 0;
end
if strcmp(get(handles.arf_text1,'Enable'),'on')
    set(handles.arf_text1,'Enable','off');
```

```
    arftext1 = 1;
else
    arftext1 = 0;
end
if strcmp(get(handles.rf_text1,'Enable'),'on')
    set(handles.rf_text1,'Enable','off');
    rftext1 = 1;
else
    rftext1 = 0;
end
if strcmp(get(handles.lpf_text1,'Enable'),'on')
    set(handles.lpf_text1,'Enable','off');
    lpftext1 = 1;
else
    lpftext1 = 0;
end
if strcmp(get(handles.dcf_text1,'Enable'),'on')
    set(handles.dcf_text1,'Enable','off');
    dcftext1 = 1;
else
    dcftext1 = 0;
end
if strcmp(get(handles.cq_text1,'Enable'),'on')
    set(handles.cq_text1,'Enable','off');
    cqtext1 = 1;
else
    cqtext1 = 0;
end

%disable text2 for all parameter
if strcmp(get(handles.mar_text2,'Enable'),'on')
    set(handles.mar_text2,'Enable','off');
    martext2 = 1;
else
    martext2 = 0;
end
if strcmp(get(handles.dr_text2,'Enable'),'on')
    set(handles.dr_text2,'Enable','off');
    drtext2 = 1;
else
    drtext2 = 0;
end
if strcmp(get(handles.arf_text2,'Enable'),'on')
    set(handles.arf_text2,'Enable','off');
    arftext2 = 1;
else
    arftext2 = 0;
end
if strcmp(get(handles.rf_text2,'Enable'),'on')
```

```
            set(handles.rf_text2,'Enable','off');
            rftext2 = 1;
        else
            rftext2 = 0;
        end
        if strcmp(get(handles.lpf_text2,'Enable'),'on')
            set(handles.lpf_text2,'Enable','off');
            lpftext2 = 1;
        else
            lpftext2 = 0;
        end
        if strcmp(get(handles.dcf_text2,'Enable'),'on')
            set(handles.dcf_text2,'Enable','off');
            dcftext2 = 1;
        else
            dcftext2 = 0;
        end


        %Disable radio buttons after run
        set(handles.radioMAR1,'Enable','off')
        set(handles.radioMAR2,'Enable','off')
        set(handles.radioMAR3,'Enable','off')
        set(handles.radioMAR4,'Enable','off')

        %Use GetResults on all complete entries.
        OriginState = CurrentMAR;
        WasSuccessful = [1,1,1,1];
        ChangeMARState(handles.radioMAR1, handles);
        cla(handles.axes1,'reset');
        if MARxisValid(handles)          %Check each MAR for valid
result, leave out invalid results and report to user.
            try
                GetResults(handles); %Run to get results
            catch
                if Parameters.MAR(1) ~= 0 && Parameters.DCF(1) ~= 0
                    waitfor(errordlg('Problem in MAR1 Result.
Skipping. Check your input for invalid or missing entries.'))
                end
                WasSuccessful(1) = 0;
                Parameters.CED1 =
zeros(str2double(get(handles.num_sample_text,'String')),1);
            end
        else
            if Parameters.MAR(1) ~= 0 && Parameters.DCF(1) ~= 0
                waitfor(errordlg('Problem in MAR1 Result. Skipping.
Check your input for invalid or missing entries.'))
            end
            WasSuccessful(1) = 0;
```

```
                Parameters.CED1 =
zeros(str2double(get(handles.num_sample_text,'String')),1);
            end
            ChangeMARState(handles.radioMAR2, handles);
            if MARxisValid(handles)
                try
                    GetResults(handles);
                catch
                    if Parameters.MAR(2) ~= 0 && Parameters.DCF(2) ~= 0
                        waitfor(errordlg('Problem in MAR2 Result.
Skipping. Check your input for invalid or missing entries.'))
                    end
                    WasSuccessful(2) = 0;
                    Parameters.CED2 =
zeros(str2double(get(handles.num_sample_text,'String')),1);
                end
            else
                if Parameters.MAR(2) ~= 0 && Parameters.DCF(2) ~= 0
                    waitfor(errordlg('Problem in MAR2 Result. Skipping.
Check your input for invalid or missing entries.'))
                end
                WasSuccessful(2) = 0;
                Parameters.CED2 =
zeros(str2double(get(handles.num_sample_text,'String')),1);
            end
            ChangeMARState(handles.radioMAR3, handles);
            if MARxisValid(handles)
                try
                    GetResults(handles);
                catch
                    if Parameters.MAR(3) ~= 0 && Parameters.DCF(3) ~= 0
                        waitfor(errordlg('Problem in MAR3 Result.
Skipping. Check your input for invalid or missing entries.'))
                    end
                    WasSuccessful(3) = 0;
                    Parameters.CED3 =
zeros(str2double(get(handles.num_sample_text,'String')),1);
                end
            else
                if Parameters.MAR(3) ~= 0 && Parameters.DCF(3) ~= 0
                    waitfor(errordlg('Problem in MAR3 Result. Skipping.
Check your input for invalid or missing entries.'))
                end
                WasSuccessful(3) = 0;
                Parameters.CED3 =
zeros(str2double(get(handles.num_sample_text,'String')),1);
            end
            ChangeMARState(handles.radioMAR4, handles);
            if MARxisValid(handles)
```

```
                try
                    GetResults(handles);
                catch
                    if Parameters.MAR(4) ~= 0 && Parameters.DCF(4) ~= 0
                        waitfor(errordlg('Problem in MAR4 Result.
Skipping. Check your input for invalid or missing entries.'))
                    end
                    WasSuccessful(4) = 0;
                    Parameters.CED4 =
zeros(str2double(get(handles.num_sample_text,'String')),1);
                end
            else
                if Parameters.MAR(4) ~= 0 && Parameters.DCF(4) ~= 0
                    waitfor(errordlg('Problem in MAR4 Result. Skipping.
Check your input for invalid or missing entries.'))
                end
                WasSuccessful(4) = 0;
                Parameters.CED4 =
zeros(str2double(get(handles.num_sample_text,'String')),1);
            end
            switch OriginState  %Change MAR state back to what it was
before show all was clicked.
                case 1
                    ChangeMARState(handles.radioMAR1, handles);
                case 2
                    ChangeMARState(handles.radioMAR2, handles);
                case 3
                    ChangeMARState(handles.radioMAR3, handles);
                case 4
                    ChangeMARState(handles.radioMAR4, handles);
            end
            if any(WasSuccessful)  %If any MAR was successful in
getting a result, plot the result.
                Parameters = SumFinal(Parameters);

bar(handles.axes1,Parameters.SumX,Parameters.SumY,'FaceColor',[.2 .6
.6],'EdgeColor',[.2 .6 .6], 'BarWidth',1);

bar(handles.axes1,Parameters.SumX,Parameters.SumY,'FaceColor','m','Edge
Color','m','BarWidth', 1);
                str = sprintf('\\fontsize{11}CED, Mean = %0.3e rem,
Median= %0.3e rem, 95 Percentile = %0.3e
rem',Parameters.SumAvgCED,Parameters.SumMed,Parameters.Sum95CI);
                title(handles.axes1,str,'Units', 'normalized', ...
                    'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
                xlabel(handles.axes1,'Commited Effective Dose (rem)')
                ylabel(handles.axes1,'Probability Density')
```

```
                legend(handles.axes1,'Random
Generated','Location','NE')
                axis tight;
                grid on;

                assignin('base','cedxi', Parameters.SumX);
                assignin('base','cedfi2', Parameters.SumY);
                assignin('base','ced', Parameters.SumCED);
                setappdata(0,'ced',Parameters.SumCED);
                set(handles.fit_dist,'Enable','on');
                Parameters.SumCED = 0;
                Parameters.CED1 = 0;
                Parameters.CED2 = 0;
                Parameters.CED3 = 0;
                Parameters.CED4 = 0;
            else
                msgbox('No Complete/Valid Data Entries, no Data to be
displayed.')
            end



            %Enable show plot button for all parameter when running ced
plot
            if marbutton == 1
                set(handles.mar_pushbutton,'Enable','on');
            end
            if drbutton == 1
                set(handles.dr_pushbutton,'Enable','on');
            end
            if arfbutton == 1
                set(handles.arf_pushbutton,'Enable','on');
            end
            if rfbutton == 1
                set(handles.rf_pushbutton,'Enable','on');
            end
            if lpfbutton == 1
                set(handles.lpf_pushbutton,'Enable','on');
            end
            if dcfbutton == 1
                set(handles.dcf_pushbutton,'Enable','on');
            end
            if cqbutton == 1
                set(handles.cq_pushbutton,'Enable','on');
            end

            %Enable text1 all parameter
            if martext1 == 1
                set(handles.mar_text1,'Enable','on');
```

```
            end
            if drtext1 == 1
                set(handles.dr_text1,'Enable','on');
            end
            if arftext1 == 1
                set(handles.arf_text1,'Enable','on');
            end
            if rftext1 == 1
                set(handles.rf_text1,'Enable','on');
            end
            if lpftext1 == 1
                set(handles.lpf_text1,'Enable','on');
            end
            if dcftext1 == 1
                set(handles.dcf_text1,'Enable','on');
            end
            if cqtext1 == 1
                set(handles.cq_text1,'Enable','on');
            end

            %Enable text2 for all parameter
            if martext2 == 1
                set(handles.mar_text2,'Enable','on');
            end
            if drtext2 == 1
                set(handles.dr_text2,'Enable','on');
            end
            if arftext2 == 1
                set(handles.arf_text2,'Enable','on');
            end
            if rftext2 == 1
                set(handles.rf_text2,'Enable','on');
            end
            if lpftext2 == 1
                set(handles.lpf_text2,'Enable','on');
            end
            if dcftext2 == 1
                set(handles.dcf_text2,'Enable','on');
            end

            %Enable radio buttons after run
            set(handles.radioMAR1,'Enable','on')
            set(handles.radioMAR2,'Enable','on')
            set(handles.radioMAR3,'Enable','on')
            set(handles.radioMAR4,'Enable','on')
        else
            errordlg('Please enter number of samples','Sample
Number','modal');
        end
```

```
    else
        msgbox('Error in selected MAR Data. Please Correct.','Input
Invalid');
    end
    set(handles.run_pushbutton,'str','Show Plot','backg',col);
else
    return
end
end

% --- Executes on button press in rf_togglebutton.
% When the toggle button of RF is pressed this codes are exceuted.
% Toggle button can be on on and off postion ON mean "Single Input"
% OFF means "Distribution Input"
function rf_togglebutton_Callback(hObject, ~, handles)
% hObject    handle to rf_togglebutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of rf_togglebutton
ispushed = get(hObject,'Value');

if ispushed
    set(hObject,'string','Single Input');
    set(handles.rf_text1,'Enable','on');
    set(handles.rf_text1,'String','');
    set(handles.rf_text2,'String','');
    set(handles.rf_text2,'Enable','off')  %
    set(handles.rf_pushbutton,'Enable','off')  %
    set(handles.rf_popup_dist,'Enable','off')  %
    set(handles.rf_popup_dist,'Value',1)

else
    set(hObject,'string','Distribution Input');
    set(handles.rf_text1,'String','');
    set(handles.rf_text2,'String','');
    set(handles.rf_text1,'Enable','off')
    set(handles.rf_text2,'Enable','off')  %
    %     set(handles.rf_pushbutton,'Enable','on')  %
    set(handles.rf_popup_dist,'Enable','on')  %
end
end

% --- Executes on button press in dr_togglebutton.
% When the toggle button of DR is pressed this codes are exceuted.
% Toggle button can be on on and off postion ON mean "Single Input"
% OFF means "Distribution Input"
function dr_togglebutton_Callback(hObject, ~, handles)
```

```
% hObject    handle to dr_togglebutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of dr_togglebutton
ispushed = get(hObject,'Value');

if ispushed
    set(hObject,'string','Single Input');
    set(handles.dr_text1,'Enable','on');
    set(handles.dr_text1,'String','');
    set(handles.dr_text2,'String','');
    set(handles.dr_text2,'Enable','off') ; %
    set(handles.dr_pushbutton,'Enable','off');  %
    set(handles.dr_popup_dist,'Enable','off');  %
    set(handles.dr_popup_dist,'Value',1)

else
    set(hObject,'string','Distribution Input');
    set(handles.dr_text1,'String','');
    set(handles.dr_text2,'String','');
    set(handles.dr_text1,'Enable','off');
    set(handles.dr_text2,'Enable','off');  %
    set(handles.dr_popup_dist,'Enable','on');  %

end
end

% --- Executes on button press in cq_pushbutton.
% THis function computes Chi/Q Using gaussian approxmiation
function cq_pushbutton_Callback(hObject, ~, handles)
% hObject    handle to cq_pushbutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

sample = get(handles.num_sample_text,'String');
samplesize = str2double(sample);
if strcmp(sample,'') == 1 || samplesize < 0
    errordlg('Please enter number of samples','Sample Number','modal');
    return;
end
col = get(handles.cq_pushbutton,'backg');
set(handles.cq_pushbutton,'str','RUNNING...','backg',[.2 .6 .6]);
distance = str2double(get(handles.distance_text1,'String'));
distance1 = str2double(get(handles.distance_text2,'String'));
pd = makedist('Normal','mu',0,'sigma',distance1);
crossdistance = random(pd,samplesize,1);

num1 = str2double(get(handles.windspeed_text1,'String'));
```

```
num2 = str2double(get(handles.windspeed_text2,'String'));

contents = get(handles.windspeed_popup_dist,'String');
popupmenuvalue = contents{get(handles.windspeed_popup_dist,'Value')};
switch popupmenuvalue
    case 'Normal'
        pd = makedist('Normal','mu',num1,'sigma',num2);
        t = truncate(pd,0.1,inf);
        windS = random(t,samplesize,1);
    case 'Uniform'
        if num1 < num2;
            % In unifrom distribution upper limt must be greater than
lower
            % limit, if not show the error message
            errordlg('Upper Limit is less than lower limt','Uniform
Distribution')
            set(handles.cq_pushbutton,'str','Show Plot','backg',col);
            return;
        else
            pd = makedist('Uniform','Upper',num1,'Lower',num2);
            t = truncate(pd,0.1,inf);
            windS = random(t,samplesize,1);
        end
end



contents2 = get(handles.terrain_popup,'String');
terrainvalue = contents2{get(handles.terrain_popup,'Value')};

contents3 = get(handles.stability_popup,'String');
stability = contents3{get(handles.stability_popup,'Value')};


height = str2double(get(handles.height_text,'String'));


switch terrainvalue
    case 'Rural/Open Country'
        switch stability
            case 'A'
                sigma_y = 0.22*distance*(1+0.0001*distance)^(-0.5);
                sigma_z = 0.20*distance;
            case 'B'
                sigma_y = 0.16*distance*(1+0.0001*distance)^(-0.5);
                sigma_z = 0.12*distance;
            case 'C'
                sigma_y = 0.11*distance*(1+0.0001*distance)^(-0.5);
                sigma_z = 0.08*distance*(1+0.0002*distance)^(-0.5);
```

```
            case 'D'
                sigma_y = 0.08*distance*(1+0.0001*distance)^(-0.5);
                sigma_z = 0.06*distance*(1+0.0015*distance)^(-0.5);
            case 'E'
                sigma_y = 0.06*distance*(1+0.0001*distance)^(-0.5);
                sigma_z = 0.03*distance*(1+0.0003*distance)^(-1);
            case 'F'
                sigma_y = 0.04*distance*(1+0.0001*distance)^(-0.5);
                sigma_z = 0.016*distance*(1+0.0003*distance)^(-1);
            case 'Select Stability Condition'
                errordlg('Select Stability
Conditions','Error','modal');
                set(handles.cq_pushbutton,'str','Show
Plot','backg',col);
                return;
        end
    case 'Select Terrain'
        switch stability
            case 'A'
                errordlg('Select terrain','Error','modal');
                set(handles.cq_pushbutton,'str','Show
Plot','backg',col);
                return;
            case 'B'
                errordlg('Select terrain''Error','modal')
                set(handles.cq_pushbutton,'str','Show
Plot','backg',col);
                return;
            case 'C'
                errordlg('Select terrain''Error','modal');
                set(handles.cq_pushbutton,'str','Show
Plot','backg',col);
                return;
            case 'D'
                errordlg('Select terrain''Error','modal');
                set(handles.cq_pushbutton,'str','Show
Plot','backg',col);
                return;
            case 'E'
                errordlg('Select terrain''Error','modal');
                set(handles.cq_pushbutton,'str','Show
Plot','backg',col);
                return;
            case 'F'
                errordlg('Select terrain''Error','modal');
                set(handles.cq_pushbutton,'str','Show
Plot','backg',col);
                return
            case 'Select Stability Condition'
```

```
                errordlg('Select Terrain & Stability
Condition''Error','modal');
                set(handles.cq_pushbutton,'str','Show
Plot','backg',col);
                return;
        end
    case 'Urban Area'
        switch stability
            case 'A-B'
                sigma_y = 0.32*distance*(1+0.0004*distance)^(-0.5);
                sigma_z = 0.24*distance*(1+0.001*distance)^(0.5);
            case 'C'
                sigma_y = 0.22*distance*(1+0.0004*distance)^(-0.5);
                sigma_z = 0.2*distance;
            case 'D'
                sigma_y = 0.16*distance*(1+0.0004*distance)^(-0.5);
                sigma_z = 0.14*distance*(1+0.0003*distance)^(-0.5);
            case 'E-F'
                sigma_y = 0.11*distance*(1+0.0004*distance)^(-0.5);
                sigma_z = 0.08*distance*(1+0.0015*distance)^(-0.5);
            case 'Select Stability Condition'
                errordlg('Select Stability Conditions''Error','modal');
                set(handles.cq_pushbutton,'str','Show
Plot','backg',col);
                return;
        end

end


c = (exp((-crossdistance.^2/(2*(sigma_y)^2))-
(height^2/(2*(sigma_z)^2))./...
    (pi*windS.*sigma_y*sigma_z));
n=c;
cla(handles.axes1,'reset');
axes(handles.axes1)
nbins = max(min(length(n)./10,100),50);
xi = linspace(min(n),max(n),nbins);
dx = mean(diff(xi));
fi = histc(n,xi-dx);
fi = fi./sum(fi)./dx;
assignin('base','cqxi', xi);
assignin('base','cqfi2', fi);
bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],'BarWidth', 1);
axis tight;
% hist(c,50)
xlabel('Chi/Q')
ylabel('Probability Density')
```

```
str = sprintf('\\fontsize{12} \\chi/Q distribution plot with\\mu=%0.2e
s/m^3 ,\\sigma =%0.2e s/m^3',...
            mean(n),std(n));
        title(str,'Units', 'normalized', ...
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')
set(handles.cq_pushbutton,'str','Show Plot','backg',col);
assignin('base','cq', c);
end
```

```
function cq_text1_Callback(hObject, ~, handles)
% hObject    handle to cq_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end
% Hints: get(hObject,'String') returns contents of cq_text1 as text
%        str2double(get(hObject,'String')) returns contents of cq_text1
as a double


% --- Executes during object creation, after setting all properties.
function cq_text1_CreateFcn(hObject, ~, handles)
% hObject    handle to cq_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on selection change in cq_popup_dist.
% Executed when user select from a list of distribution in chi/Q
function cq_popup_dist_Callback(hObject, ~, handles)
% hObject    handle to cq_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
contents = cellstr(get(hObject,'String'));
cqpopchoice = contents{get(hObject,'Value')};
```

```
switch cqpopchoice
    case 'Normal'
        set(handles.cq_text1,'Enable','inactive')  %
        set(handles.cq_text2,'Enable','inactive')  %
        set(handles.cq_pushbutton,'Enable','on')  %
        set(handles.cq_text1,'String','Mean');
        set(handles.cq_text2,'String','Std Deviation');
        set(handles.cq_text1,'TooltipString','')
        set(handles.cq_text2,'TooltipString','')
    case 'Beta'
        set(handles.cq_text1,'Enable','inactive')  %
        set(handles.cq_text2,'Enable','inactive')  %
        set(handles.cq_pushbutton,'Enable','on')  %
        set(handles.cq_text1,'String','a');
        set(handles.cq_text2,'String','b');
        set(handles.cq_text1,'TooltipString','shape parameter')
        set(handles.cq_text2,'TooltipString','shape parameter')
    case 'Uniform'
        set(handles.cq_text1,'Enable','inactive')  %
        set(handles.cq_text2,'Enable','inactive')  %
        set(handles.cq_pushbutton,'Enable','on')  %
        set(handles.cq_text1,'String','Upper Limit');
        set(handles.cq_text2,'String','Lower Limit');
        set(handles.cq_text1,'TooltipString','')
        set(handles.cq_text2,'TooltipString','')
    case 'Exponential'
        set(handles.cq_text1,'Enable','inactive')  %
        set(handles.cq_text2,'Enable','off')  %
        set(handles.cq_pushbutton,'Enable','on')  %
        set(handles.cq_text1,'String','Mean');
        set(handles.cq_text1,'TooltipString','')
        set(handles.cq_text2,'TooltipString','')
    case 'Select Distribution'
        set(handles.cq_text1,'String','');
        set(handles.cq_text2,'String','');
        set(handles.cq_text1,'Enable','off')  %
        set(handles.cq_text2,'Enable','off')  %
        set(handles.cq_pushbutton,'Enable','off')  %
        set(handles.cq_text1,'TooltipString','')
        set(handles.cq_text2,'TooltipString','')
end
end
% Hints: contents = cellstr(get(hObject,'String')) returns
cq_popup_dist contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
cq_popup_dist


% --- Executes during object creation, after setting all properties.
```

```
function cq_popup_dist_CreateFcn(hObject, ~, handles)
% hObject    handle to cq_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on button press in cq_togglebutton.
% Executed for Chi/Q toggle button is pressed
function cq_togglebutton_Callback(hObject, ~, handles)
% hObject    handle to cq_togglebutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ispushed = get(hObject,'Value');

if ispushed
    set(hObject,'string','Single Input');
    set(handles.cq_text1,'Enable','on');
    set(handles.cq_text1,'String','');
    set(handles.windspeed_text1,'String','');
    set(handles.windspeed_text2,'String','');
    set(handles.windspeed_text1,'Enable','off');
    set(handles.windspeed_text2,'Enable','off');
    set(handles.height_text,'String','');
    set(handles.height_text,'Enable','off') ; %
    set(handles.cq_pushbutton,'Enable','off');  %
    set(handles.terrain_popup,'Enable','off');  %
    set(handles.terrain_popup,'Value',1)
    set(handles.stability_popup,'Enable','off');  %
    set(handles.stability_popup,'Value',1)
    set(handles.windspeed_popup_dist,'Enable','off');  %
    set(handles.windspeed_popup_dist,'Value',1);
    set(handles.distance_text1,'String','');
    set(handles.distance_text1,'Enable','off') ; %
    set(handles.distance_text2,'String','');
    set(handles.distance_text2,'Enable','off') ; %
else
    set(hObject,'string','Distribution Input');
    set(handles.cq_text1,'Enable','off');
    set(handles.cq_text1,'String','');
    set(handles.windspeed_text1,'String','');
    set(handles.windspeed_text1,'Enable','on') ;
```

```
    set(handles.windspeed_text2,'String','');
    set(handles.windspeed_text2,'Enable','on') ; %
    set(handles.height_text,'String','Height');
    set(handles.height_text,'Enable','on') ; %
    set(handles.cq_pushbutton,'Enable','off');  %
    set(handles.terrain_popup,'Enable','on');  %
    set(handles.terrain_popup,'Value',1)
    set(handles.stability_popup,'Enable','off');  %
    set(handles.stability_popup,'Value',1)
    set(handles.distance_text1,'String','');
    set(handles.distance_text1,'Enable','on') ; %
    set(handles.distance_text2,'String','');
    set(handles.distance_text2,'Enable','on') ; %
    set(handles.windspeed_popup_dist,'Enable','on');  %
    set(handles.windspeed_popup_dist,'Value',1)
    set(handles.windspeed_text1,'String','');
    set(handles.windspeed_text1,'Enable','off') ; %
    set(handles.windspeed_text2,'String','');
    set(handles.windspeed_text2,'Enable','off') ; %
end
end
% Hint: get(hObject,'Value') returns toggle state of cq_togglebutton


% --- Executes on selection change in dcf_popup_dist.
% exceuted when Dose conversion factor distribution is selected
function dcf_popup_dist_Callback(hObject, ~, handles)
% hObject    handle to dcf_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Parameters;
global CurrentMAR;
contents = cellstr(get(hObject,'String'));
dcfpopchoice = contents{get(hObject,'Value')};
switch dcfpopchoice
    case 'Normal'
        set(handles.dcf_text1,'Enable','inactive')  %
        set(handles.dcf_text2,'Enable','inactive')  %
        set(handles.dcf_pushbutton,'Enable','on')  %
        set(handles.dcf_text1,'String','Mean');
        set(handles.dcf_text2,'String','Std Deviation');
    case 'Beta'
        set(handles.dcf_text1,'Enable','inactive')  %
        set(handles.dcf_text2,'Enable','inactive')  %
        set(handles.dcf_pushbutton,'Enable','on')  %
        set(handles.dcf_text1,'String','a');
        set(handles.dcf_text2,'String','b');
        set(handles.dcf_text1,'TooltipString','shape parameter')
        set(handles.dcf_text2,'TooltipString','shape parameter')
```

```
    case 'Uniform'
        set(handles.dcf_text1,'Enable','inactive')  %
        set(handles.dcf_text2,'Enable','inactive')  %
        set(handles.dcf_pushbutton,'Enable','on')  %
        set(handles.dcf_text1,'String','Upper Limit');
        set(handles.dcf_text2,'String','Lower Limit');
    case 'Exponential'
        set(handles.dcf_text1,'Enable','inactive')  %
        set(handles.dcf_text2,'Enable','off')  %
        set(handles.dcf_pushbutton,'Enable','on')  %
        set(handles.dcf_text1,'String','Mean');
        set(handles.dcf_text2,'String','');
    case 'Select Distribution'
        set(handles.dcf_text1,'String','');
        set(handles.dcf_text2,'String','');
        set(handles.dcf_text1,'Enable','off')  %
        set(handles.dcf_text2,'Enable','off')  %
        set(handles.dcf_pushbutton,'Enable','off')  %
    case 'User Defined'
        set(handles.dcf_text1,'Enable','off')
        set(handles.dcf_text2,'Enable','off')
        set(handles.dcf_pushbutton,'Enable','on')
        set(handles.dcf_text1,'String','User');
        set(handles.dcf_text2,'String','Defined');
        set(handles.dcf_text1,'TooltipString','')
        set(handles.dcf_text2,'TooltipString','')
        Parameters = UserDefined(Parameters);
        [Parameters, msg, flag] = Parameters.CheckUDD('DCF');
        if flag == 1
            msgbox(msg);
            set(Parameters,'UDtempX',0);
            set(Parameters,'UDtempY',0);
            set(hObject,'Value', 1);
            dcf_popup_dist_Callback(hObject, '', handles);
        else
            Parameters = Parameters.SaveUDD(CurrentMAR,'DCF');
        end
    case 'U-238'  %Old features code maintained, but not in use.
        set(handles.dcf_text1,'String','5.0*10^-7');
    case 'Select Isotope'  %Repurposed as an alternative way to access
the MAR selection screen.
        set(handles.dcf_text1,'String','');
        MARbtn_Callback(handles.MARbtn, '' , handles);
    case 'Pu-239'
        set(handles.dcf_text1,'String','1.2*10^-4');
    case 'Pu-235'
        set(handles.dcf_text1,'String','1.0*10^-12');
    case 'U-239'
        set(handles.dcf_text1,'String','1.0*10^-11');
```

```
end
end
% Hints: contents = cellstr(get(hObject,'String')) returns
dcf_popup_dist contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
dcf_popup_dist


% --- Executes during object creation, after setting all properties.
function dcf_popup_dist_CreateFcn(hObject, ~, handles)
% hObject    handle to dcf_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function dcf_text1_Callback(hObject, ~, handles)
% hObject    handle to dcf_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end
% Hints: get(hObject,'String') returns contents of dcf_text1 as text
%        str2double(get(hObject,'String')) returns contents of
dcf_text1 as a double


% --- Executes during object creation, after setting all properties.
function dcf_text1_CreateFcn(hObject, ~, handles)
% hObject    handle to dcf_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end
```

```
function dcf_text2_Callback(hObject, ~, handles)
% hObject    handle to dcf_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end
% Hints: get(hObject,'String') returns contents of dcf_text2 as text
%        str2double(get(hObject,'String')) returns contents of
dcf_text2 as a double


% --- Executes during object creation, after setting all properties.
function dcf_text2_CreateFcn(hObject, ~, handles)
% hObject    handle to dcf_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on button press in dcf_pushbutton.
% Executed whn DCF show plot button is pressed
function dcf_pushbutton_Callback(hObject, ~, handles)
% hObject    handle to dcf_pushbutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Parameters;
global CurrentMAR;
sample = get(handles.num_sample_text,'String');
samplesize = str2double(sample);
if strcmp(sample,'') == 1 || samplesize < 0
    errordlg('Please enter number of samples','Sample Number','modal');
    return;
end
col = get(handles.dcf_pushbutton,'backg');
set(handles.dcf_pushbutton,'str','RUNNING...','backg',[.2 .6 .6]);
pause(eps);
num1 = str2double(get(handles.dcf_text1,'String'));
num2 = str2double(get(handles.dcf_text2,'String'));
contents = get(handles.dcf_popup_dist,'String');
popupmenuvalue = contents{get(handles.dcf_popup_dist,'Value')};
cla(handles.axes1,'reset');
switch popupmenuvalue
```

```
    case 'Normal'
        result = InputIsValid(handles.dcf_text1, 'DCF', '');
        result2 = InputIsValid(handles.dcf_text2, 'DCF', 'LL');
        if result && result2
            pd = makedist('Normal','mu',num1,'sigma',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','dcfxi', xi);
            assignin('base','dcffi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6
.6],'BarWidth', 1);
            axis tight;
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('DCF');
            str = sprintf('\\fontsize{12} DCF distribution plot with
Normal distribution with\\mu=%0.2e Sv/Bq,\\sigma =%0.2e Sv/Bq',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in dcf_text1, dcf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in dcf_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in dcf_text2, invalid
input.','Invalid Input','modal');
            end
        end
    case 'Log Normal'
        result = InputIsValid(handles.dcf_text1, 'DCF', '');
        result2 = InputIsValid(handles.dcf_text2, 'DCF', 'LL');
        if result && result2
            pd = makedist('Lognormal','mu',num1,'sigma',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
```

```
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','drxi', xi);
            assignin('base','drfi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            axis tight;
            ylabel('Probability Density');
            xlabel('DCF');
            str = sprintf('\\fontsize{12} DCF distribution plot with
Log Normal distribution with \\mu=%0.2e , \\sigma =%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in dcf_text1, dcf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in dcf_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in dcf_text2, invalid
input.','Invalid Input','modal');
            end
        end
    case 'Beta'
        result = InputIsValid(handles.dcf_text1, 'DCF', 'ab');
        result2 = InputIsValid(handles.dcf_text2, 'DCF', 'ab');
        if result && result2
            pd = makedist('Beta','a',num1,'b',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','dcfxi', xi);
            assignin('base','dcffi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6
.6],'BarWidth', 1);
            axis tight;
            %         hist(n,50);
```

```
            ylabel('Probability Density');
            xlabel('DCF');
            str = sprintf('\\fontsize{12} DCF distribution plot with
Beta distribution with\\mu=%0.2e Sv/Bq,\\sigma =%0.2e Sv/Bq',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in dcf_text1, dcf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in dcf_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in dcf_text2, invalid
input.','Invalid Input','modal');
            end
        end
    case 'Uniform'
        result = InputIsValid(handles.dcf_text1, 'DCF', '');
        result2 = InputIsValid(handles.dcf_text2, 'DCF', 'LL');
        if result && result2
            if num1 < num2;
                % In unifrom distribution upper limt must be greater
than lower
                % limit, if not show the error message
                errordlg('Upper Limit is less than lower limt','Uniform
Distribution','modal')
                set(handles.dcf_pushbutton,'str','Show
Plot','backg',col);
                return;
            else
                pd = makedist('Uniform','Upper',num1,'Lower',num2);
                t = truncate(pd,0,1);
                n = random(t,samplesize,1);
                axes(handles.axes1)
                nbins = max(min(length(n)./10,100),50);
                xi = linspace(min(n),max(n),nbins);
                dx = mean(diff(xi));
                fi = histc(n,xi-dx);
                fi = fi./sum(fi)./dx;
                assignin('base','dcfxi', xi);
                assignin('base','dcffi2', fi);
                bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6
.6],'BarWidth', 1);
                axis tight;
                %             hist(n,50);
```

```
                ylabel('Probability Density');
                xlabel('DCF');
                str = sprintf('\\fontsize{12} DCF distribution plot
with Uniform distribution with\\mu=%0.2e Sv/Bq,\\sigma =%0.2e
Sv/Bq',...
                    mean(n),std(n));
                title(str,'Units', 'normalized', ...
                    'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
            end
        else
            if ~result && ~result2
                errordlg('Problem in dcf_text1, dcf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in dcf_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in dcf_text2, invalid
input.','Invalid Input','modal');
            end
        end
    case 'Exponential'
        result = InputIsValid(handles.dcf_text1, 'DCF', '');
        if result
            pd = makedist('Exponential','mu',num1);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','dcfxi', xi);
            assignin('base','dcffi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('DCF');
            str = sprintf('\\fontsize{12} DCF distribution plot with
Exponential distribution with\\mu=%0.2e Sv/Bq,\\sigma =%0.2e Sv/Bq',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
```

```
        else
            errordlg('Problem in dcf_text1, invalid input.','Invalid
Input','modal');
        end
    case 'User Defined'
        [Parameters,X,Y] = Parameters.GetUDD(CurrentMAR,'DCF');
        n = zeros(1,samplesize);
        for e = 1:samplesize;
            num_rand=rand;
            ter = size(X);
            for i = 1:ter(2)
                iSum = 0;
                for j = 1:i
                    iSum = iSum + Y(j);
                end
                if num_rand < iSum
                    if i == 1
                        n(e) = rand*(X(i+1)-X(i))+X(i);
                    else
                        n(e) = rand*(X(i)-X(i-1))+X(i);
                    end
                    break;
                end
            end
        end
        axes(handles.axes1)
        nbins = max(min(length(n)./10,100),50);
        xi = linspace(min(n),max(n),nbins);
        dx = mean(diff(xi));
        fi = histc(n,xi-dx);
        fi = fi./sum(fi)./dx;
        assignin('base','drxi', xi);
        assignin('base','drfi2', fi);
        bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
        axis tight;
        %         hist(n,50);
        ylabel('Probability Density');
        xlabel('DCF');
        str = sprintf('\\fontsize{12} DCF distribution plot with User
Defined Distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
            mean(n),std(n));
        title(str,'Units', 'normalized', ...
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')
end
set(handles.dcf_pushbutton,'str','Show Plot','backg',col);
end

% --- Executes on button press in dcf_togglebutton.
```

```
function dcf_togglebutton_Callback(hObject, ~, handles)
% hObject    handle to dcf_togglebutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Parameters;
global CurrentMAR;
ispushed = get(hObject,'Value');

if ispushed
    set(hObject,'string','Single Input');
    %set(handles.dcf_popup_dist,'String',{'Select Isotope';'U-238';'U-
239';'Pu-239';'Pu-235'}...
    %    ,'Value',1,'Enable','on');
    if ~strcmp(Parameters.Isotope{CurrentMAR} , '')

set(handles.dcf_popup_dist,'String',{Parameters.Isotope{CurrentMAR},'Se
lect Isotope'}...
        ,'Value',1,'Enable','on');
        set(handles.dcf_text1,'String',Parameters.DCF(CurrentMAR));
    else
        set(handles.dcf_popup_dist,'String',{'Select Isotope'}...
        ,'Value',1,'Enable','on');
        set(handles.dcf_text1,'String','');
    end
    set(handles.dcf_text1,'Enable','on');

    set(handles.dcf_text2,'String','');
    set(handles.dcf_text2,'Enable','off') ; %
    set(handles.dcf_pushbutton,'Enable','off');  %

else
    set(hObject,'string','Distribution Input');
    set(handles.dcf_popup_dist,'String',{'Select
Distribution';'Normal';...
        'Beta';'Uniform';'Exponential';'User Defined'},'Value',1);
    set(handles.dcf_text1,'String','');
    set(handles.dcf_text2,'String','');
    set(handles.dcf_text1,'Enable','off');
    set(handles.dcf_text2,'Enable','off');  %
    set(handles.dcf_popup_dist,'Enable','on');  %

end
end
% Hint: get(hObject,'Value') returns toggle state of dcf_togglebutton


% --- Executes on button press in br_pushbutton.
% executed when Breathing rate push button is pressed
function br_pushbutton_Callback(hObject, ~, handles)
```

```
% hObject    handle to br_pushbutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sample = get(handles.num_sample_text,'String');
samplesize = str2double(sample);
if strcmp(sample,'') == 1 || samplesize < 0
    errordlg('Please enter number of samples','Sample Number','modal');
    return;
end
col = get(handles.br_pushbutton,'backg');
set(handles.br_pushbutton,'str','RUNNING...','backg',[.2 .6 .6]);
pause(eps);
a = 8.33*10^-4;
b= 4.17*10^-4;
c= 1.5*10^-4;
d= 1.25*10^-4;

for e = 1:samplesize;
    num_rand=rand;
    if num_rand <= 0.17
        n(e) = rand*(a-b)+b;
    elseif num_rand > 0.17 && num_rand <= 0.34;
        n(e) = rand*(b-c)+c;
    elseif num_rand >0.34
        n(e) = rand*(c-d)+d;
    end
end
n=n';
cla(handles.axes1,'reset');
axes(handles.axes1);
nbins = max(min(length(n)./10,100),50);
xi = linspace(min(n),max(n),nbins);
dx = mean(diff(xi));
fi = histc(n,xi-dx);
fi = fi./sum(fi)./dx;
assignin('base','brxi', xi);
assignin('base','brfi2', fi);
bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6], 'BarWidth',1);
axis tight;
% hist(n,50);
xlabel('Breathing Rate')
ylabel('Probability Density')
str = sprintf('BR distribution plot with\\mu=%0.3e m^3/s ,\\sigma
=%0.3e m^3/s',...
            mean(n),std(n));
%title(str); %Replaced with below to standardize between plots.
title(str,'Units', 'normalized', ...
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')
set(handles.br_pushbutton,'str','Show Plot','backg',col);
```

```
assignin('base','br', n);
end

function br_text1_Callback(hObject, ~, handles)
% hObject    handle to br_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end
% Hints: get(hObject,'String') returns contents of br_text1 as text
%        str2double(get(hObject,'String')) returns contents of br_text1
as a double


% --- Executes during object creation, after setting all properties.
function br_text1_CreateFcn(hObject, ~, handles)
% hObject    handle to br_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on button press in br_togglebutton.
% Executed when Breathing rate toogle button is pressed
function br_togglebutton_Callback(hObject, ~, handles)
% hObject    handle to br_togglebutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ispushed = get(hObject,'Value');

if ispushed
    set(hObject,'string','Single Input');
    set(handles.br_text1,'Enable','on');
    set(handles.br_text1,'String','');
    set(handles.br_pushbutton,'Enable','off');  %

else
    set(hObject,'string','Distribution Input');
    set(handles.br_text1,'Enable','off','String','');
    set(handles.br_pushbutton,'Enable','on');
end
end
% Hint: get(hObject,'Value') returns toggle state of br_togglebutton
```

```
% --- Executes on selection change in lpf_popup_dist.
% Executed when Leak path factor is pressed
function lpf_popup_dist_Callback(hObject, ~, handles)
% hObject    handle to lpf_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global Parameters;
global CurrentMAR;
contents = cellstr(get(hObject,'String'));
lpfpopchoice = contents{get(hObject,'Value')};
switch lpfpopchoice
    case 'Normal'
        set(handles.lpf_text1,'Enable','inactive')
        set(handles.lpf_text2,'Enable','inactive')
        set(handles.lpf_pushbutton,'Enable','on')
        set(handles.lpf_text1,'String','Mean');
        set(handles.lpf_text2,'String','Std Deviation');
        set(handles.lpf_text1,'TooltipString','')
        set(handles.lpf_text2,'TooltipString','')
    case 'Beta'
        set(handles.lpf_text1,'Enable','inactive')
        set(handles.lpf_text2,'Enable','inactive')
        set(handles.lpf_pushbutton,'Enable','on')
        set(handles.lpf_text1,'String','a');
        set(handles.lpf_text2,'String','b');
        set(handles.lpf_text1,'TooltipString','shape parameter')
        set(handles.lpf_text2,'TooltipString','shape parameter')
        %
set(handles.lpf_text1,'FontName','SymbolPi','String','a');
        %
set(handles.lpf_text2,'FontName','SymbolPi','String','b');
    case 'Uniform'
        set(handles.lpf_text1,'Enable','inactive')
        set(handles.lpf_text2,'Enable','inactive')
        set(handles.lpf_pushbutton,'Enable','on')
        set(handles.lpf_text1,'String','Upper Limit');
        set(handles.lpf_text2,'String','Lower Limit');
        set(handles.lpf_text1,'TooltipString','')
        set(handles.lpf_text2,'TooltipString','')
    case 'Exponential'
        set(handles.lpf_text1,'Enable','inactive')
        set(handles.lpf_text2,'Enable','off')
        set(handles.lpf_pushbutton,'Enable','on')
        set(handles.lpf_text1,'String','Mean');
        set(handles.lpf_text2,'String','');
        set(handles.lpf_text1,'TooltipString','')
```

```
        set(handles.lpf_text2,'TooltipString','')
    case 'Select Distribution'
        set(handles.lpf_text1,'String','');
        set(handles.lpf_text2,'String','');
        set(handles.lpf_text1,'Enable','off')
        set(handles.lpf_text2,'Enable','off')
        set(handles.lpf_pushbutton,'Enable','off')
        set(handles.lpf_text1,'TooltipString','')
        set(handles.lpf_text2,'TooltipString','')
    case 'Log Normal'
        set(handles.lpf_text1,'Enable','inactive')  %
        set(handles.lpf_text2,'Enable','inactive')  %
        set(handles.lpf_pushbutton,'Enable','on')  %
        set(handles.lpf_text1,'String',{'Mode'});
        set(handles.lpf_text2,'String',{'Scale Param.'});
        set(handles.lpf_text1,'TooltipString','')
        set(handles.lpf_text2,'TooltipString','')
    case 'User Defined'
        set(handles.lpf_text1,'Enable','off')
        set(handles.lpf_text2,'Enable','off')
        set(handles.lpf_pushbutton,'Enable','on')
        set(handles.lpf_text1,'String','User');
        set(handles.lpf_text2,'String','Defined');
        set(handles.lpf_text1,'TooltipString','')
        set(handles.lpf_text2,'TooltipString','')
        Parameters = UserDefined(Parameters);
        [Parameters, msg, flag] = Parameters.CheckUDD('LPF');
        if flag == 1
            msgbox(msg);
            set(Parameters,'UDtempX',0);
            set(Parameters,'UDtempY',0);
            set(hObject,'Value', 1);
            lpf_popup_dist_Callback(hObject, '', handles);
        else
            Parameters = Parameters.SaveUDD(CurrentMAR,'LPF');
        end
end
end
% Hints: contents = cellstr(get(hObject,'String')) returns
lpf_popup_dist contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
lpf_popup_dist


% --- Executes during object creation, after setting all properties.
function lpf_popup_dist_CreateFcn(hObject, ~, handles)
% hObject    handle to lpf_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function lpf_text1_Callback(hObject, ~, handles)
% hObject    handle to lpf_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end
% Hints: get(hObject,'String') returns contents of lpf_text1 as text
%        str2double(get(hObject,'String')) returns contents of
lpf_text1 as a double


% --- Executes during object creation, after setting all properties.
function lpf_text1_CreateFcn(hObject, ~, handles)
% hObject    handle to lpf_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function lpf_text2_Callback(hObject, ~, handles)
% hObject    handle to lpf_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end
% Hints: get(hObject,'String') returns contents of lpf_text2 as text
%        str2double(get(hObject,'String')) returns contents of
lpf_text2 as a double


% --- Executes during object creation, after setting all properties.
```

```
function lpf_text2_CreateFcn(hObject, ~, handles)
% hObject    handle to lpf_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on button press in lpf_pushbutton.
% Excuted when leak path factor show plot button is pressed
function lpf_pushbutton_Callback(hObject, ~, handles)
% hObject    handle to lpf_pushbutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% samplesize = str2double(get(handles.num_sample_text,'String'));
% samplesize = get(handles.num_sample_text,'String');
% if strcmp(samplesize,'') || num2str(samplesize) < 1
%
global Parameters;
global CurrentMAR;
sample = get(handles.num_sample_text,'String');
samplesize = str2double(sample);
if strcmp(sample,'') == 1 || samplesize < 0
    errordlg('Please enter number of samples','Sample Number','modal');
    return;
end
col = get(handles.lpf_pushbutton,'backg');
set(handles.lpf_pushbutton,'str','RUNNING...','backg',[.2 .6 .6]);
pause(eps);
num1 = str2double(get(handles.lpf_text1,'String'));
num2 = str2double(get(handles.lpf_text2,'String'));
contents = get(handles.lpf_popup_dist,'String');
popupmenuvalue = contents{get(handles.lpf_popup_dist,'Value')};
cla(handles.axes1,'reset');
switch popupmenuvalue
    case 'Normal'
        result = InputIsValid(handles.lpf_text1, 'LPF', '');
        result2 = InputIsValid(handles.lpf_text2, 'LPF', 'Sig');
        if result && result2
            pd = makedist('Normal','mu',num1,'sigma',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
```

```
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','lpfxi', xi);
            assignin('base','lpffi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('LPF');
            str = sprintf('\\fontsize{12} LPF distribution plot with
Normal distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')

        else
            if ~result && ~result2
                errordlg('Problem in lpf_text1, lpf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in lpf_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in lpf_text2, invalid
input.','Invalid Input','modal');
            end
        end
    case 'Log Normal'
        result = InputIsValid(handles.lpf_text1, 'LPF', '');
        result2 = InputIsValid(handles.lpf_text2, 'LPF', 'Sig');
        if result && result2
            pd =
makedist('Lognormal','mu',log(num1)+num2^2,'sigma',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','drxi', xi);
            assignin('base','drfi2', fi);
```

```
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            axis tight;
            ylabel('Probability Density');
            xlabel('LPF');
            str = sprintf('\\fontsize{12} LPF distribution plot with
Log Normal distribution with Mean=%0.2e , Stdev=%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in lpf_text1, lpf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in lpf_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in lpf_text2, invalid
input.','Invalid Input','modal');
            end
        end
    case 'Beta'
        result = InputIsValid(handles.lpf_text1, 'LPF', 'ab');
        result2 = InputIsValid(handles.lpf_text2, 'LPF', 'ab');
        if result && result2
            pd = makedist('Beta','a',num1,'b',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','lpfxi', xi);
            assignin('base','lpffi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('LPF');
            str = sprintf('\\fontsize{12} LPF distribution plot with
Beta distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                mean(n),std(n));
```

```
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in lpf_text1, lpf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in lpf_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in lpf_text2, invalid
input.','Invalid Input','modal');
            end
        end
    case 'Uniform'
        result = InputIsValid(handles.lpf_text1, 'LPF', '');
        result2 = InputIsValid(handles.lpf_text2, 'LPF', 'LL');
        if result && result2
            if num1 < num2;
                % In unifrom distribution upper limt must be greater
than lower
                % limit, if not show the error message
                errordlg('Upper Limit is less than lower limt','Uniform
Distribution','modal')
                set(handles.lpf_pushbutton,'str','Show
Plot','backg',col);
                return;
            else
                pd = makedist('Uniform','Upper',num1,'Lower',num2);
                t = truncate(pd,0,1);
                n = random(t,samplesize,1);
                axes(handles.axes1)
                nbins = max(min(length(n)./10,100),50);
                xi = linspace(min(n),max(n),nbins);
                dx = mean(diff(xi));
                fi = histc(n,xi-dx);
                fi = fi./sum(fi)./dx;
                assignin('base','lpfxi', xi);
                assignin('base','lpffi2', fi);
                bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6
.6], 'BarWidth',1);
                axis tight;
                %             hist(n,50);

                ylabel('Probability Density');
                xlabel('LPF');
                str = sprintf('\\fontsize{12} LPF distribution plot
with Uniform distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
```

```
                    mean(n),std(n));
                title(str,'Units', 'normalized', ...
                    'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
            end
        else
            if ~result && ~result2
                errordlg('Problem in lpf_text1, lpf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in lpf_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in lpf_text2, invalid
input.','Invalid Input','modal');
            end
        end
    case 'Exponential'
        result = InputIsValid(handles.lpf_text1, 'LPF', '');
        if result
            pd = makedist('Exponential','mu',num1);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','lpfxi', xi);
            assignin('base','lpffi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('LPF');
            str = sprintf('\\fontsize{12} LPF distribution plot with
Exponential distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            errordlg('Problem in lpf_text1, invalid input.','Invalid
Input','modal');
        end
    case 'User Defined'
        [Parameters,X,Y] = Parameters.GetUDD(CurrentMAR,'LPF');
```

```
        n = zeros(1,samplesize);
        for e = 1:samplesize;
            num_rand=rand;
            ter = size(X);
            for i = 1:ter(2)
                iSum = 0;
                for j = 1:i
                    iSum = iSum + Y(j);
                end
                if num_rand < iSum
                    if i == 1
                        n(e) = rand*(X(i+1)-X(i))+X(i);
                    else
                        n(e) = rand*(X(i)-X(i-1))+X(i);
                    end
                    break;
                end
            end
        end
        axes(handles.axes1)
        nbins = max(min(length(n)./10,100),50);
        xi = linspace(min(n),max(n),nbins);
        dx = mean(diff(xi));
        fi = histc(n,xi-dx);
        fi = fi./sum(fi)./dx;
        assignin('base','drxi', xi);
        assignin('base','drfi2', fi);
        bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
        axis tight;
        %         hist(n,50);
        ylabel('Probability Density');
        xlabel('LPF');
        str = sprintf('\\fontsize{12} LPF distribution plot with User
Defined Distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
            mean(n),std(n));
        title(str,'Units', 'normalized', ...
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')
end
set(handles.lpf_pushbutton,'str','Show Plot','backg',col);
end

% --- Executes during object creation, after setting all properties.
function mar_togglebutton_CreateFcn(hObject, eventdata, handles)
% hObject    handle to mar_togglebutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
end
```

```
% --- Executes on button press in lpf_togglebutton.
% Executed when leak path factor toogle button is pressed
function lpf_togglebutton_Callback(hObject, ~, handles)
% hObject    handle to lpf_togglebutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ispushed = get(hObject,'Value');

if ispushed
    set(hObject,'string','Single Input');
    set(handles.lpf_text1,'Enable','on');
    set(handles.lpf_text1,'String','');
    set(handles.lpf_text2,'String','');
    set(handles.lpf_text2,'Enable','off') ; %
    set(handles.lpf_pushbutton,'Enable','off');  %
    set(handles.lpf_popup_dist,'Enable','off');  %
    set(handles.lpf_popup_dist,'Value',1)
else
    set(hObject,'string','Distribution Input');
    set(handles.lpf_text1,'String','');
    set(handles.lpf_text2,'String','');
    set(handles.lpf_text1,'Enable','off');
    set(handles.lpf_text2,'Enable','off');  %
    %     set(handles.dr_pushbutton,'Enable','on');  %

    set(handles.lpf_popup_dist,'Enable','on');  %

end
end
% Hint: get(hObject,'Value') returns toggle state of lpf_togglebutton


% --- Executes on button press in arf_pushbutton.
% Executed when airbone release fraction show plot button is pressed
function arf_pushbutton_Callback(hObject, ~, handles)
% hObject    handle to arf_pushbutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Parameters;
global CurrentMAR;
sample = get(handles.num_sample_text,'String');
samplesize = str2double(sample);
if strcmp(sample,'') == 1 || samplesize < 0
    errordlg('Please enter number of samples','Sample Number','modal');
    return;
end

col = get(handles.arf_pushbutton,'backg');
```

```
set(handles.arf_pushbutton,'str','RUNNING...','backg',[.2 .6 .6]);
pause(eps);
num1 = str2double(get(handles.arf_text1,'String'));
num2 = str2double(get(handles.arf_text2,'String'));
contents = get(handles.arf_popup_dist,'String');
popupmenuvalue = contents{get(handles.arf_popup_dist,'Value')};
cla(handles.axes1,'reset');
switch popupmenuvalue
    case 'Normal'
        result = InputIsValid(handles.arf_text1, 'ARF', '');
        result2 = InputIsValid(handles.arf_text2, 'ARF', 'Sig');
        if result && result2
            pd = makedist('Normal','mu',num1,'sigma',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','arfxi', xi);
            assignin('base','arffi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('ARF');
            str = sprintf('\\fontsize{12} ARF distribution plot with
Normal distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in arf_text1, arf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in arf_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in arf_text2, invalid
input.','Invalid Input','modal');
            end
        end
    case 'Log Normal'
        result = InputIsValid(handles.arf_text1, 'ARF', '');
```

```
        result2 = InputIsValid(handles.arf_text2, 'ARF', 'Sig');
        if result && result2
            pd =
makedist('Lognormal','mu',log(num1)+num2^2,'sigma',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','drxi', xi);
            assignin('base','drfi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            axis tight;
            ylabel('Probability Density');
            xlabel('ARF');
            str = sprintf('\\fontsize{12} ARF distribution plot with
Log Normal distribution with Mean=%0.2e , Stdev=%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in arf_text1, arf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in arf_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in arf_text2, invalid
input.','Invalid Input','modal');
            end
        end

    case 'Beta'
        result = InputIsValid(handles.arf_text1, 'ARF', 'ab');
        result2 = InputIsValid(handles.arf_text2, 'ARF', 'ab');
        if result && result2
            pd = makedist('Beta','a',num1,'b',num2);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
```

```
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','arfxi', xi);
            assignin('base','arffi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('ARF');
            str = sprintf('\\fontsize{12} ARF distribution plot with
Beta distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in arf_text1, arf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in arf_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in arf_text2, invalid
input.','Invalid Input','modal');
            end
        end
    case 'Uniform'
        result = InputIsValid(handles.arf_text1, 'ARF', '');
        result2 = InputIsValid(handles.arf_text2, 'ARF', 'LL');
        if result && result2
            if num1 < num2;
                % In unifrom distribution upper limt must be greater
than lower
                % limit, if not show the error message
                errordlg('Upper Limit is less than lower limt','Uniform
Distribution','modal')
                set(handles.arf_pushbutton,'str','Show
Plot','backg',col);
                return;
            else
                pd = makedist('Uniform','Upper',num1,'Lower',num2);
                t = truncate(pd,0,1);
                n = random(t,samplesize,1);
                axes(handles.axes1)
                nbins = max(min(length(n)./10,100),50);
```

```
                xi = linspace(min(n),max(n),nbins);
                dx = mean(diff(xi));
                fi = histc(n,xi-dx);
                fi = fi./sum(fi)./dx;
                assignin('base','arfxi', xi);
                assignin('base','arffi2', fi);
                bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6
.6], 'BarWidth',1);
                axis tight;
                %             hist(n,50);

                ylabel('Probability Density');
                xlabel('ARF');
                str = sprintf('\\fontsize{12} ARF distribution plot
with Uniform distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                    mean(n),std(n));
                title(str,'Units', 'normalized', ...
                    'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
            end
        else
            if ~result && ~result2
                errordlg('Problem in arf_text1, arf_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in arf_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in arf_text2, invalid
input.','Invalid Input','modal');
            end
        end
    case 'Exponential'
        result = InputIsValid(handles.arf_text1, 'ARF', '');
        if result
            pd = makedist('Exponential','mu',num1);
            t = truncate(pd,0,1);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','arfxi', xi);
            assignin('base','arffi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
```

```
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('ARF');
            str = sprintf('\\fontsize{12} ARF distribution plot with
Exponential distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            errordlg('Problem in arf_text1, invalid input.','Invalid
Input','modal');
        end
    case 'User Defined'
        [Parameters,X,Y] = Parameters.GetUDD(CurrentMAR,'ARF');
        n = zeros(1,samplesize);
        for e = 1:samplesize;
            num_rand=rand;
            ter = size(X);
            for i = 1:ter(2)
                iSum = 0;
                for j = 1:i
                    iSum = iSum + Y(j);
                end
                if num_rand < iSum
                    if i == 1
                        n(e) = rand*(X(i+1)-X(i))+X(i);
                    else
                        n(e) = rand*(X(i)-X(i-1))+X(i);
                    end
                    break;
                end
            end
        end
        axes(handles.axes1)
        nbins = max(min(length(n)./10,100),50);
        xi = linspace(min(n),max(n),nbins);
        dx = mean(diff(xi));
        fi = histc(n,xi-dx);
        fi = fi./sum(fi)./dx;
        assignin('base','drxi', xi);
        assignin('base','drfi2', fi);
        bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
        axis tight;
        %         hist(n,50);
        ylabel('Probability Density');
        xlabel('ARF');
```

```
        str = sprintf('\\fontsize{12} ARF distribution plot with User
Defined Distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
            mean(n),std(n));
        title(str,'Units', 'normalized', ...
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')
end
set(handles.arf_pushbutton,'str','Show Plot','backg',col);
end



function arf_text2_Callback(hObject, ~, handles)
% hObject    handle to arf_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end
% Hints: get(hObject,'String') returns contents of arf_text2 as text
%        str2double(get(hObject,'String')) returns contents of
arf_text2 as a double


% --- Executes during object creation, after setting all properties.
function arf_text2_CreateFcn(hObject, ~, handles)
% hObject    handle to arf_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function arf_text1_Callback(hObject, ~, handles)
% hObject    handle to arf_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end
% Hints: get(hObject,'String') returns contents of arf_text1 as text
%        str2double(get(hObject,'String')) returns contents of
arf_text1 as a double


% --- Executes during object creation, after setting all properties.
function arf_text1_CreateFcn(hObject, ~, handles)
```

```
% hObject    handle to arf_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on selection change in arf_popup_dist.
% Executed when arf distributin is selected
function arf_popup_dist_Callback(hObject, ~, handles)
% hObject    handle to arf_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Parameters;
global CurrentMAR;
contents = cellstr(get(hObject,'String'));
arfpopchoice = contents{get(hObject,'Value')};
switch arfpopchoice

    case 'Normal'
        set(handles.arf_text1,'Enable','inactive')  %
        set(handles.arf_text2,'Enable','inactive')  %
        set(handles.arf_pushbutton,'Enable','on')  %
        set(handles.arf_text1,'String','Mean');
        set(handles.arf_text2,'String','Std Deviation');
        set(handles.arf_text1,'TooltipString','')
        set(handles.arf_text2,'TooltipString','')
    case 'Beta'
        set(handles.arf_text1,'Enable','inactive')  %
        set(handles.arf_text2,'Enable','inactive')  %
        set(handles.arf_pushbutton,'Enable','on')  %
        set(handles.arf_text1,'String','a');
        set(handles.arf_text2,'String','b');
        set(handles.arf_text1,'TooltipString','shape parameter')
        set(handles.arf_text2,'TooltipString','shape parameter')
    case 'Uniform'
        set(handles.arf_text1,'Enable','inactive')  %
        set(handles.arf_text2,'Enable','inactive')  %
        set(handles.arf_pushbutton,'Enable','on')  %
        set(handles.arf_text1,'String','Upper Limit');
        set(handles.arf_text2,'String','Lower Limit');
        set(handles.arf_text1,'TooltipString','')
        set(handles.arf_text2,'TooltipString','')
```

```
    case 'Exponential'
        set(handles.arf_text1,'Enable','inactive')  %
        set(handles.arf_text2,'Enable','off')  %
        set(handles.arf_pushbutton,'Enable','on')  %
        set(handles.arf_text1,'String','Mean');
        set(handles.arf_text2,'String','');
        set(handles.arf_text1,'TooltipString','')
        set(handles.arf_text2,'TooltipString','')
    case 'Select Distribution'
        set(handles.arf_text1,'String','');
        set(handles.arf_text2,'String','');
        set(handles.arf_text1,'Enable','off')  %
        set(handles.arf_text2,'Enable','off')  %
        set(handles.arf_pushbutton,'Enable','off')  %
        set(handles.arf_text1,'TooltipString','')
        set(handles.arf_text2,'TooltipString','')
    case 'Log Normal'
        set(handles.arf_text1,'Enable','inactive')  %
        set(handles.arf_text2,'Enable','inactive')  %
        set(handles.arf_pushbutton,'Enable','on')  %
        set(handles.arf_text1,'String',{'Mode'});
        set(handles.arf_text2,'String',{'Scale Param.'});
        set(handles.arf_text1,'TooltipString','')
        set(handles.arf_text2,'TooltipString','')
    case 'User Defined'
        set(handles.arf_text1,'Enable','off')
        set(handles.arf_text2,'Enable','off')
        set(handles.arf_pushbutton,'Enable','on')
        set(handles.arf_text1,'String','User');
        set(handles.arf_text2,'String','Defined');
        set(handles.arf_text1,'TooltipString','')
        set(handles.arf_text2,'TooltipString','')
        Parameters = UserDefined(Parameters);
        [Parameters, msg, flag] = Parameters.CheckUDD('ARF');
        if flag == 1
            msgbox(msg);
            set(Parameters,'UDtempX',0);
            set(Parameters,'UDtempY',0);
            set(hObject,'Value', 1);
            arf_popup_dist_Callback(hObject, '', handles);
        else
            Parameters = Parameters.SaveUDD(CurrentMAR,'ARF');
        end
end
end
% Hints: contents = cellstr(get(hObject,'String')) returns
arf_popup_dist contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
arf_popup_dist
```

```
% --- Executes during object creation, after setting all properties.
function arf_popup_dist_CreateFcn(hObject, ~, handles)
% hObject    handle to arf_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on button press in arf_togglebutton.
% Ecectued when arf toggle button is pressed
function arf_togglebutton_Callback(hObject, ~, handles)
% hObject    handle to arf_togglebutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ispushed = get(hObject,'Value');

if ispushed
    set(hObject,'string','Single Input');
    set(handles.arf_text1,'Enable','on');
    set(handles.arf_text1,'String','');
    set(handles.arf_text2,'String','');
    set(handles.arf_text2,'Enable','off') ; %
    set(handles.arf_pushbutton,'Enable','off');  %
    set(handles.arf_popup_dist,'Enable','off');  %
    set(handles.arf_popup_dist,'Value',1)
else
    set(hObject,'string','Distribution Input');
    set(handles.arf_text1,'String','');
    set(handles.arf_text2,'String','');
    set(handles.arf_text1,'Enable','off');
    set(handles.arf_text2,'Enable','off');  %
    %     set(handles.dr_pushbutton,'Enable','on');  %

    set(handles.arf_popup_dist,'Enable','on');  %

end
end
% Hint: get(hObject,'Value') returns toggle state of arf_togglebutton
```

```
% --- Executes on selection change in mar_popup_dist.
function mar_popup_dist_Callback(hObject, ~, handles)
% hObject    handle to mar_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
contents = cellstr(get(hObject,'String'));
marpopchoice = contents{get(hObject,'Value')};

switch marpopchoice
    case 'Log Normal'
        set(handles.mar_text1,'Enable','inactive')  %
        set(handles.mar_text2,'Enable','inactive')  %
        set(handles.mar_pushbutton,'Enable','on')  %
        set(handles.mar_text1,'String',{'Mean'});
        set(handles.mar_text2,'String',{'Std Deviation'});
        set(handles.mar_text1,'TooltipString','')
        set(handles.mar_text2,'TooltipString','')
    case 'Normal'
        set(handles.mar_text1,'Enable','inactive')  %
        set(handles.mar_text2,'Enable','inactive')  %
        set(handles.mar_pushbutton,'Enable','on')  %
        set(handles.mar_text1,'String',{'Mean'});
        set(handles.mar_text2,'String',{'Std Deviation'});
        set(handles.mar_text1,'TooltipString','')
        set(handles.mar_text2,'TooltipString','')
    case 'Beta'
        set(handles.mar_text1,'Enable','inactive')  %
        set(handles.mar_text2,'Enable','inactive')  %
        set(handles.mar_pushbutton,'Enable','on')  %
        set(handles.mar_text1,'String','a');
        set(handles.mar_text2,'String','b');
        set(handles.mar_text1,'TooltipString','shape parameter')
        set(handles.mar_text2,'TooltipString','shape parameter')
    case 'Uniform'
        set(handles.mar_text1,'Enable','inactive')  %
        set(handles.mar_text2,'Enable','inactive')  %
        set(handles.mar_pushbutton,'Enable','on')  %
        set(handles.mar_text1,'String','Upper Limit');
        set(handles.mar_text2,'String','Lower Limit');
        set(handles.mar_text1,'TooltipString','')
        set(handles.mar_text2,'TooltipString','')
    case 'Exponential'
        set(handles.mar_text1,'Enable','inactive')  %
        set(handles.mar_text2,'Enable','off')  %
        set(handles.mar_pushbutton,'Enable','on')  %
        set(handles.mar_text1,'String','Mean');
        set(handles.mar_text2,'String','');
        set(handles.mar_text1,'TooltipString','')
        set(handles.mar_text2,'TooltipString','')
```

```
    case 'Select Distribution'
        set(handles.mar_text1,'String','');
        set(handles.mar_text2,'String','');
        set(handles.mar_text1,'Enable','off')  %
        set(handles.mar_text2,'Enable','off')  %
        set(handles.mar_pushbutton,'Enable','off')  %
        set(handles.mar_text1,'TooltipString','')
        set(handles.mar_text2,'TooltipString','')

end



end
% Hints: contents = cellstr(get(hObject,'String')) returns
mar_popup_dist contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
mar_popup_dist


% --- Executes during object creation, after setting all properties.
function mar_popup_dist_CreateFcn(hObject, ~, handles)
% hObject    handle to mar_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function mar_text1_Callback(hObject, ~, handles)
% hObject    handle to mar_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% set(handles.mar_text1,'string',{})

end
% Hints: get(hObject,'String') returns contents of mar_text1 as text
%        str2double(get(hObject,'String')) returns contents of
mar_text1 as a double


% --- Executes during object creation, after setting all properties.
```

```
function mar_text1_CreateFcn(hObject, ~, handles)
% hObject    handle to mar_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function mar_text2_Callback(hObject, ~, handles)
% hObject    handle to mar_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of mar_text2 as text
%        str2double(get(hObject,'String')) returns contents of
mar_text2 as a double
end

% --- Executes during object creation, after setting all properties.
function mar_text2_CreateFcn(hObject, ~, handles)
% hObject    handle to mar_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on button press in mar_pushbutton.
% Excuted when mar show plot button is pressed
function mar_pushbutton_Callback(hObject, ~, handles)
% hObject    handle to mar_pushbutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

sample = get(handles.num_sample_text,'String');
samplesize = str2double(sample);
```

```
if strcmp(sample,'') == 1 || samplesize < 0
    errordlg('Please enter number of samples','Sample Number','modal');
    return;
end

col = get(handles.mar_pushbutton,'backg');
set(handles.mar_pushbutton,'str','RUNNING...','backg',[.2 .6 .6]);
pause(eps);

num1 = str2double(get(handles.mar_text1,'String'));
num2 = str2double(get(handles.mar_text2,'String'));
contents = get(handles.mar_popup_dist,'String');
popupmenuvalue = contents{get(handles.mar_popup_dist,'Value')};
cla(handles.axes1,'reset');
switch popupmenuvalue
    case 'Normal'
        result = InputIsValid(handles.mar_text1, 'MAR', '');
        result2 = InputIsValid(handles.mar_text2, 'MAR', 'Sig');
        if result && result2
            pd = makedist('Normal','mu',num1,'sigma',num2);
            t = truncate(pd,0,inf);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','marxi', xi);
            assignin('base','marfi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('MAR');
            str = sprintf('\\fontsize{12} MAR distribution plot with
Normal distribution with\\mu=%0.2e Bq ,\\sigma =%0.2e Bq',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in mar_text1, mar_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in mar_text1, invalid
input.','Invalid Input','modal');
```

```
            else
                errordlg('Problem in mar_text2, invalid
input.','Invalid Input','modal');
            end
        end

    case 'Log Normal'
        result = InputIsValid(handles.mar_text1, 'MAR', '');
        result2 = InputIsValid(handles.mar_text2, 'MAR', 'Sig');
        if result && result2
            pd = makedist('Lognormal','mu',num1,'sigma',num2);
            t = truncate(pd,0,inf);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','drxi', xi);
            assignin('base','drfi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            axis tight;
            ylabel('Probability Density');
            xlabel('MAR');
            str = sprintf('\\fontsize{12} MAR distribution plot with
Log Normal distribution with \\mu=%0.2e , \\sigma =%0.2e',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in mar_text1, mar_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in mar_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in mar_text2, invalid
input.','Invalid Input','modal');
            end
        end

    case 'Beta'
        result = InputIsValid(handles.mar_text1, 'MAR', 'ab');
```

```
        result2 = InputIsValid(handles.mar_text2, 'MAR', 'ab');
        if result && result2
            pd = makedist('Beta','a',num1,'b',num2);
            t = truncate(pd,0,inf);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','marxi', xi);
            assignin('base','marfi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('MAR');
            str = sprintf('MAR distribution plot with Beta distribution
with\\mu=%0.3e Bq ,\\sigma =%0.3e Bq',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            if ~result && ~result2
                errordlg('Problem in mar_text1, mar_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in mar_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in mar_text2, invalid
input.','Invalid Input','modal');
            end
        end
    case 'Uniform'
        result = InputIsValid(handles.mar_text1, 'MAR', '');
        result2 = InputIsValid(handles.mar_text2, 'MAR', 'LL');
        if result && result2
            if num1 < num2;
                % In uniform distribution upper limt must be greater
than lower
                % limit, if not show the error message
                errordlg('Upper Limit is less than lower limt','Uniform
Distribution','modal')
                set(handles.mar_pushbutton,'str','Show
Plot','backg',col);
```

```
                return;
            else
                pd = makedist('Uniform','Upper',num1,'Lower',num2);
                t = truncate(pd,0,inf);
                n = random(t,samplesize,1);
                axes(handles.axes1)
                nbins = max(min(length(n)./10,100),50);
                xi = linspace(min(n),max(n),nbins);
                dx = mean(diff(xi));
                fi = histc(n,xi-dx);
                fi = fi./sum(fi)./dx;
                assignin('base','marxi', xi);
                assignin('base','marfi2', fi);
                bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6
.6], 'BarWidth',1);
                axis tight;
                %             hist(n,50);

                ylabel('Probability Density');
                xlabel('MAR');
                str = sprintf('MAR distribution plot with Uniform
distribution with\\mu=%0.3e Bq ,\\sigma =%0.3e Bq',...
                    mean(n),std(n));
                title(str,'Units', 'normalized', ...
                    'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
            end
        else
            if ~result && ~result2
                errordlg('Problem in mar_text1, mar_text2, invalid
input.','Invalid Input','modal');
            elseif ~result && result2
                errordlg('Problem in mar_text1, invalid
input.','Invalid Input','modal');
            else
                errordlg('Problem in mar_text2, invalid
input.','Invalid Input','modal');
            end
        end
    case 'Exponential'
        result = InputIsValid(handles.mar_text1, 'MAR', '');
        if result
            pd = makedist('Exponential','mu',num1);
            t = truncate(pd,0,inf);
            n = random(t,samplesize,1);
            axes(handles.axes1)
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
```

```
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','marxi', xi);
            assignin('base','marfi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            %         hist(n,50);
            ylabel('Probability Density');
            xlabel('MAR');
            str = sprintf('MAR distribution plot with Exponential
distribution with\\mu=%0.3e Bq ,\\sigma =%0.3e Bq',...
                mean(n),std(n));
            title(str,'Units', 'normalized', ...
                'Position', [0.5 1.02], 'HorizontalAlignment',
'center')
        else
            errordlg('Problem in mar_text1, invalid input.','Invalid
Input','modal');
        end
end
set(handles.mar_pushbutton,'str','Show Plot','backg',col);
end

% --- Executes on button press in mar_togglebutton.
function mar_togglebutton_Callback(hObject, ~, handles)
% hObject    handle to mar_togglebutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ispushed = get(hObject,'Value');

if ispushed
    %
    set(hObject,'string','Single Input');
    set(handles.mar_text1,'Enable','on');
    set(handles.mar_text1,'String','');
    set(handles.mar_text2,'String','');
    set(handles.mar_text2,'Enable','off') ; %
    set(handles.mar_pushbutton,'Enable','off');  %
    set(handles.mar_popup_dist,'Enable','off');  %
    set(handles.mar_popup_dist,'Value',1)


else


    set(hObject,'string','Distribution Input');
    set(handles.mar_text1,'String','');
    set(handles.mar_text2,'String','');
```

```
    set(handles.mar_text1,'Enable','off');
    set(handles.mar_text2,'Enable','off');  %
    set(handles.mar_popup_dist,'Enable','on');
end
end
% Hint: get(hObject,'Value') returns toggle state of mar_togglebutton


% --------------------------------------------------------------------
function file_menu_Callback(hObject, ~, handles)
% hObject    handle to file_menu (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end

% --------------------------------------------------------------------
function help_menu_Callback(hObject, ~, handles)
% hObject    handle to help_menu (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end

% --------------------------------------------------------------------
function help_running_menu_Callback(hObject, ~, handles)
% hObject    handle to help_running_menu (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%for windows system
if ispc
    filename = 'C:\Program Files\ISU\SODA\application\help.pdf';
    if exist('help.pdf','file') == 2
        winopen('help.pdf');
    else
        winopen(filename);
    end
elseif ismac || isunix %Not tested on Unix or mac, though mac code
worked at a previous time.
    % mac system
    if exist('help.pdf','file') == 2
        system('open help.pdf')
    else
        system('open /Applications/ISU/SODA/application/help.pdf')
    end
end
end

% --------------------------------------------------------------------
function about_menu_Callback(hObject, ~, handles)
```

```
% hObject    handle to about_menu (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
About;

%  msgbox({'Dr. Chad Pope, Idaho State University' ' ' 'Jason Andrus,
Idaho National Lab'...
%     ' ' 'Graduate Student' ' ' 'Kushal Bhattarai, Idaho State
University',...
%     ' ' 'Undergraduate Students' ' ' 'Abdullah Alomani' ' ' 'Abraham
Chupp'...
%     ' ' 'Mason Jaussi'},'About');
end


% --------------------------------------------------------------------
% This fucntion let user load saved *.mat file so that user does not
have
% to type every parameter every single time he want to run SODA
function load_menu_Callback(hObject, ~, handles)
% hObject    handle to load_menu (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Parameters;
[filename,pathname] = uigetfile('*.mat','Load Work Space');
if isequal(filename,0)
    return
end

load(fullfile(pathname,filename),'userinput');
Parameters = userinput.data;

set(handles.num_sample_text,'string',(userinput.num_sample_text));

%load DR
set(handles.dr_togglebutton,'Value',userinput.dr_togglebutton);

if userinput.dr_togglebutton == 0;
    set(handles.dr_togglebutton,'String','Distribution Input');

set(handles.dr_popup_dist,'Enable','on','Value',userinput.dr_popup_dist
);
    if userinput.dr_popup_dist > 1 && userinput.dr_popup_dist < 5 ||
userinput.dr_popup_dist == 6;

set(handles.dr_text1,'Enable','on','String',userinput.dr_text1);

set(handles.dr_text2,'Enable','on','String',userinput.dr_text2);
        set(handles.dr_pushbutton,'Enable','on');
```

```
    elseif userinput.dr_popup_dist == 5;

set(handles.dr_text1,'Enable','on','String',userinput.dr_text1);
        set(handles.dr_pushbutton,'Enable','on');
    elseif userinput.dr_popup_dist == 7
        set(handles.dr_text1,'Enable','off','String','User');
        set(handles.dr_text2,'Enable','off','String','Defined');
        set(handles.dr_pushbutton,'Enable','on');
    end
else
    set(handles.dr_togglebutton,'String','Single Input');
    set(handles.dr_popup_dist,'Enable','off','Value',1)
    set(handles.dr_text1,'Enable','on','String',userinput.dr_text1);
    set(handles.dr_text2,'Enable','off','String','');
    set(handles.dr_pushbutton,'Enable','off');
end

%load LPF
set(handles.lpf_togglebutton,'Value',userinput.lpf_togglebutton);

if userinput.lpf_togglebutton == 0;
    set(handles.lpf_togglebutton,'String','Distribution Input');

set(handles.lpf_popup_dist,'Enable','on','Value',userinput.lpf_popup_di
st);
    if userinput.lpf_popup_dist > 1 && userinput.lpf_popup_dist < 5 ||
userinput.lpf_popup_dist == 6;

set(handles.lpf_text1,'Enable','on','String',userinput.lpf_text1);

set(handles.lpf_text2,'Enable','on','String',userinput.lpf_text2);
        set(handles.lpf_pushbutton,'Enable','on');
    elseif userinput.lpf_popup_dist == 5;

set(handles.lpf_text1,'Enable','on','String',userinput.lpf_text1);
        set(handles.lpf_pushbutton,'Enable','on');
    elseif userinput.lpf_popup_dist == 7
        set(handles.lpf_text1,'Enable','off','String','User');
        set(handles.lpf_text2,'Enable','off','String','Defined');
        set(handles.lpf_pushbutton,'Enable','on');
    end
else
    set(handles.lpf_togglebutton,'String','Single Input');
    set(handles.lpf_popup_dist,'Enable','off','Value',1);
    set(handles.lpf_text2,'Enable','off','String','');
    set(handles.lpf_text1,'Enable','on','String',userinput.lpf_text1);
    set(handles.lpf_pushbutton,'Enable','off');
end
```

```
%load BR
set(handles.br_togglebutton,'Value',userinput.br_togglebutton);

if userinput.br_togglebutton == 0;
    set(handles.br_togglebutton,'String','Distribution Input');
    set(handles.br_pushbutton,'Enable','on');
else
    set(handles.br_togglebutton,'Value',1,'String','Single Input');
    set(handles.br_text1,'Enable','on','String',userinput.br_text1);
    set(handles.br_pushbutton,'Enable','off');
end

%load cq

%
set(handles.cq_togglebutton,'Value',userinput.cq_togglebutton);
if userinput.cq_togglebutton == 0;
    set(handles.cq_togglebutton,'String','Distribution Input');

set(handles.terrain_popup,'Enable','on','Value',userinput.terrain_popup
);
    if userinput.terrain_popup == 2;
        set(handles.stability_popup,'Enable','on','String',{'Select
Stability';'A';'B';...
            'C';'D';'E';'F'},'Value',userinput.stability_popup);
    elseif userinput.terrain_popup == 3;
        set(handles.stability_popup,'Enable','on','String',{'Select
Stability';'A-B';'C';...
            'D';'E-F'},'Value',userinput.stability_popup);
    else
        set(handles.stability_popup,'Enable','off''String',{'Select
Stability'});
    end

set(handles.windspeed_popup_dist,'Enable','on','Value',userinput.windsp
eed_popup_dist);
    set(handles.cq_text1,'Enable','off','String','');
    set(handles.height_text,'String',userinput.height_text);

set(handles.distance_text1,'Enable','on','String',userinput.distance_te
xt1);

set(handles.distance_text2,'Enable','on','String',userinput.distance_te
xt2);
    set(handles.cq_pushbutton,'Enable','off');
    if userinput.windspeed_popup_dist > 1

set(handles.windspeed_text1,'Enable','on','String',userinput.windspeed_
text1);
```

```
set(handles.windspeed_text2,'Enable','on','String',userinput.windspeed_
text2);
        set(handles.cq_pushbutton,'Enable','on');
    end

else
    set(handles.cq_togglebutton,'Value',1,'String','Single Input');
    set(handles.terrain_popup,'Enable','off','Value',1);
    set(handles.stability_popup,'Enable','off','Value',1);
    set(handles.windspeed_popup_dist,'Enable','off','Value',1);
    set(handles.cq_pushbutton,'Enable','off');
    set(handles.cq_text1,'Enable','on','String',userinput.cq_text1);
    set(handles.distance_text1,'Enable','off','String','');
    set(handles.distance_text2,'Enable','off','String','');
    set(handles.height_text,'Enable','off','String','');
    set(handles.windspeed_text1,'Enable','off','String','');
    set(handles.windspeed_text2,'Enable','off','String','');

end
radioMAR4_Callback(handles.radioMAR4, '', handles); %Trick the code
into resetting the load info.
load(fullfile(pathname,filename),'userinput');
Parameters = userinput.data;
radioMAR1_Callback(handles.radioMAR1, '', handles);
load(fullfile(pathname,filename),'userinput');
Parameters = userinput.data;   %The final result is a correct load of
all data.

%The steps above are nessesary due to the fact that the code will erase
the
%loaded data in the currently selected MAR when the MAR state is
changed to
%allow the others to load. Once the others are done, the parameters
data
%that is saved is refresehed to the state that was saved, and the
program
%is ready to be used.
end




% --------------------------------------------------------------------
% This fucntion gather all the user input and saves it in *.mat file so
% that the user can load it in SODA for futur runs.
function save_work_menu_Callback(hObject, ~, handles)
% hObject    handle to save_work_menu (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
global Parameters;
[filename,pathname] = uiputfile('*.mat','Save Work Space As');
if pathname == 0 %if the user pressed cancelled, then we exit this
callback
    return
end

userinput.num_sample_text =
str2double(get(handles.num_sample_text,'String'));

userinput.dr_togglebutton = get(handles.dr_togglebutton,'Value');
userinput.dr_popup_dist = get(handles.dr_popup_dist,'Value');
userinput.dr_text1 = str2double(get(handles.dr_text1,'String'));
userinput.dr_text2 = str2double(get(handles.dr_text2,'String'));

userinput.lpf_togglebutton = get(handles.lpf_togglebutton,'Value');
userinput.lpf_popup_dist = get(handles.lpf_popup_dist,'Value');
userinput.lpf_text1 = str2double(get(handles.lpf_text1,'String'));
userinput.lpf_text2 = str2double(get(handles.lpf_text2,'String'));

userinput.br_togglebutton = get(handles.br_togglebutton,'Value');
userinput.br_text1 = str2double(get(handles.br_text1,'String'));

userinput.cq_togglebutton = get(handles.cq_togglebutton,'Value');
userinput.distance_text1 =
str2double(get(handles.distance_text1,'String'));
userinput.distance_text2 =
str2double(get(handles.distance_text2,'String'));
userinput.terrain_popup = get(handles.terrain_popup,'Value');
userinput.stability_popup = get(handles.stability_popup,'Value');
userinput.windspeed_popup_dist =
get(handles.windspeed_popup_dist,'Value');
userinput.cq_text1= str2double(get(handles.cq_text1,'String'));

userinput.windspeed_text1=
str2double(get(handles.windspeed_text1,'String'));
userinput.windspeed_text2=
str2double(get(handles.windspeed_text2,'String'));
userinput.height_text = str2double(get(handles.height_text,'String'));

SaveMARSpecificData(handles);
userinput.data = Parameters;

save(fullfile(pathname,filename),'userinput')  %This may fail if too
much data is saved in the class object.
end
```

```
% --------------------------------------------------------------------
function save_image_menu_Callback(hObject, ~, handles)
% hObject    handle to save_image_menu (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



[filename,pathname] = uiputfile('*.jpg;*.png;*.tif','Save as');
if pathname == 0 %if the user pressed cancelled, then we exit this
callback
    return
end
haxes=handles.axes1;
ftmp = figure('visible','off');
set(ftmp,'Position',[0 0 1024 576]);
new_axes = copyobj(haxes, ftmp);
set(new_axes,'fontsize',10);
set(new_axes,'Units','normalized','Position',[0.06 0.12 0.90 0.80]);
saveas(ftmp, fullfile(pathname,filename));
delete(ftmp);
end



% --------------------------------------------------------------------
% when user wants to exit from exit menu
function exit_menu_Callback(hObject, ~, handles)
% hObject    handle to exit_menu (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
exit_button = questdlg('Exit Now?','Exit SODA','Yes','No','No');
switch exit_button;
    case 'Yes'
        delete(handles.SodaMain);
    case 'No'
        return
end
end

% --- Executes when mar_uipanel is resized.
function mar_uipanel_ResizeFcn(hObject, ~, handles)
% hObject    handle to mar_uipanel (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end

% --- Executes on button press in cq_togglebutton.
function togglebutton9_Callback(hObject, ~, handles)
```

```
% hObject    handle to cq_togglebutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of cq_togglebutton
end

% --- Executes on button press in cq_pushbutton.
function pushbutton10_Callback(hObject, ~, handles)
% hObject    handle to cq_pushbutton (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end


function cq_text_Callback(hObject, ~, handles)
% hObject    handle to cq_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cq_text1 as text
%        str2double(get(hObject,'String')) returns contents of cq_text1
as a double
end

% --- Executes during object creation, after setting all properties.
function cq_text_CreateFcn(hObject, ~, handles)
% hObject    handle to cq_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function distance_text2_Callback(hObject, ~, handles)
% hObject    handle to distance_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of distance_text2 as
text
```

```
%        str2double(get(hObject,'String')) returns contents of
distance_text2 as a double
end

% --- Executes during object creation, after setting all properties.
function distance_text2_CreateFcn(hObject, ~, handles)
% hObject    handle to distance_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function distance_text1_Callback(hObject, ~, handles)
% hObject    handle to distance_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of distance_text1 as
text
%        str2double(get(hObject,'String')) returns contents of
distance_text1 as a double
end

% --- Executes during object creation, after setting all properties.
function distance_text1_CreateFcn(hObject, ~, handles)
% hObject    handle to distance_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on selection change in distance_popup_dist.
% Used by CHi/Q calucation for downwind distance.
```

```
% THis fuction is excuted when user selected downwind distance
distribution
function distance_popup_dist_Callback(hObject, ~, handles)
% hObject    handle to distance_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
contents = cellstr(get(hObject,'String'));
distance_popchoice = contents{get(hObject,'Value')};
switch distance_popchoice
    case 'Normal'
        set(handles.distance_text1,'Enable','inactive')  %
        set(handles.distance_text2,'Enable','inactive')  %
        set(handles.distance_text1,'String','Mean');
        set(handles.distance_text2,'String','Std Deviation');
        set(handles.cq_pushbutton,'Enable','off')
        set(handles.distance_text1,'TooltipString','')
        set(handles.distance_text2,'TooltipString','')
    case 'Beta'
        set(handles.distance_text1,'Enable','inactive')  %
        set(handles.distance_text2,'Enable','inactive')  %
        set(handles.distance_text1,'String','a');
        set(handles.distance_text2,'String','b');
        set(handles.distance_text1,'TooltipString','shape parameter')
        set(handles.distance_text2,'TooltipString','shape parameter')
        set(handles.cq_pushbutton,'Enable','off')
    case 'Uniform'
        set(handles.distance_text1,'Enable','inactive')  %
        set(handles.distance_text2,'Enable','inactive')  %
        set(handles.distance_text1,'String','Upper Limit');
        set(handles.distance_text2,'String','Lower Limit');
        set(handles.cq_pushbutton,'Enable','off')
        set(handles.distance_text1,'TooltipString','')
        set(handles.distance_text2,'TooltipString','')
    case 'Exponential'
        set(handles.distance_text1,'Enable','inactive')  %
        set(handles.distance_text2,'Enable','off')  %
        set(handles.distance_text1,'String','Mean');
        set(handles.cq_pushbutton,'Enable','off')
        set(handles.distance_text2,'String','');
        set(handles.distance_text1,'TooltipString','')
        set(handles.distance_text2,'TooltipString','')
    case 'Select Distribution'
        set(handles.distance_text1,'String','');
        set(handles.distance_text2,'String','');
        set(handles.distance_text1,'Enable','off')  %
        set(handles.distance_text2,'Enable','off')  %
        set(handles.cq_pushbutton,'Enable','off')  %
        set(handles.distance_text1,'TooltipString','')
        set(handles.distance_text2,'TooltipString','')
```

```
end
end
% Hints: contents = cellstr(get(hObject,'String')) returns
distance_popup_dist contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
distance_popup_dist


% --- Executes during object creation, after setting all properties.
function distance_popup_dist_CreateFcn(hObject, ~, handles)
% hObject    handle to distance_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function height_text_Callback(hObject, ~, handles)
% hObject    handle to height_text (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of height_text as text
%        str2double(get(hObject,'String')) returns contents of
height_text as a double
end

% --- Executes during object creation, after setting all properties.
function height_text_CreateFcn(hObject, ~, handles)
% hObject    handle to height_text (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end
```

```
% --- Executes on selection change in stability_popup.
function stability_popup_Callback(hObject, ~, handles)
% hObject    handle to stability_popup (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
stability_popup contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
stability_popup
end

% --- Executes during object creation, after setting all properties.
function stability_popup_CreateFcn(hObject, ~, handles)
% hObject    handle to stability_popup (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on selection change in terrain_popup.
% Executed when terrain for CHi/Q is selected
function terrain_popup_Callback(hObject, ~, handles)
% hObject    handle to terrain_popup (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
contents = cellstr(get(hObject,'String'));
terrain_popchoice = contents{get(hObject,'Value')};
switch terrain_popchoice
    case 'Urban Area'
        set(handles.stability_popup,'Enable','on')
        set(handles.stability_popup,'String',{'Select Stability';'A-
B';'C';...
            'D';'E-F'},'Value', 1);
    case 'Rural/Open Country'
        set(handles.stability_popup,'Enable','on')
        set(handles.stability_popup,'String',{'Select
Stability';'A';'B';...
            'C';'D';'E';'F'},'Value', 1);
```

```
    case 'Select Terrain'
        set(handles.stability_popup,'Value', 1,'String','Select
Stability','Enable','off');
end
end

% Hints: contents = cellstr(get(hObject,'String')) returns
terrain_popup contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
terrain_popup


% --- Executes during object creation, after setting all properties.
function terrain_popup_CreateFcn(hObject, ~, handles)
% hObject    handle to terrain_popup (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes when SodaMain is resized.
function figure1_ResizeFcn(hObject, ~, handles)
% hObject    handle to SodaMain (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end


function windspeed_text2_Callback(hObject, ~, handles)
% hObject    handle to windspeed_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of windspeed_text2 as
text
%        str2double(get(hObject,'String')) returns contents of
windspeed_text2 as a double
end

% --- Executes during object creation, after setting all properties.
function windspeed_text2_CreateFcn(hObject, ~, handles)
% hObject    handle to windspeed_text2 (see GCBO)
```

```
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

function windspeed_text1_Callback(hObject, ~, handles)
% hObject    handle to windspeed_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of windspeed_text1 as
text
%        str2double(get(hObject,'String')) returns contents of
windspeed_text1 as a double
end

% --- Executes during object creation, after setting all properties.
function windspeed_text1_CreateFcn(hObject, ~, handles)
% hObject    handle to windspeed_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on selection change in windspeed_popup_dist.
% for chi/q, when user selected a wind speed distribution
function windspeed_popup_dist_Callback(hObject, ~, handles)
% hObject    handle to windspeed_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
windspeed_popup_dist contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
windspeed_popup_dist
```

```
contents = cellstr(get(hObject,'String'));
windspeed_popchoice = contents{get(hObject,'Value')};
switch windspeed_popchoice
    case 'Normal'
        set(handles.windspeed_text1,'Enable','inactive')  %
        set(handles.windspeed_text2,'Enable','inactive')  %
        set(handles.windspeed_text1,'String','Mean');
        set(handles.windspeed_text2,'String','Std Deviation');
        set(handles.cq_pushbutton,'Enable','on')
        set(handles.windspeed_text1,'TooltipString','')
        set(handles.windspeed_text2,'TooltipString','')
    case 'Beta'
        set(handles.windspeed_text1,'Enable','inactive')  %
        set(handles.windspeed_text2,'Enable','inactive')  %
        set(handles.windspeed_text1,'String','a');
        set(handles.windspeed_text2,'String','b');
        set(handles.windspeed_text1,'TooltipString','shape parameter')
        set(handles.windspeed_text2,'TooltipString','shape parameter')
        set(handles.cq_pushbutton,'Enable','on')
    case 'Uniform'
        set(handles.windspeed_text1,'Enable','inactive')  %
        set(handles.windspeed_text2,'Enable','inactive')  %
        set(handles.windspeed_text1,'String','Upper Limit');
        set(handles.windspeed_text2,'String','Lower Limit');
        set(handles.cq_pushbutton,'Enable','on')
        set(handles.windspeed_text1,'TooltipString','')
        set(handles.windspeed_text2,'TooltipString','')
    case 'Exponential'
        set(handles.windspeed_text1,'Enable','inactive')  %
        set(handles.windspeed_text2,'Enable','off')  %
        set(handles.windspeed_text1,'String','Mean');
        set(handles.cq_pushbutton,'Enable','on')
        set(handles.windspeed_text2,'String','');
        set(handles.windspeed_text1,'TooltipString','')
        set(handles.windspeed_text2,'TooltipString','')
    case 'Select Distribution'
        set(handles.windspeed_text1,'String','');
        set(handles.windspeed_text2,'String','');
        set(handles.windspeed_text1,'Enable','off')  %
        set(handles.windspeed_text2,'Enable','off')  %
        set(handles.cq_pushbutton,'Enable','off')  %
        set(handles.windspeed_text1,'TooltipString','')
        set(handles.windspeed_text2,'TooltipString','')
end
end


% --- Executes during object creation, after setting all properties.
function windspeed_popup_dist_CreateFcn(hObject, ~, handles)
```

```
% hObject    handle to windspeed_popup_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes when user attempts to close SodaMain.
function SodaMain_CloseRequestFcn(hObject, ~, handles)
% hObject    handle to SodaMain (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
exit_button = questdlg('Exit Now?','Exit SODA','Yes','No','Yes');
switch exit_button;
    case 'Yes'
        delete(hObject);
    case 'No'
        return
end
end

% --------------------------------------------------------------------
function random_gen_Callback(hObject, ~, handles)
% hObject    handle to random_gen (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
rng('default');
msgbox('Random  Number  Generator has been reset','Reset');
end

% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
mar_text1.
function mar_text1_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to mar_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% set(hObject,'String','','Enable','on')
set(hObject,'Enable','on');
set(handles.mar_text1,'string',[]);
end
```

```
% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
mar_text2.
function mar_text2_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to mar_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'Enable','on');
set(handles.mar_text2,'string',[]);
end

% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
dr_text1.
function dr_text1_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to dr_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'Enable','on');
set(handles.dr_text1,'string',[]);
end

% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
dr_text2.
function dr_text2_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to dr_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'Enable','on');
set(handles.dr_text2,'string',[]);
end

% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
arf_text1.
function arf_text1_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to arf_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'Enable','on');
set(handles.arf_text1,'string',[]);
end
% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
arf_text2.
function arf_text2_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to arf_text2 (see GCBO)
```

```
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'Enable','on');
set(handles.arf_text2,'string',[]);
end

% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
rf_text1.
function rf_text1_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to rf_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'Enable','on');
set(handles.rf_text1,'string',[]);
end

% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
rf_text2.
function rf_text2_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to rf_text2 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'Enable','on');
set(handles.rf_text2,'string',[]);
end

% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
lpf_text1.
function lpf_text1_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to lpf_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'Enable','on');
set(handles.lpf_text1,'string',[]);
end
% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
lpf_text1.
function lpf_text2_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to lpf_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'Enable','on');
set(handles.lpf_text2,'string',[]);
end
```

```
% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
height_text.
function height_text_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to height_text (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'Enable','on');
set(handles.height_text,'string',[]);
end

% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
windspeed_text1.
function windspeed_text1_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to windspeed_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'Enable','on');
set(handles.windspeed_text1,'string',[]);
end
% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
windspeed_text1.
function windspeed_text2_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to windspeed_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'Enable','on');
set(handles.windspeed_text2,'string',[]);
end

% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
dcf_text1.
function dcf_text1_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to dcf_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'Enable','on');
set(handles.dcf_text1,'string',[]);
end
% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
dcf_text1.
function dcf_text2_ButtonDownFcn(hObject, ~, handles)
% hObject    handle to dcf_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
set(hObject,'Enable','on');
set(handles.dcf_text2,'string',[]);
end
% --- Executes during object deletion, before destroying properties.
function SodaMain_DeleteFcn(hObject, ~, handles)
% hObject    handle to SodaMain (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end

% --- Executes during object deletion, before destroying properties.
function mar_text1_DeleteFcn(hObject, ~, handles)
% hObject    handle to mar_text1 (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end

% --- Executes on button press in fit_dist.
% Used to find best fit for the distribution
function fit_dist_Callback(hObject, ~, handles)
% hObject    handle to fit_dist (see GCBO)
% ~  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ced = getappdata(0,'ced');
col = get(handles.fit_dist,'backg');
set(handles.fit_dist,'str','RUNNING...','backg',[.2 .6 .6]);
pause(eps);
[~,PD]=allfitdistBICPDF(ced,handles);
set(handles.fit_dist,'str','Fit Distribution','backg',col);
% assignin('base','PD', PD);
switch PD{1, 1}.DistributionName
    case 'Generalized Extreme Value'
        msgbox({PD{1, 1}.DistributionName ['k =' num2str(PD{1,
1}.k)]...
            ['Mean= ' num2str(PD{1, 1}.mu)] ['Sigma =' num2str(PD{1,
1}.sigma)]},'Best Fit','modal')
    case 'Inverse Gaussian'
        msgbox({PD{1, 1}.DistributionName ['Mean =' num2str(PD{1,
1}.mu)]...
            ['Lambda= ' num2str(PD{1, 1}.lambda)]},'Best Fit','modal')
    case 'Lognormal'
        msgbox({PD{1, 1}.DistributionName ['Mean =' num2str(PD{1,
1}.mu)]...
            ['Sigma= ' num2str(PD{1, 1}.sigma)] },'Best Fit','modal')
    case 'Log-Logistic'
        msgbox({PD{1, 1}.DistributionName ['Mean =' num2str(PD{1,
1}.mu)]...
            ['Sigma= ' num2str(PD{1, 1}.sigma)] },'Best Fit','modal')
    case 't Location-Scale'
```

```
        msgbox({PD{1, 1}.DistributionName ['Mean =' num2str(PD{1,
1}.mu)]...
            ['Sigma= ' num2str(PD{1, 1}.sigma)] ['Nu= ' num2str(PD{1,
1}.nu)]},'Best Fit','modal')
    case 'Gamma'
        msgbox({PD{1, 1}.DistributionName ['a =' num2str(PD{1,
1}.a)]...
            ['b= ' num2str(PD{1, 1}.b)] },'Best Fit','modal')
    case 'Beta'
        msgbox({PD{1, 1}.DistributionName ['a =' num2str(PD{1,
1}.a)]...
            ['b= ' num2str(PD{1, 1}.b)] },'Best Fit','modal')
    case 'Weibull'
        msgbox({PD{1, 1}.DistributionName ['A =' num2str(PD{1,
1}.A)]...
            ['B= ' num2str(PD{1, 1}.B)] },'Best Fit','modal')
    case 'Generalized Pareto'
        msgbox({PD{1, 1}.DistributionName ['k =' num2str(PD{1,
1}.k)]...
            ['Sigma= ' num2str(PD{1, 1}.sigma)] ['Theta= '
num2str(PD{1, 1}.theta)]},'Best Fit','modal')
    case 'Exponential'
        msgbox({PD{1, 1}.DistributionName ['Mean =' num2str(PD{1,
1}.mu)]},'Best Fit','modal')
    case 'Rayleigh'
        msgbox({PD{1, 1}.DistributionName ['B =' num2str(PD{1,
1}.B)]},'Best Fit','modal')
    case 'Logistic'
        msgbox({PD{1, 1}.DistributionName ['Mean =' num2str(PD{1,
1}.mu)]...
            ['Sigma= ' num2str(PD{1, 1}.sigma)] },'Best Fit','modal')
    case 'Normal'
        msgbox({PD{1, 1}.DistributionName ['Mean =' num2str(PD{1,
1}.mu)]...
            ['Sigma= ' num2str(PD{1, 1}.sigma)] },'Best Fit','modal')
    case 'Extreme Value'
        msgbox({PD{1, 1}.DistributionName ['Mean =' num2str(PD{1,
1}.mu)]...
            ['Sigma= ' num2str(PD{1, 1}.sigma)] },'Best Fit','modal')
end


end



% This function is used to find best fit distribution for the CED data
function [D, PD] = allfitdistBICPDF(data,handles)
```

```
%ALLFITDIST Fit all valid parametric probability distributions to data.
%   [D PD] = ALLFITDIST(DATA) fits all valid parametric probability
%   distributions to the data in vector DATA by BIC method, and returns
%   a struct D of fitted distributions and parameters and a struct of
%   objects PD representing the fitted distributions. PD is an object
%   in a class derived from the ProbDist class.
%
%   [...] = ALLFITDIST(...,'PDF') or (...,'CDF') plots either the PDF
or CDF
%   of a subset of the fitted distribution. The distributions are
plotted in
%   order of fit, according to SORTBY.
%
%   List of distributions it will try to fit
%       Beta
%       Exponential
%       Gamma
%       Inverse Gaussian
%       Logistic
%       Log-logistic
%       Lognormal
%       Normal
%
%   EXAMPLE 1
%     Given random data from an unknown continuous distribution, find
the
%     best distribution which fits that data, and plot the PDFs to
compare
%     graphically.
%        data = normrnd(5,3,1e4,1);         %Assumed from unknown
distribution
%        [D PD] = allfitdist(data,'PDF');   %Compute and plot results
%        D(1)                               %Show output from best fit
%

%    Mike Sheppard
%    Last Modified: 17-Feb-2012
%    Arr. Steffanie Nestor
%    Last Modified: 31-Mar-2015
%    Arr. Kushal Bhattarai
%    Last Modified: 04-02-2015



%% Check Inputs
vin={'pdf'};

distname={'beta',  'exponential', ...
    'extreme value', 'gamma', 'generalized extreme value', ...
```

```
    'inversegaussian', 'logistic', 'loglogistic', ...
    'lognormal', 'normal','rayleigh', 'tlocationscale', 'weibull'};

vin(1)=[];
n=numel(data); %Number of data points
data = data(:);
D=[];

%% Run through all distributions in FITDIST function
warning('off','all'); %Turn off all future warnings
for indx=1:length(distname)
    try
        dname=distname{indx};
        PD = fitdist(data,dname,vin{:});

        NLL=PD.NLogL; % -Log(L)
        %If NLL is non-finite number, produce error to ignore
distribution
        if ~isfinite(NLL)
            error('non-finite NLL');
        end
        num=length(D)+1;
        PDs(num) = {PD}; %#ok<*AGROW>
        k=numel(PD.Params); %Number of parameters
        % assigns response to return/plot variable
        D(num).DistName=PD.DistName;
        D(num).BIC=-2*(-NLL)+k*log(n);
        D(num).ParamNames=PD.ParamNames;
        D(num).ParamDescription=PD.ParamDescription;
        D(num).Params=PD.Params;
        D(num).Paramci=PD.paramci;
        D(num).ParamCov=PD.ParamCov;
        D(num).Support=PD.Support;
    catch err %#ok<NASGU>
        %Ignore distribution
    end
end
warning('on','all'); %Turn back on warnings
if numel(D)==0
    errordlg('No distributions were found','Error');
    return;
end


%% Sort distributions
% prepares distribution fits according to BIC best fit to data
indx1=1:length(D); %Identity Map
[~,indx1]=sort([D.BIC]);
D=D(indx1); PD = PDs(indx1);
```

```
% Plot
plotfigs(data,D,PD,handles);

end



function plotfigs(data,D,PD,handles)
%Plot functionality for continuous case due to Jonathan Sullivan
%Modified by author for discrete case

%Maximum number of distributions to include
%max_num_dist=Inf;  %All valid distributions
max_num_dist=4;

cla(handles.axes1,'reset');
axes(handles.axes1);

%% Probability Density / Mass Plot


%Continuous Data

nbins = max(min(length(data)./10,100),50);
xi = linspace(min(data),max(data),nbins);
dx = mean(diff(xi));
xi2 = linspace(min(data),max(data),nbins*10)';
fi = histc(data,xi-dx);
fi = fi./sum(fi)./dx;
assignin('base','fitxi', xi);
assignin('base','fitfi2', fi);
inds = 1:min([max_num_dist,numel(PD)]);
ys = cellfun(@(PD) pdf(PD,xi2),PD(inds),'UniformOutput',0);
ys = cat(2,ys{:});
[r_gen,x_gen] = ksdensity(data);
plot(x_gen,r_gen,'LineWidth',3,'color','k');
%         hold on;
%         bar(xi,fi,'FaceColor','m','EdgeColor','m','BarWidth', 1);
hold on;
plot(xi2,ys,'LineWidth',1.5)
axis tight;
legend(['Random Generated',{D(inds).DistName}],'Location','NE');
xlabel('Commited Effective Dose (rem)');
ylabel('Probability Density');
title(['Probability Density Function with \mu =' num2str(mean(data)) '
\sigma =' num2str(std(data))]);
grid on;
```

```
end


% --- Executes on key press with focus on mar_text1 and none of its
controls.
function mar_text1_KeyPressFcn(hObject, ~, handles)
% hObject    handle to mar_text1 (see GCBO)
% ~  structure with the following fields (see UICONTROL)
%   Key: name of the key that was pressed, in lower case
%   Character: character interpretation of the key(s) that was pressed
%   Modifier: name(s) of the modifier key(s) (i.e., control, shift)
pressed
% handles    structure with handles and user data (see GUIDATA)
end

% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
mar_pushbutton.
function mar_pushbutton_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to mar_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end


% --- Executes when SodaMain is resized.
function SodaMain_SizeChangedFcn(hObject, eventdata, handles)
% hObject    handle to SodaMain (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end

% --- Executes on button press in radioMAR1.
function radioMAR1_Callback(hObject, eventdata, handles)
% hObject    handle to radioMAR1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radioMAR1
ChangeMARState(hObject, handles);
end

% --- Executes on button press in radioMAR2.
function radioMAR2_Callback(hObject, eventdata, handles)
% hObject    handle to radioMAR2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of radioMAR2
ChangeMARState(hObject, handles);
end

% --- Executes on button press in radioMAR3.
function radioMAR3_Callback(hObject, eventdata, handles)
% hObject    handle to radioMAR3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radioMAR3
ChangeMARState(hObject, handles);
end

% --- Executes on button press in radioMAR4.
function radioMAR4_Callback(hObject, eventdata, handles)
% hObject    handle to radioMAR4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radioMAR4
ChangeMARState(hObject, handles);
end


% --- Executes on button press in MARbtn.
function MARbtn_Callback(hObject, eventdata, handles)
% hObject    handle to MARbtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Parameters
global CurrentMAR
Parameters = MAR_Selection(Parameters);  %Let the user select their MAR
information.
CurrentMAR = GetCurrentMAR();
if CurrentMAR ~= 0
    if get(handles.dcf_togglebutton, 'Value') %check if DCF is set to
single value.
        %Set MAR boxes and DCF boxes after recieving output from
MAR_Selection.
        if Parameters.MAR(CurrentMAR)~= 0
            set(handles.mar_text1, 'String',
Parameters.MAR(CurrentMAR));
        else
            set(handles.mar_text1, 'String', '');
        end
        if Parameters.DCF(CurrentMAR)~= 0
```

```
            set(handles.dcf_text1, 'String',
Parameters.DCF(CurrentMAR));
        else
            set(handles.dcf_text1, 'String', '');
        end

set(handles.dcf_popup_dist,'String',{Parameters.Isotope{CurrentMAR},'Se
lect Isotope'}...
            ,'Value',1,'Enable','on');
    else  %if DCF is set to distribution input...
        contents = get(handles.dcf_popup_dist,'String');
        popupmenuvalue = contents{get(handles.dcf_popup_dist,'Value')};
        if strcmp(popupmenuvalue, 'User Defined')
            msgbox('User Defined Distribution is Selected, and will
override the DCF setting saved in MAR Selection.');
        else
            if Parameters.MAR(CurrentMAR)~= 0
                set(handles.mar_text1, 'String',
Parameters.MAR(CurrentMAR));
            else
                set(handles.mar_text1, 'String', '');
            end
            if Parameters.DCF(CurrentMAR)~= 0
                set(handles.dcf_text1, 'String',
Parameters.DCF(CurrentMAR));
            else
                set(handles.dcf_text1, 'String', '');
            end
        end
    end
else
    msgbox('MAR State Exclusivity Error; SODA will close.','Fatal
Error')
    delete(handles.Soda_Main);
end
end

%*********************************************

function ChangeMARState(hObject, handles)
global CurrentMAR
global Parameters

MARStr = get(hObject, 'String');
SizeCheck = size(MARStr);

%If this assertion is thrown, check Radiobutton that is clicked for
having
%extra lines in its String property.
```

```
assert(SizeCheck(1) == 1,'MARStr does not have size 1xX, Property
Error.')

MARStr = MARStr(5); %Get MAR Number from String
MARNum = str2double(MARStr);

result = SaveMARSpecificData(handles);
if result > 0
    msgbox('Some entered data was invalid, and not saved. Please return
to previous MAR selection and input valid values. Show All will only
use complete entries in its calculation.','Invalid Property')
end

CurrentMAR = MARNum; %Set new CurrentMAR after saving old data.

if strcmp(Parameters.MARdist{CurrentMAR},'1') ||
strcmp(Parameters.MARdist{CurrentMAR},'')
    if ~get(handles.mar_togglebutton,'Value')
        set(handles.mar_togglebutton,'Value',1);
        mar_togglebutton_Callback(handles.mar_togglebutton, '',
handles);
    end
else
    if get(handles.mar_togglebutton,'Value')
        set(handles.mar_togglebutton,'Value',0);
        mar_togglebutton_Callback(handles.mar_togglebutton, '',
handles);
    end
    contents = cellstr(get(handles.mar_popup_dist,'String'));
        for i=1:size(contents)
            if strcmp(contents{i},Parameters.MARdist{CurrentMAR})
                set(handles.mar_popup_dist,'Value',i);

mar_popup_dist_Callback(handles.mar_popup_dist,'',handles);
                break;
            end
        end
end

if strcmp(Parameters.ARFdist{CurrentMAR},'1')
    if ~get(handles.arf_togglebutton,'Value')
        set(handles.arf_togglebutton,'Value',1);
        arf_togglebutton_Callback(handles.arf_togglebutton, '',
handles);
    end
elseif strcmp(Parameters.ARFdist{CurrentMAR},'')
    set(handles.arf_togglebutton,'Value',1);
    arf_togglebutton_Callback(handles.arf_togglebutton, '', handles);
    set(handles.arf_togglebutton,'Value',0);
```

```
    arf_togglebutton_Callback(handles.arf_togglebutton, '', handles);
else
    if get(handles.arf_togglebutton,'Value')
        set(handles.arf_togglebutton,'Value',0);
        arf_togglebutton_Callback(handles.arf_togglebutton, '',
handles);
    end
    contents = cellstr(get(handles.arf_popup_dist,'String'));
        for i=1:size(contents)
            if strcmp(contents{i},Parameters.ARFdist{CurrentMAR})
                if strcmp(Parameters.ARFdist{CurrentMAR},'User
Defined')
                    set(handles.arf_popup_dist,'Value',i);
                    set(handles.arf_text1,'String','User');
                    set(handles.arf_text2,'String','Defined');
                    set(handles.arf_text1,'Enable','off');
                    set(handles.arf_text2,'Enable','off');
                    set(handles.arf_pushbutton,'Enable','on');
                else
                    set(handles.arf_popup_dist,'Value',i);

arf_popup_dist_Callback(handles.arf_popup_dist,'',handles);
                end
                break;
            end
        end
end

if strcmp(Parameters.RFdist{CurrentMAR},'1')
    if ~get(handles.rf_togglebutton,'Value')
        set(handles.rf_togglebutton,'Value',1);
        rf_togglebutton_Callback(handles.rf_togglebutton, '', handles);
    end
elseif strcmp(Parameters.RFdist{CurrentMAR},'')
    set(handles.rf_togglebutton,'Value',1);
    rf_togglebutton_Callback(handles.rf_togglebutton, '', handles);
    set(handles.rf_togglebutton,'Value',0);
    rf_togglebutton_Callback(handles.rf_togglebutton, '', handles);
else
    if get(handles.rf_togglebutton,'Value')
        set(handles.rf_togglebutton,'Value',0);
        rf_togglebutton_Callback(handles.rf_togglebutton, '', handles);
    end
    contents = cellstr(get(handles.rf_popup_dist,'String'));
        for i=1:size(contents)
            if strcmp(contents{i},Parameters.RFdist{CurrentMAR})
                if strcmp(Parameters.RFdist{CurrentMAR},'User Defined')
                    set(handles.rf_popup_dist,'Value',i);
                    set(handles.rf_text1,'String','User');
```

```
                    set(handles.rf_text2,'String','Defined');
                    set(handles.rf_text1,'Enable','off');
                    set(handles.rf_text2,'Enable','off');
                    set(handles.rf_pushbutton,'Enable','on');
                else
                    set(handles.rf_popup_dist,'Value',i);

rf_popup_dist_Callback(handles.rf_popup_dist,'',handles);
                end
                break;
            end
        end
end

if strcmp(Parameters.DCFdist{CurrentMAR},'1') ||
strcmp(Parameters.DCFdist{CurrentMAR},'')
    if ~get(handles.dcf_togglebutton,'Value')
        set(handles.dcf_togglebutton,'Value',1);
        dcf_togglebutton_Callback(handles.dcf_togglebutton, '',
handles);
    end
else
    if get(handles.dcf_togglebutton,'Value')
        set(handles.dcf_togglebutton,'Value',0);
        dcf_togglebutton_Callback(handles.dcf_togglebutton, '',
handles);
    end
    contents = cellstr(get(handles.dcf_popup_dist,'String'));
    for i=1:size(contents)
        if strcmp(contents{i},Parameters.DCFdist{CurrentMAR})
            if strcmp(Parameters.DCFdist{CurrentMAR},'User Defined')
                set(handles.dcf_popup_dist,'Value',i);
                set(handles.dcf_text1,'String','User');
                set(handles.dcf_text2,'String','Defined');
                set(handles.dcf_text1,'Enable','off');
                set(handles.dcf_text2,'Enable','off');
                set(handles.dcf_pushbutton,'Enable','on');
            else
                set(handles.dcf_popup_dist,'Value',i);

dcf_popup_dist_Callback(handles.dcf_popup_dist,'',handles);
            end
            break;
        end
    end
end

if Parameters.MAR(CurrentMAR)~= 0
    set(handles.mar_text1, 'String', Parameters.MAR(CurrentMAR));
```

```
else
    if get(handles.mar_togglebutton, 'Value')
        set(handles.mar_text1, 'String', '');
    else
        contents = cellstr(get(handles.mar_popup_dist,'String'));
        marpopchoice = contents{get(handles.mar_popup_dist,'Value')};

        switch marpopchoice
            case 'Normal'
                %if normal is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.mar_text1,'String','Mean');
                set(handles.mar_text1,'TooltipString','')
            case 'Log Normal'
                %if log normal is selected enable input text box and
also display parameter
                %required in those text box.
                set(handles.mar_text1,'String','Mode');
                set(handles.mar_text1,'TooltipString','')
            case 'Beta'
                %if Beta is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.mar_text1,'String','a');
                set(handles.mar_text1,'TooltipString','shape
parameter')
            case 'Uniform'
                %if Uniform is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.mar_text1,'String','Upper Limit');
                set(handles.mar_text1,'TooltipString','')
            case 'Exponential'
                %if Exponential is selected enable input text box and
also display parameter
                %required in those text box.
                set(handles.mar_text1,'String','Mean');
                set(handles.mar_text1,'TooltipString','')
            case 'Select Distribution'
                %if Select distribution is selected disable input text
box and also
                %disable show plot button.
                set(handles.mar_text1,'String','');
                set(handles.mar_text1,'TooltipString','')
        end
    end
end
if Parameters.MAR2(CurrentMAR)~= 0
```

```
    set(handles.mar_text2, 'String', Parameters.MAR2(CurrentMAR));
else
    if get(handles.mar_togglebutton, 'Value')
        set(handles.mar_text2, 'String', '');
    else
        contents = cellstr(get(handles.mar_popup_dist,'String'));
        marpopchoice = contents{get(handles.mar_popup_dist,'Value')};

        switch marpopchoice
            case 'Normal'
                %if normal is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.mar_text2,'String','Std Deviation');
                set(handles.mar_text2,'TooltipString','')
            case 'Log Normal'
                %if log normal is selected enable input text box and
also display parameter
                %required in those text box.
                set(handles.mar_text2,'String','Scale Param.');
                set(handles.mar_text2,'TooltipString','')
            case 'Beta'
                %if Beta is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.mar_text2,'String','b');
                set(handles.mar_text2,'TooltipString','shape
parameter')
            case 'Uniform'
                %if Uniform is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.mar_text2,'String','Lower Limit');
                set(handles.mar_text2,'TooltipString','')
            case 'Exponential'
                %if Exponential is selected enable input text box and
also display parameter
                %required in those text box.
                set(handles.mar_text2,'String','');
                set(handles.mar_text2,'TooltipString','')
            case 'Select Distribution'
                %if Select distribution is selected disable input text
box and also
                %disable show plot button.
                set(handles.mar_text2,'String','');
                set(handles.mar_text2,'TooltipString','')
        end
    end
end
```

```
if Parameters.DCF(CurrentMAR)~= 0
    if get(handles.dcf_togglebutton,'Value')

set(handles.dcf_popup_dist,'String',{Parameters.Isotope{CurrentMAR},'Se
lect Isotope'}...
        ,'Value',1,'Enable','on');
    end
    set(handles.dcf_text1, 'String', Parameters.DCF(CurrentMAR));
else
    if get(handles.dcf_togglebutton, 'Value')
        set(handles.dcf_text1, 'String', '');
        set(handles.dcf_popup_dist,'String',{'Select Isotope'}...
        ,'Value',1,'Enable','on');
    else
        contents = cellstr(get(handles.dcf_popup_dist,'String'));
        dcfpopchoice = contents{get(handles.dcf_popup_dist,'Value')};

        switch dcfpopchoice
            case 'Normal'
                %if normal is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.dcf_text1,'String','Mean');
                set(handles.dcf_text1,'TooltipString','')
            case 'Log Normal'
                %if log normal is selected enable input text box and
also display parameter
                %required in those text box.
                set(handles.dcf_text1,'String','Mode');
                set(handles.dcf_text1,'TooltipString','')
            case 'Beta'
                %if Beta is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.dcf_text1,'String','a');
                set(handles.dcf_text1,'TooltipString','shape
parameter')
            case 'Uniform'
                %if Uniform is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.dcf_text1,'String','Upper Limit');
                set(handles.dcf_text1,'TooltipString','')
            case 'Exponential'
                %if Exponential is selected enable input text box and
also display parameter
                %required in those text box.
                set(handles.dcf_text1,'String','Mean');
                set(handles.dcf_text1,'TooltipString','')
```

```
            case 'Select Distribution'
                %if Select distribution is selected disable input text
box and also
                %disable show plot button.
                set(handles.dcf_text1,'String','');
                set(handles.dcf_text1,'TooltipString','')
        end
    end
end
if Parameters.DCF2(CurrentMAR)~= 0
    set(handles.dcf_text2, 'String', Parameters.DCF2(CurrentMAR));
else
    if get(handles.dcf_togglebutton, 'Value')
        set(handles.dcf_text2, 'String', '');
    else
        contents = cellstr(get(handles.dcf_popup_dist,'String'));
        dcfpopchoice = contents{get(handles.dcf_popup_dist,'Value')};

        switch dcfpopchoice
            case 'Normal'
                %if normal is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.dcf_text2,'String','Std Deviation');
                set(handles.dcf_text2,'TooltipString','')
            case 'Log Normal'
                %if log normal is selected enable input text box and
also display parameter
                %required in those text box.
                set(handles.dcf_text2,'String','Scale Param.');
                set(handles.dcf_text2,'TooltipString','')
            case 'Beta'
                %if Beta is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.dcf_text2,'String','b');
                set(handles.dcf_text2,'TooltipString','shape
parameter')
            case 'Uniform'
                %if Uniform is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.dcf_text2,'String','Lower Limit');
                set(handles.dcf_text2,'TooltipString','')
            case 'Exponential'
                %if Exponential is selected enable input text box and
also display parameter
                %required in those text box.
                set(handles.dcf_text2,'String','');
```

```
                set(handles.dcf_text2,'TooltipString','')
            case 'Select Distribution'
                %if Select distribution is selected disable input text
box and also
                %disable show plot button.
                set(handles.dcf_text2,'String','');
                set(handles.dcf_text2,'TooltipString','')
        end
    end
end
if Parameters.ARF(CurrentMAR)~= 0
    set(handles.arf_text1, 'String', Parameters.ARF(CurrentMAR));
else
    if get(handles.arf_togglebutton, 'Value')
        set(handles.arf_text1, 'String', '');
    else
        contents = cellstr(get(handles.arf_popup_dist,'String'));
        arfpopchoice = contents{get(handles.arf_popup_dist,'Value')};

        switch arfpopchoice
            case 'Normal'
                %if normal is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.arf_text1,'String','Mean');
                set(handles.arf_text1,'TooltipString','')
            case 'Log Normal'
                %if log normal is selected enable input text box and
also display parameter
                %required in those text box.
                set(handles.arf_text1,'String','Mode');
                set(handles.arf_text1,'TooltipString','')
            case 'Beta'
                %if Beta is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.arf_text1,'String','a');
                set(handles.arf_text1,'TooltipString','shape
parameter')
            case 'Uniform'
                %if Uniform is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.arf_text1,'String','Upper Limit');
                set(handles.arf_text1,'TooltipString','')
            case 'Exponential'
                %if Exponential is selected enable input text box and
also display parameter
                %required in those text box.
```

```
                set(handles.arf_text1,'String','Mean');
                set(handles.arf_text1,'TooltipString','')
            case 'Select Distribution'
                %if Select distribution is selected disable input text
box and also
                %disable show plot button.
                set(handles.arf_text1,'String','');
                set(handles.arf_text1,'TooltipString','')
        end
    end
end
if Parameters.ARF2(CurrentMAR)~= 0
    set(handles.arf_text2, 'String', Parameters.ARF2(CurrentMAR));
else
    if get(handles.arf_togglebutton, 'Value')
        set(handles.arf_text2, 'String', '');
    else
        contents = cellstr(get(handles.arf_popup_dist,'String'));
        arfpopchoice = contents{get(handles.arf_popup_dist,'Value')};

        switch arfpopchoice
            case 'Normal'
                %if normal is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.arf_text2,'String','Std Deviation');
                set(handles.arf_text2,'TooltipString','')
            case 'Log Normal'
                %if log normal is selected enable input text box and
also display parameter
                %required in those text box.
                set(handles.arf_text2,'String','Scale Param.');
                set(handles.arf_text2,'TooltipString','')
            case 'Beta'
                %if Beta is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.arf_text2,'String','b');
                set(handles.arf_text2,'TooltipString','shape
parameter')
            case 'Uniform'
                %if Uniform is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.arf_text2,'String','Lower Limit');
                set(handles.arf_text2,'TooltipString','')
            case 'Exponential'
                %if Exponential is selected enable input text box and
also display parameter
```

```
                %required in those text box.
                set(handles.arf_text2,'String','');
                set(handles.arf_text2,'TooltipString','')
            case 'Select Distribution'
                %if Select distribution is selected disable input text
box and also
                %disable show plot button.
                set(handles.arf_text2,'String','');
                set(handles.arf_text2,'TooltipString','')
        end
    end
end
if Parameters.RF(CurrentMAR)~= 0
    set(handles.rf_text1, 'String', Parameters.RF(CurrentMAR));
else
    if get(handles.rf_togglebutton, 'Value')
        set(handles.rf_text1, 'String', '');
    else
        contents = cellstr(get(handles.rf_popup_dist,'String'));
        rfpopchoice = contents{get(handles.rf_popup_dist,'Value')};

        switch rfpopchoice
            case 'Normal'
                %if normal is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.rf_text1,'String','Mean');
                set(handles.rf_text1,'TooltipString','')
            case 'Log Normal'
                %if log normal is selected enable input text box and
also display parameter
                %required in those text box.
                set(handles.rf_text1,'String','Mode');
                set(handles.rf_text1,'TooltipString','')
            case 'Beta'
                %if Beta is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.rf_text1,'String','a');
                set(handles.rf_text1,'TooltipString','shape parameter')
            case 'Uniform'
                %if Uniform is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.rf_text1,'String','Upper Limit');
                set(handles.rf_text1,'TooltipString','')
            case 'Exponential'
                %if Exponential is selected enable input text box and
also display parameter
```

```
                %required in those text box.
                set(handles.rf_text1,'String','Mean');
                set(handles.rf_text1,'TooltipString','')
            case 'Select Distribution'
                %if Select distribution is selected disable input text
box and also
                %disable show plot button.
                set(handles.rf_text1,'String','');
                set(handles.rf_text1,'TooltipString','')
        end
    end
end
if Parameters.RF2(CurrentMAR)~= 0
    set(handles.rf_text2, 'String', Parameters.RF2(CurrentMAR));
else
    if get(handles.rf_togglebutton, 'Value')
        set(handles.rf_text2, 'String', '');
    else
        contents = cellstr(get(handles.rf_popup_dist,'String'));
        rfpopchoice = contents{get(handles.rf_popup_dist,'Value')};

        switch rfpopchoice
            case 'Normal'
                %if normal is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.rf_text2,'String','Std Deviation');
                set(handles.rf_text2,'TooltipString','')
            case 'Log Normal'
                %if log normal is selected enable input text box and
also display parameter
                %required in those text box.
                set(handles.rf_text2,'String','Scale Param.');
                set(handles.rf_text2,'TooltipString','')
            case 'Beta'
                %if Beta is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.rf_text2,'String','b');
                set(handles.rf_text2,'TooltipString','shape parameter')
            case 'Uniform'
                %if Uniform is selected enable input text box and also
display parameter
                %required in those text box.
                set(handles.rf_text2,'String','Lower Limit');
                set(handles.rf_text2,'TooltipString','')
            case 'Exponential'
                %if Exponential is selected enable input text box and
also display parameter
```

```
                %required in those text box.
                set(handles.rf_text2,'String','');
                set(handles.rf_text2,'TooltipString','')
            case 'Select Distribution'
                %if Select distribution is selected disable input text
box and also
                %disable show plot button.
                set(handles.rf_text2,'String','');
                set(handles.rf_text2,'TooltipString','')
        end
    end
end
set(handles.runall_pushbutton, 'Enable','on');
end

function result = SaveMARSpecificData(handles) %Save the entered data
to the obj
global Parameters
global CurrentMAR

TempArray = Parameters.MAR; %recall existing saved data
if strcmp(num2str(str2double(get(handles.mar_text1, 'String'))), 'NaN')
== 0 %Check that value is numeric
    if str2double(get(handles.mar_text1, 'String')) ~= 0  %Check if
value is non-zero
        TempArray(CurrentMAR) = str2double(get(handles.mar_text1,
'String'));
    else
        TempArray(CurrentMAR) = 0;
    end
    result = 0;
elseif ~strcmp(get(handles.mar_text1,'String'),'')  %Check if textbox
is empty
    if ~CheckAcceptableText(get(handles.mar_text1,'String')) %Check if
text is a preloaded entry, such as mean etc
        result = 1;
    else
        result = 0;
    end
else
    result = 0;
end
Parameters.MAR = TempArray;

TempArray = Parameters.MARdist;
if get(handles.mar_togglebutton,'Value') == 0  %Save Currently selected
dist, or single value
    contents = get(handles.mar_popup_dist,'String');
    popupmenuvalue = contents{get(handles.mar_popup_dist,'Value')};
```

```
    switch popupmenuvalue
        case 'Select Distribution'
            TempArray{CurrentMAR} = 'Select Distribution';
        case 'Normal'
            TempArray{CurrentMAR} = 'Normal';
        case 'Beta'
            TempArray{CurrentMAR} = 'Beta';
        case 'Uniform'
            TempArray{CurrentMAR} = 'Uniform';
        case 'Exponential'
            TempArray{CurrentMAR} = 'Exponential';
        case 'Log Normal'
            TempArray{CurrentMAR} = 'Log Normal';
        case 'User Defined'
            TempArray{CurrentMAR} = 'User Defined';
    end
else
    TempArray{CurrentMAR} = '1';
end
Parameters.MARdist = TempArray;

TempArray = Parameters.MAR2;
if ~strcmp(num2str(str2double(get(handles.mar_text2, 'String'))),
'NaN')
    if ~get(handles.mar_togglebutton, 'Value')
        if str2double(get(handles.mar_text2, 'String')) ~= 0
            TempArray(CurrentMAR) = str2double(get(handles.mar_text2,
'String'));
        else
            TempArray(CurrentMAR) = 0;
        end
    else
        TempArray(CurrentMAR) = 0;
    end
elseif ~strcmp(get(handles.mar_text2,'String'),'')
    if ~get(handles.mar_togglebutton, 'Value')
        if ~CheckAcceptableText(get(handles.mar_text2,'String'))
            result = result+1;
        end
    else
        TempArray(CurrentMAR) = 0;
    end
end
Parameters.MAR2 = TempArray;

TempArray = Parameters.DCF;
if ~strcmp(num2str(str2double(get(handles.dcf_text1, 'String'))),
'NaN')
    if str2double(get(handles.dcf_text1, 'String')) ~= 0
```

```
        TempArray(CurrentMAR) = str2double(get(handles.dcf_text1,
'String'));
    else
        TempArray(CurrentMAR) = 0;
    end
elseif ~strcmp(get(handles.dcf_text1,'String'),'')
    if ~CheckAcceptableText(get(handles.dcf_text1,'String'))
        result = result+1;
    end
end
Parameters.DCF = TempArray;

TempArray = Parameters.DCFdist;
if get(handles.dcf_togglebutton,'Value') == 0  %Save Currently selected
dist, or single value
    contents = get(handles.dcf_popup_dist,'String');
    popupmenuvalue = contents{get(handles.dcf_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Select Distribution'
            TempArray{CurrentMAR} = 'Select Distribution';
        case 'Normal'
            TempArray{CurrentMAR} = 'Normal';
        case 'Beta'
            TempArray{CurrentMAR} = 'Beta';
        case 'Uniform'
            TempArray{CurrentMAR} = 'Uniform';
        case 'Exponential'
            TempArray{CurrentMAR} = 'Exponential';
        case 'Log Normal'
            TempArray{CurrentMAR} = 'Log Normal';
        case 'User Defined'
            TempArray{CurrentMAR} = 'User Defined';
    end
else
    TempArray{CurrentMAR} = '1';
end
Parameters.DCFdist = TempArray;

TempArray = Parameters.DCF2;
if ~strcmp(num2str(str2double(get(handles.dcf_text2, 'String'))),
'NaN')
    if ~get(handles.dcf_togglebutton, 'Value')
        if str2double(get(handles.dcf_text2, 'String')) ~= 0
            TempArray(CurrentMAR) = str2double(get(handles.dcf_text2,
'String'));
        else
            TempArray(CurrentMAR) = 0;
        end
    else
```

```
        TempArray(CurrentMAR) = 0;
    end
elseif ~strcmp(get(handles.dcf_text2,'String'),'')
    if ~get(handles.dcf_togglebutton, 'Value')
        if ~CheckAcceptableText(get(handles.dcf_text2,'String'))
            result = result+1;
        end
    else
        TempArray(CurrentMAR) = 0;
    end
end
Parameters.DCF2 = TempArray;

TempArray = Parameters.ARF;
if ~strcmp(num2str(str2double(get(handles.arf_text1, 'String'))),
'NaN')
    if str2double(get(handles.arf_text1, 'String')) ~= 0
        TempArray(CurrentMAR) = str2double(get(handles.arf_text1,
'String'));
    else
        TempArray(CurrentMAR) = 0;
    end
elseif ~strcmp(get(handles.arf_text1,'String'),'')
    if ~CheckAcceptableText(get(handles.arf_text1,'String'))
        result = result+1;
    end
end
Parameters.ARF = TempArray;

TempArray = Parameters.ARFdist;
if get(handles.arf_togglebutton,'Value') == 0  %Save Currently selected
dist, or single value
    contents = get(handles.arf_popup_dist,'String');
    popupmenuvalue = contents{get(handles.arf_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Select Distribution'
            TempArray{CurrentMAR} = 'Select Distribution';
        case 'Normal'
            TempArray{CurrentMAR} = 'Normal';
        case 'Beta'
            TempArray{CurrentMAR} = 'Beta';
        case 'Uniform'
            TempArray{CurrentMAR} = 'Uniform';
        case 'Exponential'
            TempArray{CurrentMAR} = 'Exponential';
        case 'Log Normal'
            TempArray{CurrentMAR} = 'Log Normal';
        case 'User Defined'
            TempArray{CurrentMAR} = 'User Defined';
```

```
    end
else
    TempArray{CurrentMAR} = '1';
end
Parameters.ARFdist = TempArray;

TempArray = Parameters.ARF2;
if ~strcmp(num2str(str2double(get(handles.arf_text2, 'String'))),
'NaN')
    if ~get(handles.arf_togglebutton, 'Value')
        if str2double(get(handles.arf_text2, 'String')) ~= 0
            TempArray(CurrentMAR) = str2double(get(handles.arf_text2,
'String'));
        else
            TempArray(CurrentMAR) = 0;
        end
    else
        TempArray(CurrentMAR) = 0;
    end
elseif ~strcmp(get(handles.arf_text2,'String'),'')
    if ~get(handles.arf_togglebutton, 'Value')
        if ~CheckAcceptableText(get(handles.arf_text2,'String'))
            result = result+1;
        end
    else
        TempArray(CurrentMAR) = 0;
    end
end
Parameters.ARF2 = TempArray;

TempArray = Parameters.RF;
if ~strcmp(num2str(str2double(get(handles.rf_text1, 'String'))), 'NaN')
    if str2double(get(handles.rf_text1, 'String')) ~= 0
        TempArray(CurrentMAR) = str2double(get(handles.rf_text1,
'String'));
    else
        TempArray(CurrentMAR) = 0;
    end
elseif ~strcmp(get(handles.rf_text1,'String'),'')
    if ~CheckAcceptableText(get(handles.rf_text1,'String'))
        result = result+1;
    end
end
Parameters.RF = TempArray;

TempArray = Parameters.RFdist;
if get(handles.rf_togglebutton,'Value') == 0  %Save Currently selected
dist, or single value
    contents = get(handles.rf_popup_dist,'String');
```

```
    popupmenuvalue = contents{get(handles.rf_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Select Distribution'
            TempArray{CurrentMAR} = 'Select Distribution';
        case 'Normal'
            TempArray{CurrentMAR} = 'Normal';
        case 'Beta'
            TempArray{CurrentMAR} = 'Beta';
        case 'Uniform'
            TempArray{CurrentMAR} = 'Uniform';
        case 'Exponential'
            TempArray{CurrentMAR} = 'Exponential';
        case 'Log Normal'
            TempArray{CurrentMAR} = 'Log Normal';
        case 'User Defined'
            TempArray{CurrentMAR} = 'User Defined';
    end
else
    TempArray{CurrentMAR} = '1';
end
Parameters.RFdist = TempArray;

TempArray = Parameters.RF2;
if ~strcmp(num2str(str2double(get(handles.rf_text2, 'String'))), 'NaN')
    if ~get(handles.rf_togglebutton, 'Value')
        if str2double(get(handles.rf_text2, 'String')) ~= 0
            TempArray(CurrentMAR) = str2double(get(handles.rf_text2,
'String'));
        else
            TempArray(CurrentMAR) = 0;
        end
    else
        TempArray(CurrentMAR) = 0;
    end
elseif ~strcmp(get(handles.rf_text2,'String'),'') %if field is not
empty
    if ~get(handles.rf_togglebutton, 'Value')
        if ~CheckAcceptableText(get(handles.rf_text2,'String')) %check
for our
            result = result+1;                                    %text.
        end
    else
        TempArray(CurrentMAR) = 0;
    end
end
Parameters.RF2 = TempArray;


end
```

```
function result = CheckAcceptableText(str) %Check for Mean, Std, etc.
%If non numeric input is acceptable, returns true. When we put text
into
%boxes, we do not want an error message to the user.

if strcmp(str,'Mean')
    result = 1;
elseif strcmp(str,'Std Deviation')
    result = 1;
elseif strcmp(str,'a')
    result = 1;
elseif strcmp(str,'b')
    result = 1;
elseif strcmp(str,'Upper Limit')
    result = 1;
elseif strcmp(str,'Lower Limit')
    result = 1;
elseif strcmp(str,'Mode')
    result = 1;
elseif strcmp(str,'Scale Param.')
    result = 1;
elseif strcmp(str,'User')
    result = 1;
elseif strcmp(str,'Defined')
    result = 1;
else
    result = 0;
end

end

function result = CheckSamples(handles)  %Warn the user if they select
too many samples.
    samples = str2double(get(handles.num_sample_text,'string'));
    if samples >= 1e10
        errordlg('Sample count too high, would result in extreme memory
requirement.','Excessive Sample Count');
        result = 0;
    else
        if samples >= 1e8
            str = ['For sample counts in excess of 1E8, a system memory
size'...
                ' of at least 12GB may be required. This requirement is
somewhat'...
                ' lessened if single value inputs are used. Do you wish
to proceed?'];
            YesNo = questdlg(str,'Sample Size Warning');
            switch YesNo
```

```
                case 'Yes'
                    result = 1;
                case 'No'
                    result = 0;
                case 'Cancel'
                    result = 0;
                case ''
                    result = 0;
            end
        else
            result = 1;
        end
    end
end

function result = CheckInput(hObject) %Check function for inputs, see
table below
    Input = get(hObject, 'String');
    Value = str2double(Input);
    if ((Value > 0) && (Value <= 1))
        if Value <= 1e-3
            result = 11; %result of 11 implies numeric input between 0
and 1e-3
        else
            result = 1;  %result of 1 implies numeric input between 0
and 1
        end
    elseif Value == 0
        result = 0;  %result of 0 implies input is 0.
    elseif isnan(Value)
        result = -1; %result of -1 implies non-numeric input.
    elseif Value > 1 && Value ~= inf
        result = 2;  %result of 2 implies numeric input between 1 and
inf (not inclusive)
    elseif Value == inf || Value < 0
        result = -2;  % result of -2 implies numeric input not within
acceptable region of any parameter.
    end
end

%Check input returns a flag which tells the InputIsValid function
%what type of input was recieved.
% result = 0 => Input is 0.
% result = 1 => Numeric between 0 and 1
% result = 2 => non inf numeric greater than 1
% result = -1 => Non numeric
% result = -2 => Invalid numeric for any param.
% result = 11 => Valid for DCF
```

```
function result = InputIsValid(hObject, Param, specVal)
    res1 = CheckInput(hObject);
    if strcmp(specVal, '')
        if strcmp(Param,'MAR')
            if res1 == 1 || res1 == 11 || res1 == 2
                result = 1;
            else
                result = 0;
            end
        elseif strcmp(Param,'DCF')
            if res1 == 11
                result = 1;
            else
                result = 0;
            end
        else
            if res1 == 1 || res1 == 11
                result = 1;
            else
                result = 0;
            end
        end
    elseif strcmp(specVal,'Sig')
        if res1 == 1 || res1 == 11 || res1 == 2
            result = 1;
        else
            result = 0;
        end
    elseif strcmp(specVal,'LL')
        if res1 == 1 || res1 == 11 || res1 == 0
            result = 1;
        else
            result = 0;
        end
    elseif strcmp(specVal,'ab') %ab checks for beta dist case.
        if res1 == 1 || res1 == 11 || res1 == 2
            result = 1;
        else
            result = 0;
        end
    else
        if res1 == 1 || res1 == 11
            result = 1;
        else
            result = 0;
        end
    end
end
```

```
%specVal specifies what the box 2 input is, if any. Options are ''
(empty
%string) for the box 1 case, 'Sig' for sigma or scale parameter, ''LL'
for
%lower limit in the uniform dist, and 'b' for the beta dist case.

function result = MARxisValid(handles) %check all entries in a MAR for
validity
if ~get(handles.mar_togglebutton,'Value')
    contents = get(handles.mar_popup_dist,'String');
    popupmenuvalue = contents{get(handles.mar_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Select Distribution'
            r1 = 0;
            r2 = 0;
        case 'Normal'
            r1 = InputIsValid(handles.mar_text1, 'MAR', '');
            r2 = InputIsValid(handles.mar_text2, 'MAR', 'Sig');
        case 'Uniform'
            r1 = InputIsValid(handles.mar_text1, 'MAR', '');
            r2 = InputIsValid(handles.mar_text2, 'MAR', 'LL');
        case 'Exponential'
            r1 = InputIsValid(handles.mar_text1, 'MAR', '');
            r2 = 1;
    end
else
    r1 = InputIsValid(handles.mar_text1, 'MAR', '');
    r2 = 1;
end
if ~get(handles.dr_togglebutton,'Value')
    contents = get(handles.dr_popup_dist,'String');
    popupmenuvalue = contents{get(handles.dr_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Select Distribution'
            r3 = 0;
            r4 = 0;
        case 'Normal'
            r3 = InputIsValid(handles.dr_text1, 'DR', '');
            r4 = InputIsValid(handles.dr_text2, 'DR', 'Sig');
        case 'Uniform'
            r3 = InputIsValid(handles.dr_text1, 'DR', '');
            r4 = InputIsValid(handles.dr_text2, 'DR', 'LL');
        case 'Exponential'
            r3 = InputIsValid(handles.dr_text1, 'DR', '');
            r4 = 1;
        case 'Log Normal'
            r3 = InputIsValid(handles.dr_text1, 'DR', '');
            r4 = InputIsValid(handles.dr_text2, 'DR', 'Sig');
        case 'Beta'
```

```
            r3 = InputIsValid(handles.dr_text1, 'DR', 'ab');
            r4 = InputIsValid(handles.dr_text2, 'DR', 'ab');
        case 'User Defined'
            r3 = 1;
            r4 = 1;
    end
else
    r3 = InputIsValid(handles.dr_text1, 'DR', '');
    r4 = 1;
end
if ~get(handles.arf_togglebutton,'Value')
    contents = get(handles.arf_popup_dist,'String');
    popupmenuvalue = contents{get(handles.arf_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Select Distribution'
            r5 = 0;
            r6 = 0;
        case 'Normal'
            r5 = InputIsValid(handles.arf_text1, 'ARF', '');
            r6 = InputIsValid(handles.arf_text2, 'ARF', 'Sig');
        case 'Uniform'
            r5 = InputIsValid(handles.arf_text1, 'ARF', '');
            r6 = InputIsValid(handles.arf_text2, 'ARF', 'LL');
        case 'Exponential'
            r5 = InputIsValid(handles.arf_text1, 'ARF', '');
            r6 = 1;
        case 'Log Normal'
            r5 = InputIsValid(handles.arf_text1, 'ARF', '');
            r6 = InputIsValid(handles.arf_text2, 'ARF', 'Sig');
        case 'Beta'
            r5 = InputIsValid(handles.arf_text1, 'ARF', 'ab');
            r6 = InputIsValid(handles.arf_text2, 'ARF', 'ab');
        case 'User Defined'
            r5 = 1;
            r6 = 1;
    end
else
    r5 = InputIsValid(handles.arf_text1, 'ARF', '');
    r6 = 1;
end
if ~get(handles.rf_togglebutton,'Value')
    contents = get(handles.rf_popup_dist,'String');
    popupmenuvalue = contents{get(handles.rf_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Select Distribution'
            r7 = 0;
            r8 = 0;
        case 'Normal'
            r7 = InputIsValid(handles.rf_text1, 'RF', '');
```

```
            r8 = InputIsValid(handles.rf_text2, 'RF', 'Sig');
        case 'Uniform'
            r7 = InputIsValid(handles.rf_text1, 'RF', '');
            r8 = InputIsValid(handles.rf_text2, 'RF', 'LL');
        case 'Exponential'
            r7 = InputIsValid(handles.rf_text1, 'RF', '');
            r8 = 1;
        case 'Log Normal'
            r7 = InputIsValid(handles.rf_text1, 'RF', '');
            r8 = InputIsValid(handles.rf_text2, 'RF', 'Sig');
        case 'Beta'
            r7 = InputIsValid(handles.rf_text1, 'RF', 'ab');
            r8 = InputIsValid(handles.rf_text2, 'RF', 'ab');
        case 'User Defined'
            r7 = 1;
            r8 = 1;
    end
else
    r7 = InputIsValid(handles.rf_text1, 'RF', '');
    r8 = 1;
end
if ~get(handles.lpf_togglebutton,'Value')
    contents = get(handles.lpf_popup_dist,'String');
    popupmenuvalue = contents{get(handles.lpf_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Select Distribution'
            r9 = 0;
            r10 = 0;
        case 'Normal'
            r9 = InputIsValid(handles.lpf_text1, 'LPF', '');
            r10 = InputIsValid(handles.lpf_text2, 'LPF', 'Sig');
        case 'Uniform'
            r9 = InputIsValid(handles.lpf_text1, 'LPF', '');
            r10 = InputIsValid(handles.lpf_text2, 'LPF', 'LL');
        case 'Exponential'
            r9 = InputIsValid(handles.lpf_text1, 'LPF', '');
            r10 = 1;
        case 'Log Normal'
            r9 = InputIsValid(handles.lpf_text1, 'LPF', '');
            r10 = InputIsValid(handles.lpf_text2, 'LPF', 'Sig');
        case 'Beta'
            r9 = InputIsValid(handles.lpf_text1, 'LPF', 'ab');
            r10 = InputIsValid(handles.lpf_text2, 'LPF', 'ab');
        case 'User Defined'
            r9 = 1;
            r10 = 1;
    end
else
    r9 = InputIsValid(handles.lpf_text1, 'LPF', '');
```

```
    r10 = 1;
end
if ~get(handles.dcf_togglebutton,'Value')
    contents = get(handles.dcf_popup_dist,'String');
    popupmenuvalue = contents{get(handles.dcf_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Select Distribution'
            r11 = 0;
            r12 = 0;
        case 'Normal'
            r11 = InputIsValid(handles.dcf_text1, 'DCF', '');
            r12 = InputIsValid(handles.dcf_text2, 'DCF', 'Sig');
        case 'Uniform'
            r11 = InputIsValid(handles.dcf_text1, 'DCF', '');
            r12 = InputIsValid(handles.dcf_text2, 'DCF', 'LL');
        case 'Exponential'
            r11 = InputIsValid(handles.dcf_text1, 'DCF', '');
            r12 = 1;
        case 'Log Normal'
            r11 = InputIsValid(handles.dcf_text1, 'DCF', '');
            r12 = InputIsValid(handles.dcf_text2, 'DCF', 'Sig');
        case 'Beta'
            r11 = InputIsValid(handles.dcf_text1, 'DCF', 'ab');
            r12 = InputIsValid(handles.dcf_text2, 'DCF', 'ab');
        case 'User Defined'
            r11 = 1;
            r12 = 1;
    end
else
    r11 = InputIsValid(handles.dcf_text1, 'DCF', '');
    r12 = 1;
end
if all([r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12])
    result = 1;
else
    result = 0;
end
end

function GetResults(handles)    %Get CED distribution for currently
selected MAR.
global Parameters;
global CurrentMAR;
sample = get(handles.num_sample_text,'String');
samplesize = str2double(sample);
col = get(handles.run_pushbutton,'backg');
set(handles.run_pushbutton,'str','RUNNING...','backg',[.2 .6 .6]);
pause(eps);
a = get(handles.mar_togglebutton,'Value');
```

```
b = get(handles.dr_togglebutton,'Value');
c = get(handles.arf_togglebutton,'Value');
d = get(handles.rf_togglebutton,'Value');
e = get(handles.lpf_togglebutton,'Value');
f = get(handles.br_togglebutton,'Value');
g = get(handles.dcf_togglebutton,'Value');
h = get(handles.cq_togglebutton,'Value');
if a == 0 || b == 0 || c == 0 || d == 0 || e == 0 || f == 0 || g == 0
|| h == 0; %If any parameter has distribution input selected, check
samples
    if strcmp(sample,'') == 1 || samplesize < 0
        waitfor(errordlg('Please enter number of samples','Sample
Number','modal'));
        set(handles.run_pushbutton,'str','Show Plot','backg',col);
        return;
    end
end

%compute material at risk

if a == 0 ;
    num1 = str2double(get(handles.mar_text1,'String'));
    num2 = str2double(get(handles.mar_text2,'String'));
    contents = get(handles.mar_popup_dist,'String');
    popupmenuvalue = contents{get(handles.mar_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Normal'
            pd = makedist('Normal','mu',num1,'sigma',num2);
            t = truncate(pd,0,inf);
            mar = random(t,samplesize,1);
        case 'Log Normal'
            pd =
makedist('Lognormal','mu',log(num1)+num2^2,'sigma',num2);
            t = truncate(pd,0,inf);
            mar = random(t,samplesize,1);
        case 'Beta'
            pd = makedist('Beta','a',num1,'b',num2);
            t = truncate(pd,0,inf);
            mar = random(t,samplesize,1);
        case 'Uniform'
            if num1 < num2;
                % In unifrom distribution upper limt must be greater
than lower
                % limit, if not show the error message
                waitfor(errordlg('Upper Limit is less than lower
limt','Uniform Distribution','modal'))
                set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                return;
```

```
            else
                pd = makedist('Uniform','Upper',num1,'Lower',num2);
                t = truncate(pd,0,inf);
                mar = random(t,samplesize,1);
            end
        case 'Exponential'
            pd = makedist('Exponential','mu',num1);
            t = truncate(pd,0,inf);
            mar = random(t,samplesize,1);
    end
    clearvars pd t;
else
    mar = str2double(get(handles.mar_text1,'String'));
    if mar <= 0;
        waitfor(errordlg('Material at Risk cannot be less than or equal
zero','Error','modal'));
        set(handles.run_pushbutton,'str','Show Plot','backg',col);
        return;
    end
end

%compute damage ratio

if b == 0;
    num1 = str2double(get(handles.dr_text1,'String'));
    num2 = str2double(get(handles.dr_text2,'String'));
    contents = get(handles.dr_popup_dist,'String');
    popupmenuvalue = contents{get(handles.dr_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Normal'
            pd = makedist('Normal','mu',num1,'sigma',num2);
            t = truncate(pd,0,1);
            dr = random(t,samplesize,1);
        case 'Log Normal'
            pd =
makedist('Lognormal','mu',log(num1)+num2^2,'sigma',num2);
            t = truncate(pd,0,1);
            dr = random(t,samplesize,1);
        case 'Beta'
            pd = makedist('Beta','a',num1,'b',num2);
            t = truncate(pd,0,1);
            dr = random(t,samplesize,1);
        case 'Uniform'
            if num1 < num2;
                % In unifrom distribution upper limt must be greater
than lower
                % limit, if not show the error message
                waitfor(errordlg('Upper Limit is less than lower
limt','Uniform Distribution','modal'))
```

```
                set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                return;
            else
                pd = makedist('Uniform','Upper',num1,'Lower',num2);
                t = truncate(pd,0,1);
                dr = random(t,samplesize,1);
            end
        case 'Exponential'
            pd = makedist('Exponential','mu',num1);
            t = truncate(pd,0,1);
            dr = random(t,samplesize,1);
        case 'User Defined'
            [Parameters,X,Y] = Parameters.GetUDD(CurrentMAR,'DR');
            dr = zeros(samplesize,1);
            for e = 1:samplesize;
                num_rand=rand;
                ter = size(X);
                for i = 1:ter(2)
                    iSum = 0;
                    for j = 1:i
                        iSum = iSum + Y(j);
                    end
                    if num_rand < iSum
                        if i == 1
                            dr(e) = rand*(X(i+1)-X(i))+X(i);
                        else
                            dr(e) = rand*(X(i)-X(i-1))+X(i);
                        end
                        break;
                    end
                end
            end
    end
    clearvars pd t;
else
    dr = str2double(get(handles.dr_text1,'String'));
    if dr > 1 || dr <= 0;
        waitfor(errordlg('Damage Ratio cannot be greater than 1 or less
than or equal to  zero','Error','modal'));
        set(handles.run_pushbutton,'str','Show Plot','backg',col);
        return;
    end
end

%compute airborne release fraction

if c == 0;
    num1 = str2double(get(handles.arf_text1,'String'));
```

```
    num2 = str2double(get(handles.arf_text2,'String'));
    contents = get(handles.arf_popup_dist,'String');
    popupmenuvalue = contents{get(handles.arf_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Normal'
            pd = makedist('Normal','mu',num1,'sigma',num2);
            t = truncate(pd,0,1);
            arf = random(t,samplesize,1);
        case 'Log Normal'
            pd =
makedist('Lognormal','mu',log(num1)+num2^2,'sigma',num2);
            t = truncate(pd,0,1);
            arf = random(t,samplesize,1);
        case 'Beta'
            pd = makedist('Beta','a',num1,'b',num2);
            t = truncate(pd,0,1);
            arf = random(t,samplesize,1);

        case 'Uniform'
            if num1 < num2;
                % In unifrom distribution upper limt must be greater
than lower
                % limit, if not show the error message
                waitfor(errordlg('Upper Limit is less than lower
limt','Uniform Distribution','modal'))
                set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                return;
            else
                pd = makedist('Uniform','Upper',num1,'Lower',num2);
                t = truncate(pd,0,1);
                arf = random(t,samplesize,1);
            end
        case 'Exponential'
            pd = makedist('Exponential','mu',num1);
            t = truncate(pd,0,1);
            arf = random(t,samplesize,1);
        case 'User Defined'
            [Parameters,X,Y] = Parameters.GetUDD(CurrentMAR,'ARF');
            arf = zeros(samplesize,1);
            for e = 1:samplesize;
                num_rand=rand;
                ter = size(X);
                for i = 1:ter(2)
                    iSum = 0;
                    for j = 1:i
                        iSum = iSum + Y(j);
                    end
                    if num_rand < iSum
```

```
                        if i == 1
                            arf(e) = rand*(X(i+1)-X(i))+X(i);
                        else
                            arf(e) = rand*(X(i)-X(i-1))+X(i);
                        end
                        break;
                    end
                end
            end
    end
    clearvars pd t;
else
    arf = str2double(get(handles.arf_text1,'String'));
    if arf <= 0 || arf > 1
        waitfor(errordlg('Airborne Release Factor cannot be less than 0
or greater than 1', 'Error'));
        set(handles.run_pushbutton,'str','Show Plot','backg',col);
        return
    end
end

%compute Respirable fraction

if d == 0 ;
    num1 = str2double(get(handles.rf_text1,'String'));
    num2 = str2double(get(handles.rf_text2,'String'));
    contents = get(handles.rf_popup_dist,'String');
    popupmenuvalue = contents{get(handles.rf_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Normal'
            pd = makedist('Normal','mu',num1,'sigma',num2);
            t = truncate(pd,0,1);
            rf = random(t,samplesize,1);
        case 'Log Normal'
            pd =
makedist('Lognormal','mu',log(num1)+num2^2,'sigma',num2);
            t = truncate(pd,0,1);
            rf = random(t,samplesize,1);
        case 'Beta'
            pd = makedist('Beta','a',num1,'b',num2);
            t = truncate(pd,0,1);
            rf = random(t,samplesize,1);
        case 'Uniform'
            if num1 < num2;
                % In unifrom distribution upper limt must be greater
than lower
                % limit, if not show the error message
                waitfor(errordlg('Upper Limit is less than lower
limt','Uniform Distribution','modal'))
```

```
                set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                return;
            else
                pd = makedist('Uniform','Upper',num1,'Lower',num2);
                t = truncate(pd,0,1);
                rf= random(t,samplesize,1);
            end
        case 'Exponential'
            pd = makedist('Exponential','mu',num1);
            t = truncate(pd,0,1);
            rf = random(t,samplesize,1);
        case 'User Defined'
            [Parameters,X,Y] = Parameters.GetUDD(CurrentMAR,'RF');
            rf = zeros(samplesize,1);
            for e = 1:samplesize;
                num_rand=rand;
                ter = size(X);
                for i = 1:ter(2)
                    iSum = 0;
                    for j = 1:i
                        iSum = iSum + Y(j);
                    end
                    if num_rand < iSum
                        if i == 1
                            rf(e) = rand*(X(i+1)-X(i))+X(i);
                        else
                            rf(e) = rand*(X(i)-X(i-1))+X(i);
                        end
                        break;
                    end
                end
            end
    end
    clearvars pd t;
else
    rf = str2double(get(handles.rf_text1,'String'));
    if rf <= 0 || rf > 1;
        waitfor(errordlg('Respirable Factor cannot be less than or
equal 0 or greater than 1', 'Error'));
        set(handles.run_pushbutton,'str','Show Plot','backg',col);
        return
    end
end

%compute leak path factor

if e == 0;
    num1 = str2double(get(handles.lpf_text1,'String'));
```

```
    num2 = str2double(get(handles.lpf_text2,'String'));
    contents = get(handles.lpf_popup_dist,'String');
    popupmenuvalue = contents{get(handles.lpf_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Normal'
            pd = makedist('Normal','mu',num1,'sigma',num2);
            t = truncate(pd,0,1);
            lpf = random(t,samplesize,1);
        case 'Log Normal'
            pd =
makedist('Lognormal','mu',log(num1)+num2^2,'sigma',num2);
            t = truncate(pd,0,1);
            lpf = random(t,samplesize,1);
        case 'Beta'
            pd = makedist('Beta','a',num1,'b',num2);
            t = truncate(pd,0,1);
            lpf = random(t,samplesize,1);
        case 'Uniform'
            if num1 < num2;
                % In unifrom distribution upper limt must be greater
than lower
                % limit, if not show the error message
                waitfor(errordlg('Upper Limit is less than lower
limt','Uniform Distribution','modal'))
                set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                return;
            else
                pd = makedist('Uniform','Upper',num1,'Lower',num2);
                t = truncate(pd,0,1);
                lpf = random(t,samplesize,1);
            end
        case 'Exponential'
            pd = makedist('Exponential','mu',num1);
            t = truncate(pd,0,1);
            lpf = random(t,samplesize,1);
        case 'User Defined'
            [Parameters,X,Y] = Parameters.GetUDD(CurrentMAR,'LPF');
            lpf = zeros(samplesize,1);
            for e = 1:samplesize;
                num_rand=rand;
                ter = size(X);
                for i = 1:ter(2)
                    iSum = 0;
                    for j = 1:i
                        iSum = iSum + Y(j);
                    end
                    if num_rand < iSum
                        if i == 1
```

```
                            lpf(e) = rand*(X(i+1)-X(i))+X(i);
                        else
                            lpf(e) = rand*(X(i)-X(i-1))+X(i);
                        end
                        break;
                    end
                end
            end

    end
    clearvars pd t;
else
    lpf = str2double(get(handles.lpf_text1,'String'));
    if lpf > 1 || lpf <= 0;
        waitfor(errordlg('Leak Path Factor cannot be less than or equal
to 0 or greater than 1', 'Error'));
        set(handles.run_pushbutton,'str','Show Plot','backg',col);
        return;
    end
end

%compute source term
%st = mar.*dr.*arf.*rf.*lpf; %Redunant code, ST is not used until after
                             %it is recalculated below.
%compute breathing rate

if f == 0;
    a = 8.33*10^-4;
    b = 4.17*10^-4 ;
    c = 1.5*10^-4 ;
    d = 1.25*10^-4;

    for e = 1:samplesize;
        num_rand=rand;
        if num_rand <= 0.17
            n(e) = rand*(8.33E-4-4.17E-4)+4.17E-4;
        elseif num_rand > 0.17 && num_rand <= 0.34;
            n(e) = rand*(4.17E-4-1.5E-4)+1.5E-4;
        elseif num_rand >0.34
            n(e) = rand*(1.5E-4-1.25E-4)+1.25E-4;
        end
    end

    br=n';
    clearvars n;
else
    br = str2double(get(handles.br_text1,'String'));
    if br <= 0;
```

```
        waitfor(errordlg('Breathing Rate cannot be less than or equal
to 0', 'Error'));
        set(handles.run_pushbutton,'str','Show Plot','backg',col);
        return;
    end
end

%compute dose conversion factor


if g == 0;
    num1 = str2double(get(handles.dcf_text1,'String'));
    num2 = str2double(get(handles.dcf_text2,'String'));
    contents = get(handles.dcf_popup_dist,'String');
    popupmenuvalue = contents{get(handles.dcf_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Normal'
            pd = makedist('Normal','mu',num1,'sigma',num2);
            t = truncate(pd,0,inf);
            dcf = random(t,samplesize,1);
        case 'Log Normal'
            pd =
makedist('Lognormal','mu',log(num1)+num2^2,'sigma',num2);
            t = truncate(pd,0,inf);
            dcf = random(t,samplesize,1);
        case 'Beta'
            pd = makedist('Beta','a',num1,'b',num2);
            t = truncate(pd,0,inf);
            dcf = random(t,samplesize,1);
        case 'Uniform'
            if num1 < num2;
                % In unifrom distribution upper limt must be greater
than lower
                % limit, if not show the error message
                waitfor(errordlg('Upper Limit is less than lower
limt','Uniform Distribution','modal'))
                set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                return;
            else
                pd = makedist('Uniform','Upper',num1,'Lower',num2);
                t = truncate(pd,0,inf);
                dcf = random(t,samplesize,1);

            end
        case 'Exponential'
            pd = makedist('Exponential','mu',num1);
            t = truncate(pd,0,inf);
            dcf = random(t,samplesize,1);
```

```
        case 'User Defined'
            [Parameters,X,Y] = Parameters.GetUDD(CurrentMAR,'DCF');
            dcf = zeros(samplesize,1);
            for e = 1:samplesize;
                num_rand=rand;
                ter = size(X);
                for i = 1:ter(2)
                    iSum = 0;
                    for j = 1:i
                        iSum = iSum + Y(j);
                    end
                    if num_rand < iSum
                        if i == 1
                            dcf(e) = rand*(X(i+1)-X(i))+X(i);
                        else
                            dcf(e) = rand*(X(i)-X(i-1))+X(i);
                        end
                        break;
                    end
                end
            end
    end
    clearvars pd t;
else
    dcf = str2double(get(handles.dcf_text1,'String'));
    if dcf <= 0;
        waitfor(errordlg('Dose Conversion Factor cannot be less than or
equal to 0', 'Error'));
        set(handles.run_pushbutton,'str','Show Plot','backg',col);
        return;
    end
end

%compute chi/Q

if h == 0;
    distance = str2double(get(handles.distance_text1,'String'));
    distance1 = str2double(get(handles.distance_text2,'String'));
    pd = makedist('Normal','mu',0,'sigma',distance1);
    crossdistance = random(pd,samplesize,1);

    num1 = str2double(get(handles.windspeed_text1,'String'));
    num2 = str2double(get(handles.windspeed_text2,'String'));

    contents = get(handles.windspeed_popup_dist,'String');
    popupmenuvalue =
contents{get(handles.windspeed_popup_dist,'Value')};
    switch popupmenuvalue
        case 'Normal'
```

```
            pd = makedist('Normal','mu',num1,'sigma',num2);
            t = truncate(pd,0.1,inf);
            windS = random(t,samplesize,1);
        case 'Uniform'
            if num1 < num2;
                % In unifrom distribution upper limt must be greater
than lower
                % limit, if not show the error message
                waitfor(errordlg('Upper Limit is less than lower
limt','Uniform Distribution','modal'))
                set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                return;
            else
                pd = makedist('Uniform','Upper',num1,'Lower',num2);
                t = truncate(pd,0.1,inf);
                windS = random(t,samplesize,1);
            end
    end



    contents2 = get(handles.terrain_popup,'String');
    terrainvalue = contents2{get(handles.terrain_popup,'Value')};

    contents3 = get(handles.stability_popup,'String');
    stability = contents3{get(handles.stability_popup,'Value')};


    height = str2double(get(handles.height_text,'String'));


    switch terrainvalue
        case 'Rural/Open Country'
            switch stability
                case 'A'
                    sigma_y = 0.22*distance*(1+0.0001*distance)^(-0.5);
                    sigma_z = 0.20*distance;
                case 'B'
                    sigma_y = 0.16*distance*(1+0.0001*distance)^(-0.5);
                    sigma_z = 0.12*distance;
                case 'C'
                    sigma_y = 0.11*distance*(1+0.0001*distance)^(-0.5);
                    sigma_z = 0.08*distance*(1+0.0002*distance)^(-0.5);
                case 'D'
                    sigma_y = 0.08*distance*(1+0.0001*distance)^(-0.5);
                    sigma_z = 0.06*distance*(1+0.0015*distance)^(-0.5);
                case 'E'
                    sigma_y = 0.06*distance*(1+0.0001*distance)^(-0.5);
```

```
                    sigma_z = 0.03*distance*(1+0.0003*distance)^(-1);
                case 'F'
                    sigma_y = 0.04*distance*(1+0.0001*distance)^(-0.5);
                    sigma_z = 0.016*distance*(1+0.0003*distance)^(-1);
                case 'Select Stability Condition'
                    waitfor(errordlg('Select Stability
Conditions','Error','modal'));
                    set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                    return;
            end
        case 'Select Terrain'
            switch stability
                case 'A'
                    waitfor(errordlg('Select
terrain','Error','modal'));
                    set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                    return;
                case 'B'
                    waitfor(errordlg('Select terrain','Error','modal'))
                    set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                    return;
                case 'C'
                    waitfor(errordlg('Select
terrain','Error','modal'));
                    set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                    return;
                case 'D'
                    waitfor(errordlg('Select
terrain','Error','modal'));
                    set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                    return;
                case 'E'
                    waitfor(errordlg('Select
terrain','Error','modal'));
                    set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                    return;
                case 'F'
                    waitfor(errordlg('Select
terrain','Error','modal'));
                    set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                    return;
                case 'Select Stability Condition'
```

```
                    waitfor(errordlg('Select Terrain & Stability
Condition','Error','modal'));
                    set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                    return;
            end
        case 'Urban Area'
            switch stability
                case 'A-B'
                    sigma_y = 0.32*distance*(1+0.0004*distance)^(-0.5);
                    sigma_z = 0.24*distance*(1+0.001*distance)^(0.5);
                case 'C'
                    sigma_y = 0.22*distance*(1+0.0004*distance)^(-0.5);
                    sigma_z = 0.2*distance;
                case 'D'
                    sigma_y = 0.16*distance*(1+0.0004*distance)^(-0.5);
                    sigma_z = 0.14*distance*(1+0.0003*distance)^(-0.5);
                case 'E-F'
                    sigma_y = 0.11*distance*(1+0.0004*distance)^(-0.5);
                    sigma_z = 0.08*distance*(1+0.0015*distance)^(-0.5);
                case 'Select Stability Condition'
                    waitfor(errordlg('Select Stability
Conditions','Error','modal'));
                    set(handles.run_pushbutton,'str','Show
Plot','backg',col);
                    return;
            end

    end


    cq = (exp((-crossdistance.^2/(2*(sigma_y)^2))-
(height^2/(2*(sigma_z)^2))./...
        (pi*windS.*sigma_y*sigma_z));
    clearvars pd t windS;
else
    cq = str2double(get(handles.cq_text1,'String'));
    if cq <= 0;
        errordlg('\Chi/Q Conversion Factor cannot be less than or equal
to 0', 'Error');
        set(handles.run_pushbutton,'str','Show Plot','backg',col);
        return;
    end
end
%Compute Source Term
st = mar.*dr.*arf.*rf.*lpf;
clearvars mar dr arf rf lpf;
% assignin('base','mar', mar);
% assignin('base','dr', dr);
```

```
% assignin('base','arf', arf);
% assignin('base','rf', rf);
% assignin('base','lpf', lpf);
% assignin('base','st', st);
% assignin('base','cq', cq);
% assignin('base','br', br);
% assignin('base','dcf', dcf);

cla(handles.axes1,'reset');
ced = (cq.*st.*br.*dcf).*100; %Times 100 for Sv->Rem conversion.
clearvars cq st br dcf;
axes(handles.axes1)
if length(ced)== 1;
%     plot(ced,ced,'*');
%     grid off;
    str1 = sprintf('CED is %0.3e rem',ced);
    text(0.3,0.5,['\fontsize{22}' str1]);
    axis auto
    set(handles.fit_dist,'Enable','off')
else
    set(handles.fit_dist,'Enable','on')
    x = mean(ced); %average ced
    % code here
    y = std(ced); %Sigma of ced
    z = median(ced); % median
    prc90 = prctile(ced,95); % 95 percentile
    nbins = max(min(length(ced)./10,100),50); %Break domain up into 100
"bins"
    xi = linspace(min(ced),max(ced),nbins); %Draw a linespace over the
range
    assignin('base','cedxi', xi);           %of the ced.
    dx = mean(diff(xi));
    fi = histc(ced,xi-dx); %Count the number of ced's between a point
in xi
                           %and the next point.
    fi = fi./sum(fi)./dx;
    assignin('base','cedfi2', fi);
    %     assignin('base','fi2', fi);
    %Commented out the below due to changes in functionality.
    %{
    bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
    bar(xi,fi,'FaceColor','m','EdgeColor','m','BarWidth', 1);

    str = sprintf('\\fontsize{13} Mean Value of CED = %0.3e rem with
std devitation= %0.3e rem',x,y);
    title(str,'Units', 'normalized', ...
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')
    xlabel('Commited Effective Dose (rem)')
```

```
    ylabel('Probability Density')
    legend('Random Generated','Location','NE')
    axis tight;
    grid on;
    %}
end
assignin('base','ced', ced);
setappdata(0,'ced',ced);
set(handles.run_pushbutton,'str','Show Plot','backg',col);

I = CurrentMAR;
if ~I==0    %Save data to object after generating CED.
    switch I
        case 1
            set(Parameters, 'CED1',ced);

            TempArray = get(Parameters,'AvgCED');
            TempArray(I) = x;
            set(Parameters, 'AvgCED',TempArray);

            TempArray = get(Parameters,'StdCED');
            TempArray(I) = y;
            set(Parameters, 'StdCED',TempArray);

            TempArray = get(Parameters,'MedCED');
            TempArray(I) = z;
            set(Parameters, 'MedCED',TempArray);

            TempArray = get(Parameters,'Ninty_fifth');
            TempArray(I) = prc90;
            set(Parameters, 'Ninty_fifth',TempArray);
            set(Parameters, 'XResult1',xi);
            set(Parameters, 'YResult1',fi);

        case 2
            set(Parameters, 'CED2',ced);
            TempArray = get(Parameters,'AvgCED');
            TempArray(I) = x;
            set(Parameters, 'AvgCED',TempArray);
            TempArray = get(Parameters,'StdCED');
            TempArray(I) = y;
            set(Parameters, 'StdCED',TempArray);
            TempArray = get(Parameters,'MedCED');
            TempArray(I) = z;
            set(Parameters, 'MedCED',TempArray);

            TempArray = get(Parameters,'Ninty_fifth');
            TempArray(I) = prc90;
            set(Parameters, 'Ninty_fifth',TempArray);
```

```
            set(Parameters, 'XResult2',xi);
            set(Parameters, 'YResult2',fi);
        case 3
            set(Parameters, 'CED3',ced);
            TempArray = get(Parameters,'AvgCED');
            TempArray(I) = x;
            set(Parameters, 'AvgCED',TempArray);
            TempArray = get(Parameters,'StdCED');
            TempArray(I) = y;
            set(Parameters, 'StdCED',TempArray);
            TempArray = get(Parameters,'MedCED');
            TempArray(I) = z;
            set(Parameters, 'MedCED',TempArray);

            TempArray = get(Parameters,'Ninty_fifth');
            TempArray(I) = prc90;
            set(Parameters, 'Ninty_fifth',TempArray);
            set(Parameters, 'XResult3',xi);
            set(Parameters, 'YResult3',fi);
        case 4
            set(Parameters, 'CED4',ced);
            TempArray = get(Parameters,'AvgCED');
            TempArray(I) = x;
            set(Parameters, 'AvgCED',TempArray);
            TempArray = get(Parameters,'StdCED');
            TempArray(I) = y;
            set(Parameters, 'StdCED',TempArray);
            TempArray = get(Parameters,'MedCED');
            TempArray(I) = z;
            set(Parameters, 'MedCED',TempArray);

            TempArray = get(Parameters,'Ninty_fifth');
            TempArray(I) = prc90;
            set(Parameters, 'Ninty_fifth',TempArray);
            set(Parameters, 'XResult4',xi);
            set(Parameters, 'YResult4',fi);
    end
else
    msgbox('MAR State Exclusivity Error; MAR Selection will
close.','Fatal Error')
    close(handles.SodaMain);
end

end

function I = GetCurrentMAR()
%Set a variable to this function's output to get the number of the
currently selected MAR.
h1 = findobj('Tag', 'radioMAR1');
```

```
h2 = findobj('Tag', 'radioMAR2');
h3 = findobj('Tag', 'radioMAR3');
h4 = findobj('Tag', 'radioMAR4');
if get(h1, 'Value')
    I = 1;
elseif get(h2, 'Value')
    I = 2;
elseif get(h3, 'Value')
    I = 3;
elseif get(h4, 'Value')
    I = 4;
else
    I = 0;
end
end
```

File 2, MAR_Selection.m:

```
function varargout = MAR_Selection(varargin)
% MAR_SELECTION MATLAB code for MAR_Selection.fig
%      MAR_SELECTION, by itself, creates a new MAR_SELECTION or raises
the existing
%      singleton*.
%
%      H = MAR_SELECTION returns the handle to a new MAR_SELECTION or
the handle to
%      the existing singleton*.
%
%      MAR_SELECTION('CALLBACK',hObject,eventData,handles,...) calls
the local
%      function named CALLBACK in MAR_SELECTION.M with the given input
arguments.
%
%      MAR_SELECTION('Property','Value',...) creates a new
MAR_SELECTION or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before MAR_Selection_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to MAR_Selection_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
```

```
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help MAR_Selection

% Last Modified by GUIDE v2.5 04-Jan-2016 17:09:15

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @MAR_Selection_OpeningFcn, ...
                   'gui_OutputFcn',  @MAR_Selection_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before MAR_Selection is made visible.
function MAR_Selection_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to MAR_Selection (see VARARGIN)

% Choose default command line output for MAR_Selection
handles.output = hObject;
axes(handles.logo_axes);
imshow('sodaLogo1.png');
% Update handles structure
guidata(hObject, handles);
%Check Code that Disables Element Buttons for which there is no
%Isotopes in the database   *********
global Parameters
global CurrentMAR
Parameters = SODA_Parameters();
if ~isempty(varargin)
    Parameters = varargin{1,1};
```

```
end
S = csvread('MAR_Database.csv', 1, 0);
for i=1:112 %Step through each element and check for data
    if S(i,2) == 0
        ObjName = strcat('element', num2str(i));
        H = findobj('Tag', ObjName);
        set(H, 'Enable', 'inactive');   %Disable if no data
        set(H, 'BackgroundColor', [0.5,0.5,0.5]); %Change color gray
    end

end
CurrentMAR = 1;
h1 = findobj('Tag', 'textIso');
A = size(get(h1,'String'));
assert(A(1) == 0,'textIso String must be empty on load. Check String
property for extra lines or characters.');

%Check for existing MAR1 data.
if Parameters.MAR(1) ~= 0
    h1 = findobj('Tag', 'textIso'); %Get handle to the selected isotope
text
    h2 = findobj('Tag', 'editBq'); %Get handle to quantity textbox.
    set(h1,'String',Parameters.Isotope{CurrentMAR});
    set(h2,'String',num2str(Parameters.MAR(CurrentMAR)));
    set(h2,'Enable', 'on');
end

% UIWAIT makes MAR_Selection wait for user response (see UIRESUME)
 uiwait(handles.figure1);

% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
if isequal(get(hObject, 'waitstatus'), 'waiting')
% The GUI is still in UIWAIT, us UIRESUME
uiresume(hObject);
else
% The GUI is no longer waiting, just close it
delete(hObject);
end


% --- Outputs from this function are returned to the command line.
function varargout = MAR_Selection_OutputFcn(hObject, eventdata,
handles)
```

```
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
global Parameters;
Parameters = EnsureDataIntegrity(Parameters);
varargout{1} = Parameters;
delete(handles.figure1);



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as
a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in element1.
function element1_Callback(hObject, eventdata, handles)
% hObject    handle to element1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);

%Each element button sends its handle to GetAvailIso, where most of the
%work is performed.

% --- Executes on button press in element3.
```

```
function element3_Callback(hObject, eventdata, handles)
% hObject    handle to element3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element11.
function element11_Callback(hObject, eventdata, handles)
% hObject    handle to element11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element19.
function element19_Callback(hObject, eventdata, handles)
% hObject    handle to element19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element37.
function element37_Callback(hObject, eventdata, handles)
% hObject    handle to element37 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element55.
function element55_Callback(hObject, eventdata, handles)
% hObject    handle to element55 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element87.
function element87_Callback(hObject, eventdata, handles)
% hObject    handle to element87 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in pushbutton13.
function pushbutton13_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in pushbutton15.
function pushbutton15_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element4.
function element4_Callback(hObject, eventdata, handles)
% hObject    handle to element4 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element12.
function element12_Callback(hObject, eventdata, handles)
% hObject    handle to element12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element20.
function element20_Callback(hObject, eventdata, handles)
% hObject    handle to element20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element38.
function element38_Callback(hObject, eventdata, handles)
% hObject    handle to element38 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element56.
function element56_Callback(hObject, eventdata, handles)
% hObject    handle to element56 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element88.
function element88_Callback(hObject, eventdata, handles)
% hObject    handle to element88 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element21.
function element21_Callback(hObject, eventdata, handles)
% hObject    handle to element21 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element39.
function element39_Callback(hObject, eventdata, handles)
% hObject    handle to element39 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element57.
function element57_Callback(hObject, eventdata, handles)
% hObject    handle to element57 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element89.
function element89_Callback(hObject, eventdata, handles)
% hObject    handle to element89 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element22.
function element22_Callback(hObject, eventdata, handles)
% hObject    handle to element22 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element40.
function element40_Callback(hObject, eventdata, handles)
% hObject    handle to element40 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element72.
function element72_Callback(hObject, eventdata, handles)
% hObject    handle to element72 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
GetAvailIso(hObject);


% --- Executes on button press in element104.
function element104_Callback(hObject, eventdata, handles)
% hObject    handle to element104 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element23.
function element23_Callback(hObject, eventdata, handles)
% hObject    handle to element23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element41.
function element41_Callback(hObject, eventdata, handles)
% hObject    handle to element41 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element73.
function element73_Callback(hObject, eventdata, handles)
% hObject    handle to element73 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element105.
function element105_Callback(hObject, eventdata, handles)
% hObject    handle to element105 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element24.
function element24_Callback(hObject, eventdata, handles)
% hObject    handle to element24 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);
```

```
% --- Executes on button press in element42.
function element42_Callback(hObject, eventdata, handles)
% hObject    handle to element42 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element74.
function element74_Callback(hObject, eventdata, handles)
% hObject    handle to element74 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element106.
function element106_Callback(hObject, eventdata, handles)
% hObject    handle to element106 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element25.
function element25_Callback(hObject, eventdata, handles)
% hObject    handle to element25 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element43.
function element43_Callback(hObject, eventdata, handles)
% hObject    handle to element43 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element75.
function element75_Callback(hObject, eventdata, handles)
% hObject    handle to element75 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);
```

```
% --- Executes on button press in element107.
function element107_Callback(hObject, eventdata, handles)
% hObject    handle to element107 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element26.
function element26_Callback(hObject, eventdata, handles)
% hObject    handle to element26 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element44.
function element44_Callback(hObject, eventdata, handles)
% hObject    handle to element44 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element76.
function element76_Callback(hObject, eventdata, handles)
% hObject    handle to element76 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element108.
function element108_Callback(hObject, eventdata, handles)
% hObject    handle to element108 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element27.
function element27_Callback(hObject, eventdata, handles)
% hObject    handle to element27 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);
```

```
% --- Executes on button press in element45.
function element45_Callback(hObject, eventdata, handles)
% hObject    handle to element45 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element77.
function element77_Callback(hObject, eventdata, handles)
% hObject    handle to element77 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element109.
function element109_Callback(hObject, eventdata, handles)
% hObject    handle to element109 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element28.
function element28_Callback(hObject, eventdata, handles)
% hObject    handle to element28 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element46.
function element46_Callback(hObject, eventdata, handles)
% hObject    handle to element46 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element78.
function element78_Callback(hObject, eventdata, handles)
% hObject    handle to element78 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element110.
```

```
function element110_Callback(hObject, eventdata, handles)
% hObject    handle to element110 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element29.
function element29_Callback(hObject, eventdata, handles)
% hObject    handle to element29 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element47.
function element47_Callback(hObject, eventdata, handles)
% hObject    handle to element47 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element79.
function element79_Callback(hObject, eventdata, handles)
% hObject    handle to element79 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element111.
function element111_Callback(hObject, eventdata, handles)
% hObject    handle to element111 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element30.
function element30_Callback(hObject, eventdata, handles)
% hObject    handle to element30 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element48.
function element48_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to element48 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element80.
function element80_Callback(hObject, eventdata, handles)
% hObject    handle to element80 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element112.
function element112_Callback(hObject, eventdata, handles)
% hObject    handle to element112 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element5.
function element5_Callback(hObject, eventdata, handles)
% hObject    handle to element5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element13.
function element13_Callback(hObject, eventdata, handles)
% hObject    handle to element13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element31.
function element31_Callback(hObject, eventdata, handles)
% hObject    handle to element31 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element49.
function element49_Callback(hObject, eventdata, handles)
% hObject    handle to element49 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element81.
function element81_Callback(hObject, eventdata, handles)
% hObject    handle to element81 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in pushbutton99.
function pushbutton99_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton99 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element6.
function element6_Callback(hObject, eventdata, handles)
% hObject    handle to element6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element14.
function element14_Callback(hObject, eventdata, handles)
% hObject    handle to element14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element32.
function element32_Callback(hObject, eventdata, handles)
% hObject    handle to element32 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element50.
function element50_Callback(hObject, eventdata, handles)
% hObject    handle to element50 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element82.
function element82_Callback(hObject, eventdata, handles)
% hObject    handle to element82 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in pushbutton106.
function pushbutton106_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton106 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element7.
function element7_Callback(hObject, eventdata, handles)
% hObject    handle to element7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element15.
function element15_Callback(hObject, eventdata, handles)
% hObject    handle to element15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element33.
function element33_Callback(hObject, eventdata, handles)
% hObject    handle to element33 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element51.
function element51_Callback(hObject, eventdata, handles)
% hObject    handle to element51 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
GetAvailIso(hObject);


% --- Executes on button press in element83.
function element83_Callback(hObject, eventdata, handles)
% hObject    handle to element83 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in pushbutton113.
function pushbutton113_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton113 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element8.
function element8_Callback(hObject, eventdata, handles)
% hObject    handle to element8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element16.
function element16_Callback(hObject, eventdata, handles)
% hObject    handle to element16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element34.
function element34_Callback(hObject, eventdata, handles)
% hObject    handle to element34 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element52.
function element52_Callback(hObject, eventdata, handles)
% hObject    handle to element52 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);
```

```
% --- Executes on button press in element84.
function element84_Callback(hObject, eventdata, handles)
% hObject    handle to element84 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in pushbutton120.
function pushbutton120_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton120 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element9.
function element9_Callback(hObject, eventdata, handles)
% hObject    handle to element9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element17.
function element17_Callback(hObject, eventdata, handles)
% hObject    handle to element17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element35.
function element35_Callback(hObject, eventdata, handles)
% hObject    handle to element35 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element53.
function element53_Callback(hObject, eventdata, handles)
% hObject    handle to element53 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);
```

```
% --- Executes on button press in element85.
function element85_Callback(hObject, eventdata, handles)
% hObject    handle to element85 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in pushbutton127.
function pushbutton127_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton127 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element2.
function element2_Callback(hObject, eventdata, handles)
% hObject    handle to element2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element10.
function element10_Callback(hObject, eventdata, handles)
% hObject    handle to element10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element18.
function element18_Callback(hObject, eventdata, handles)
% hObject    handle to element18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element36.
function element36_Callback(hObject, eventdata, handles)
% hObject    handle to element36 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);
```

```
% --- Executes on button press in element54.
function element54_Callback(hObject, eventdata, handles)
% hObject    handle to element54 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element86.
function element86_Callback(hObject, eventdata, handles)
% hObject    handle to element86 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in pushbutton134.
function pushbutton134_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton134 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element58.
function element58_Callback(hObject, eventdata, handles)
% hObject    handle to element58 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element59.
function element59_Callback(hObject, eventdata, handles)
% hObject    handle to element59 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element60.
function element60_Callback(hObject, eventdata, handles)
% hObject    handle to element60 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element61.
```

```
function element61_Callback(hObject, eventdata, handles)
% hObject    handle to element61 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element62.
function element62_Callback(hObject, eventdata, handles)
% hObject    handle to element62 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element63.
function element63_Callback(hObject, eventdata, handles)
% hObject    handle to element63 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);




% --- Executes on button press in element64.
function element64_Callback(hObject, eventdata, handles)
% hObject    handle to element64 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element65.
function element65_Callback(hObject, eventdata, handles)
% hObject    handle to element65 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element66.
function element66_Callback(hObject, eventdata, handles)
% hObject    handle to element66 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);
```

```
% --- Executes on button press in element67.
function element67_Callback(hObject, eventdata, handles)
% hObject    handle to element67 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element68.
function element68_Callback(hObject, eventdata, handles)
% hObject    handle to element68 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element69.
function element69_Callback(hObject, eventdata, handles)
% hObject    handle to element69 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element70.
function element70_Callback(hObject, eventdata, handles)
% hObject    handle to element70 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element71.
function element71_Callback(hObject, eventdata, handles)
% hObject    handle to element71 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element90.
function element90_Callback(hObject, eventdata, handles)
% hObject    handle to element90 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element91.
```

```
function element91_Callback(hObject, eventdata, handles)
% hObject    handle to element91 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element92.
function element92_Callback(hObject, eventdata, handles)
% hObject    handle to element92 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)\
GetAvailIso(hObject);




% --- Executes on button press in element93.
function element93_Callback(hObject, eventdata, handles)
% hObject    handle to element93 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element94.
function element94_Callback(hObject, eventdata, handles)
% hObject    handle to element94 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element95.
function element95_Callback(hObject, eventdata, handles)
% hObject    handle to element95 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element96.
function element96_Callback(hObject, eventdata, handles)
% hObject    handle to element96 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);

% --- Executes on button press in element97.
```

```
function element97_Callback(hObject, eventdata, handles)
% hObject    handle to element97 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element98.
function element98_Callback(hObject, eventdata, handles)
% hObject    handle to element98 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element99.
function element99_Callback(hObject, eventdata, handles)
% hObject    handle to element99 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element100.
function element100_Callback(hObject, eventdata, handles)
% hObject    handle to element100 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element101.
function element101_Callback(hObject, eventdata, handles)
% hObject    handle to element101 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element102.
function element102_Callback(hObject, eventdata, handles)
% hObject    handle to element102 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in element103.
function element103_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to element103 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GetAvailIso(hObject);


% --- Executes on button press in isotope1.
function isotope1_Callback(hObject, eventdata, handles)
% hObject    handle to isotope1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
SetMarDCF(hObject);


% --- Executes on button press in isotope2.
function isotope2_Callback(hObject, eventdata, handles)
% hObject    handle to isotope2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
SetMarDCF(hObject);

% --- Executes on button press in isotope3.
function isotope3_Callback(hObject, eventdata, handles)
% hObject    handle to isotope3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
SetMarDCF(hObject);

% --- Executes on button press in isotope4.
function isotope4_Callback(hObject, eventdata, handles)
% hObject    handle to isotope4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
SetMarDCF(hObject);

% --- Executes on button press in isotope5.
function isotope5_Callback(hObject, eventdata, handles)
% hObject    handle to isotope5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
SetMarDCF(hObject);

% --- Executes on button press in isotope6.
function isotope6_Callback(hObject, eventdata, handles)
% hObject    handle to isotope6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
SetMarDCF(hObject);
```

```
% --- Executes on button press in isotope7.
function isotope7_Callback(hObject, eventdata, handles)
% hObject    handle to isotope7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
SetMarDCF(hObject);

% --- Executes on button press in exportbtn.
function exportbtn_Callback(hObject, eventdata, handles)
% hObject    handle to exportbtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
CloseCond = SaveMARData();
if CloseCond == 1
    close(handles.figure1);
else
    msgbox('One or more of MAR Selections are Incomplete. Input
quantity with any selected isotope. ' ,'Not Ready for Export')
end


% --- Executes on selection change in isotope_list.
function isotope_list_Callback(hObject, eventdata, handles)
% hObject    handle to isotope_list (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns isotope_list
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
isotope_list


% --- Executes during object creation, after setting all properties.
function isotope_list_CreateFcn(hObject, eventdata, handles)
% hObject    handle to isotope_list (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function editBq_Callback(hObject, eventdata, handles)
% hObject    handle to editBq (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editBq as text
%        str2double(get(hObject,'String')) returns contents of editBq
as a double


% --- Executes during object creation, after setting all properties.
function editBq_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editBq (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
editBq.
function editBq_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to editBq (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject,'Enable','on');
set(handles.editBq,'string',[]);

% --- Executes on button press in MAR1_radio.
function MAR1_radio_Callback(hObject, eventdata, handles)
% hObject    handle to MAR1_radio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ChangeMAR(hObject);
% Hint: get(hObject,'Value') returns toggle state of MAR1_radio

% --- Executes on button press in MAR2_radio.
function MAR2_radio_Callback(hObject, eventdata, handles)
% hObject    handle to MAR2_radio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ChangeMAR(hObject);
% Hint: get(hObject,'Value') returns toggle state of MAR2_radio
```

```
% --- Executes on button press in MAR3_radio.
function MAR3_radio_Callback(hObject, eventdata, handles)
% hObject    handle to MAR3_radio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ChangeMAR(hObject);
% Hint: get(hObject,'Value') returns toggle state of MAR3_radio


% --- Executes on button press in MAR4_radio.
function MAR4_radio_Callback(hObject, eventdata, handles)
% hObject    handle to MAR4_radio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ChangeMAR(hObject);
% Hint: get(hObject,'Value') returns toggle state of MAR4_radio




%**************************************************************************
****
function GetAvailIso(hObject)
%Get Element Number and Call SetIsoBox for that Element
EleStr = get(hObject, 'Tag');
EleStr = EleStr(8:length(EleStr)); %Get Atomic Number from Tag
EleNum = str2double(EleStr);
EleLet = get(hObject, 'String');
SizeCheck = size(EleLet);

%If this assertion is thrown, check element that is clicked for having
%extra lines in its String property.
assert(SizeCheck(1) == 1,'EleLet does not have size 1x2, Property
Error.')

%Find Isotopes of a Clicked Element and Fill Isotope Boxes
h = zeros(7);
S = csvread('MAR_Database.csv', 1, 0);
global DCFArray
DCFArray = zeros(7);
for i = 1:7 %Loop to set the isotope boxes
    str = ['isotope' num2str(i)]; %Set string to object name
dynamically
    h(i) = findobj('Tag', str);  %Get tag to isotope(i)
    IsoStr = strcat(num2str(S(EleNum, i+1)), EleLet); %Add element
symbol
    if S(EleNum, i+1) ~= 0                              %after isotope
mass
```

```
        set(h(i), 'Enable', 'On');  %Enable active isotope buttons
        set(h(i), 'String', IsoStr);
        DCFArray(i) = S(EleNum, i+8); %Set DCF data for the selected
isotp
    else
        set(h(i), 'Enable', 'Off'); %Disable unused isotope buttons
        set(h(i), 'String', '');
    end
end


function I = GetCurrentMAR()
%Set a variable to this function's output to get the number of selected
MAR.
h1 = findobj('Tag', 'MAR1_radio');
h2 = findobj('Tag', 'MAR2_radio');
h3 = findobj('Tag', 'MAR3_radio');
h4 = findobj('Tag', 'MAR4_radio');
if get(h1, 'Value')
    I = 1;
elseif get(h2, 'Value')
    I = 2;
elseif get(h3, 'Value')
    I = 3;
elseif get(h4, 'Value')
    I = 4;
else
    I = 0;
end


function SetMarDCF(hObject)
%Find and set DCF for selected isotope
IsoStr = get(hObject, 'Tag');
IsoStr = IsoStr(8);
IsoNum = str2double(IsoStr); %Get index to select correct DCF
IsoLet = get(hObject, 'String'); %Get string to display

global DCFArray %Declare so this function can access this Global.
global Parameters   %""

%Select a DCF that is ready for export corresponding to a selection
%by user.
I = GetCurrentMAR();
TempArray = get(Parameters,'DCF');
if ~I==0
    TempArray(I) = DCFArray(IsoNum);
    set(Parameters,'DCF',TempArray);
else %This should not happen under any normal circumstance.
```

```
     msgbox('MAR State Exclusivity Error; MAR Selection will
close.','Fatal Error')
     set(Parameters,'DCF',[0 0 0 0]); %Ensure no bad data is returned
     set(Parameters,'MAR',[0 0 0 0]);
     set(Parameters,'Isotope',{'','','',''});
     close(handles.figure1);
end
set(findobj('Tag', 'textIso'), 'String', IsoLet); %Set text to selected
iso
set(findobj('Tag', 'exportbtn'), 'Enable', 'On'); %Enable the export
btn


function ChangeMAR(hObject)
%Change UI on selection of different MAR while saving selections in
%current MAR.
global Parameters
global CurrentMAR

MARStr = get(hObject, 'String');
SizeCheck = size(MARStr);

%If this assertion is thrown, check Radiobutton that is clicked for
having
%extra lines in its String property.
assert(SizeCheck(1) == 1,'MARStr does not have size 1xX, Property
Error.')

MARStr = MARStr(5); %Get MAR Number from Tag
MARNum = str2double(MARStr);

SaveMARData();

h1 = findobj('Tag', 'textIso'); %Get handle to the selected isotope
text
h2 = findobj('Tag', 'editBq'); %Get handle to quantity textbox.
CurrentMAR = MARNum; %Dont reset this value until end, so that the
                              %previously selected MAR is known.
set(h1,'String',Parameters.Isotope{CurrentMAR});
if Parameters.MAR(CurrentMAR) ~= 0
    set(h2,'String',num2str(Parameters.MAR(CurrentMAR)));
    set(h2,'Enable', 'on');
else
    set(h2,'Enable', 'off');
    set(h2,'String', 'Quantity (Bq)');
end


function result = SaveMARData()
```

```
global Parameters;
global CurrentMAR;

h1 = findobj('Tag', 'textIso'); %Get handle to the selected isotope
text
TempArray = get(Parameters,'Isotope');
if size(get(h1,'String'))~= 0 %If there is a selected isotope, save
that
    TempArray{CurrentMAR} = get(h1,'String'); %selection to Parameters.
end
set(Parameters,'Isotope',TempArray);          %^^

h2 = findobj('Tag', 'editBq'); %Get handle to quantity textbox.
TempArray = get(Parameters, 'MAR');
if ~strcmp(get(h2,'String'), 'Quantity (Bq)')
    A = str2double(get(h2,'String')); %This and below check that user
    if A >= 1 && A ~= inf                          %entered a numeric
value.
        TempArray(CurrentMAR) = str2double(get(h2,'String'));
        result = 1;
    elseif isnan(A)
        TempArray(CurrentMAR) = 0;
        if size(get(h2,'String')) == 0 %Set to zero if nothing entered
in quantity.
            if size(get(h1,'String'))~= 0 %Complain if isotope is
selected with no quantitiy.
                msgbox('MAR Quantity not entered. Data not saved.
Return to previous selection and try again.','MAR Quantity Error')
                result = 0;
            else
                result = 1;
            end
        else
            msgbox('MAR Quantity is not numeric. Data not saved. Return
to previous selection and try again.','MAR Quantity Error')
            result = 0;
        end
    elseif A <= 0 || A == inf
        TempArray(CurrentMAR) = 0;
        msgbox('MAR Quantity cannot be negative, zero, or infinite.
Data not saved. Return to previous selection and try again.','MAR
Quantity Error')
        result = 0;
    else
        result = 0;
    end
else
    if size(get(h1,'String'))~= 0 %Complain if isotope is selected with
no quantitiy.
```

```
        msgbox('MAR Quantity not entered. Data not saved. Return to
previous selection and try again.','MAR Quantity Error')
        result = 0;
    else
        result = 1;
    end
end

set(Parameters,'MAR',TempArray);
```

File 3, UserDefined.m:

```
function varargout = UserDefined(varargin)
% UserDefined MATLAB code for UserDefined.fig
%      UserDefined, by itself, creates a new UserDefined or raises the
existing
%      singleton*.
%
%      H = UserDefined returns the handle to a new UserDefined or the
handle to
%      the existing singleton*.
%
%      UserDefined('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in UserDefined.M with the given input
arguments.
%
%      UserDefined('Property','Value',...) creates a new UserDefined or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before UserDefined_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to UserDefined_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help UserDefined

% Last Modified by GUIDE v2.5 06-Jul-2016 14:18:33

% Begin initialization code - DO NOT EDIT
```

```
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @UserDefined_OpeningFcn, ...
                   'gui_OutputFcn',  @UserDefined_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% End initialization code - DO NOT EDIT
end


% --- Executes just before UserDefined is made visible.
function UserDefined_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to UserDefined (see VARARGIN)

% Choose default command line output for UserDefined
global Parameters
handles.output = hObject;
axes(handles.logo_axes);
imshow('sodaLogo1.png');
axes(handles.axes1);
plot(rand(1));
% Update handles structure
guidata(hObject, handles);

set(handles.Bin_1,'Enable','off');
set(handles.Bin_2,'Enable','off');
set(handles.Bin_3,'Enable','off');
set(handles.Bin_4,'Enable','off');
set(handles.Bin_5,'Enable','off');
set(handles.Bin_6,'Enable','off');
set(handles.Bin_7,'Enable','off');
set(handles.Bin_8,'Enable','off');
set(handles.Bin_9,'Enable','off');
set(handles.Bin_10,'Enable','off');
```

```
Parameters = SODA_Parameters();
if ~isempty(varargin)
    Parameters = varargin{1,1};
end
% UIWAIT makes UserDefined wait for user response (see UIRESUME)
 uiwait(handles.figure1);
end

% --- Outputs from this function are returned to the command line.
function varargout = UserDefined_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
global Parameters;
varargout{1} = Parameters;
delete(handles.figure1);
end

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1



% --- Executes during object creation, after setting all properties.
popup_sel_index = get(handles.popupmenu1, 'Value');
switch popup_sel_index
    case 1
        set(handles.Bins,'String','Number of Bins');
        set(handles.binWidth,'String','Bin Width');
        set(handles.CurrentTotal,'String','0');
        axes =(handles.axes1);
        cla reset;
        set(handles.Bin_1,'Enable','off');
        set(handles.Bin_2,'Enable','off');
        set(handles.Bin_3,'Enable','off');
        set(handles.Bin_4,'Enable','off');
        set(handles.Bin_5,'Enable','off');
```

```
        set(handles.Bin_6,'Enable','off');
        set(handles.Bin_7,'Enable','off');
        set(handles.Bin_8,'Enable','off');
        set(handles.Bin_9,'Enable','off');
        set(handles.Bin_10,'Enable','off');
    case 2
        set(handles.pushbutton1,'Enable','off');
        set(handles.pushbutton1,'String','Enter distribution below.');
        axes =(handles.axes1);
        cla reset;
        set(handles.Bins,'String','Number of Bins');
        set(handles.binWidth,'String','Bin Width');
        set(handles.CurrentTotal,'String','0');

end
end


function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end

set(hObject, 'String', {'Click to generate', 'Type distribution
values'});
end

function Bins_Callback(hObject, ~, handles)
% hObject    handle to Bins (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Bins as text
%        str2double(get(hObject,'String')) returns contents of Bins as
a double

% --- Executes during object creation, after setting all properties.
Bins = str2double(get(handles.Bins,'String'));
popup_sel_index = get(handles.popupmenu1, 'Value');
if Bins>10
```

```
    TenBins = 'The maximum number of bins allowed for this distribution
entry option is 10.';
    msgbox(TenBins);
    set(handles.Bins,'String','10');
elseif Bins<2
    msgbox('There must be at least 2 bins.','modal');
    set(handles.Bins,'String','1');
elseif round(Bins) ~= Bins
    msgbox('There must be an Integer quantity of bins.','modal');
    set(handles.Bins,'String','Number of Bins');
else
    switch popup_sel_index
    case 1
        set(handles.pushbutton1,'Enable','On');
        if ~isnan(str2double(get(handles.binWidth,'String')))
            set(handles.pushbutton1,'String','Start') ;
        end
    case 2
        set(handles.pushbutton1,'Enable','On');
        set(handles.pushbutton1,'String','Enter Values Below, Then
Click');
        handlesStructure=guihandles(gcf);
        for k=1:10
            % sprintf creates the strings Bin_1, Bin_2, etc.
            % handlesStructure.(x) retrieves the field x from the
handles structure
            h = handlesStructure.(sprintf('Bin_%d',k));
            set(h,'Enable','off');
        end
        for i=1:Bins
            % sprintf creates the strings Bin_1, Bin_2, etc.
            % handlesStructure.(x) retrieves the field x from the
handles structure
            h = handlesStructure.(sprintf('Bin_%d',i));
            set(h,'Enable','on')
        end
    end
end
end


function Bins_CreateFcn(hObject, ~, handles)
% hObject    handle to Bins (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

function binWidth_Callback(hObject, ~, handles)
% hObject    handle to binWidth (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of binWidth as text
%        str2double(get(hObject,'String')) returns contents of binWidth
as a double

popup_sel_index = get(handles.popupmenu1, 'Value');
Bins= str2double(get(handles.Bins,'String'));
switch popup_sel_index
    case 1
        set(handles.pushbutton1,'Enable','On');
        if ~isnan(str2double(get(handles.Bins,'String')))
            set(handles.pushbutton1,'String','Start') ;
        end
    case 2
        set(handles.pushbutton1,'Enable','On');
        set(handles.pushbutton1,'String','Enter Values Below, Then
Click');
        handlesStructure=guihandles(gcf);
        for k=1:10
            % sprintf creates the strings Bin_1, Bin_2, etc.
            % handlesStructure.(x) retrieves the field x from the
handles structure
            h = handlesStructure.(sprintf('Bin_%d',k));
            set(h,'Enable','off');
        end
        for i=1:Bins
            % sprintf creates the strings Bin_1, Bin_2, etc.
            % handlesStructure.(x) retrieves the field x from the
handles structure
            h = handlesStructure.(sprintf('Bin_%d',i));
            set(h,'Enable','on')
        end
end

end


% --- Executes during object creation, after setting all properties.
function binWidth_CreateFcn(hObject, ~, handles)
```

```
% hObject    handle to binWidth (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

function Bin_1_Callback(hObject, eventdata, handles)
% hObject    handle to Bin_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Bin_1 as text
%        str2double(get(hObject,'String')) returns contents of Bin_1 as
a double
end

% --- Executes during object creation, after setting all properties.
function Bin_1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Bin_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end



function Bin_2_Callback(hObject, eventdata, handles)
% hObject    handle to Bin_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Bin_2 as text
%        str2double(get(hObject,'String')) returns contents of Bin_2 as
a double
end
```

```
% --- Executes during object creation, after setting all properties.
function Bin_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Bin_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end



function Bin_3_Callback(hObject, eventdata, handles)
% hObject    handle to Bin_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Bin_3 as text
%        str2double(get(hObject,'String')) returns contents of Bin_3 as
a double
end

% --- Executes during object creation, after setting all properties.
function Bin_3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Bin_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function Bin_4_Callback(hObject, eventdata, handles)
% hObject    handle to Bin_4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of Bin_4 as text
%        str2double(get(hObject,'String')) returns contents of Bin_4 as
a double
end

% --- Executes during object creation, after setting all properties.
function Bin_4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Bin_4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function Bin_5_Callback(hObject, eventdata, handles)
% hObject    handle to Bin_5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Bin_5 as text
%        str2double(get(hObject,'String')) returns contents of Bin_5 as
a double
end

% --- Executes during object creation, after setting all properties.
function Bin_5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Bin_5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

end

function Bin_6_Callback(hObject, eventdata, handles)
% hObject    handle to Bin_6 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Bin_6 as text
%        str2double(get(hObject,'String')) returns contents of Bin_6 as
a double
end

% --- Executes during object creation, after setting all properties.
function Bin_6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Bin_6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

end

function Bin_7_Callback(hObject, eventdata, handles)
% hObject    handle to Bin_7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Bin_7 as text
%        str2double(get(hObject,'String')) returns contents of Bin_7 as
a double

end
% --- Executes during object creation, after setting all properties.
function Bin_7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Bin_7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end
```

```
function Bin_9_Callback(hObject, eventdata, handles)
% hObject    handle to Bin_9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Bin_9 as text
%        str2double(get(hObject,'String')) returns contents of Bin_9 as
a double


% --- Executes during object creation, after setting all properties.
end

function Bin_8_Callback(hObject, eventdata, handles)
% hObject    handle to Bin_8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Bin_8 as text
%        str2double(get(hObject,'String')) returns contents of Bin_8 as
a double
end

% --- Executes during object creation, after setting all properties.
function Bin_8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Bin_8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

function Bin_9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Bin_9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end
end

function Bin_10_Callback(hObject, eventdata, handles)
% hObject    handle to Bin_10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Bin_10 as text
%        str2double(get(hObject,'String')) returns contents of Bin_10
as a double
end

% --- Executes during object creation, after setting all properties.
function Bin_10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Bin_10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, ~, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Parameters;
if checkInput(handles)
    axes(handles.axes1);
    cla;
    popup_sel_index = get(handles.popupmenu1, 'Value');
    Bins=str2double(get(handles.Bins,'String'));
    Width=str2double(get(handles.binWidth,'String'));
    switch popup_sel_index
        case 1
            % clicked user defined distribution
            set(hObject,'Enable','off');
            Distance = Bins*Width;
            i=1;
            while i<5                                %sets up plot axes
for user to click (allows 4 chances)
                k=1;
```

```
                y1 = zeros(1,Bins);
                %x1 = linspace(0,Bins+1,Bins);
                while k<Bins+1
                    x1 = linspace(0,Distance-Distance/Bins,Bins);
                    grid on
                    ax = gca;
                    ax.XLim = [0,Distance];
                    ax.YLim = [0,1];
                    ax.XTick = x1;
                    ax.YTick =
[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0];
                    [x,y]= ginput(1);  %takes user clicks as data
points
                    refresh;

                    if y>1 || y<0 || x>Distance || x<0
                        OutOfBounds='You must click on the axes. Please
try again.';
                        uiwait(msgbox(OutOfBounds,'modal'));
                    else
                        y=round(y*20)/20;
                        y1(k) = y;
                        plot1 = bar(x1,y1,'histc');
                        set(handles.CurrentTotal,'String',sum(y1));
                        k=k+1;
                        pause(0.1);
                    end
                end
                ProbTotal=round(sum(y1),2);
%sums the probabilities
                if ProbTotal~=round(str2double('1'),2)
%check if probabilities sum to 1
                    if i>3
% if no, and max tries, return to main page
                        MaxTries='Maximum number of tries has been met.
You will now return to main page';
                        uiwait(msgbox(MaxTries,'modal'));
                        close(gcf);
                        i=i+1;
                    else
                        msg='The probabilities must sum to one. Please
try again.';     %if no, and not max tries, try again
                        uiwait(msgbox(msg,'modal'));
                        axes(handles.axes1);
                        cla reset;
                        set(handles.CurrentTotal,'String','0');
                        i=i+1;
                    end
                else
```

```
                    plot1 = bar(x1,y1,'histc');
% if sum to 1, show plot and ask if correct
                    ylim([0,1])
                    YesNo = questdlg('Does this look correct?', ...
                        'Check Distribution', ...
                        'Yes','No','No');
                    % Handle response
                    switch YesNo
                        case 'Yes'
% if yes, close plot and set distribution
                            i = 5;
                            set(Parameters,'UDtempY', y1);
                            set(Parameters,'UDtempX', x1);
                            close(handles.figure1);
                            return;
                        case 'No'
% if no, try again
                            TryAgain='Please try again.';
                            uiwait(msgbox(TryAgain,'modal'));
                            axes(handles.axes1);
                            cla reset;
                            set(handles.CurrentTotal,'String','0');
                            i=i+1;
                    end
                    set(hObject,'enable','on');
                end
            end
        case 2
            set(handles.pushbutton1,'Enable','on');
            Distance = Bins*Width ;
%calculate the total distance using bins*width
            %i=1;
            axes(handles.axes1);
            cla reset;
            Probability = zeros(1,Bins);
% set up array for probabilities
            x1 = linspace(0,Distance-Distance/Bins,Bins);
            handlesStructure=guihandles(gcf);
            set(handles.CurrentTotal,'String', '0');
            for i=1:Bins
                % sprintf creates the strings edit1, edit2, etc.
                % handlesStructure.(x) retrieves the field x from the
handles structure
                h = handlesStructure.(sprintf('Bin_%d',i));
                Probability(i) = (str2double(get(h,'String')));
                Total = sum(Probability);

set(handles.CurrentTotal,'String',(num2str(sum(Probability))))
            end
```

```
            TotalProb = round(Total,3);                             %sum
them to see if they equal 1
            if TotalProb~=round(str2double('1'),3)
                msg='The probabilities must sum to one. Please try
again.';     % if sum does not equal 1, error message
                uiwait(msgbox((msg)));
                cla reset;
                set(handles.CurrentTotal,'String','0');
            else
                bar(x1,Probability,'histc');
%show plot of probabilities entered by user
                YesNo = questdlg('Does this look correct?', ...
%ask if this is the correct distribution
                    'Check Distribution', ...
                    'Yes','No','No');
                % Handle response
                switch YesNo
                    case 'Yes'
                        % if yes, set distribution and close plot
                        set(Parameters,'UDtempY', Probability);
                        set(Parameters,'UDtempX', x1);
                        close(handles.figure1);
                        return;
                    case 'No'
                        TryAgain='Please try again.';       %if no, try
again, close plot
                        uiwait(msgbox((TryAgain)));
                        cla reset;
                        set(handles.CurrentTotal,'String','0');
                end
            end



    end
    set(hObject,'enable','on');
else
    set(hObject,'enable','on');
    return
end
end

function result = checkInput(handles) % Check bins and Bin width before
allowing the pushbutton routine to run.
Bins = str2double(get(handles.Bins,'String'));
Width = str2double(get(handles.binWidth,'String'));
    if ~isnan(Bins) && ~isnan(Width)
        if Bins <= 0 || Bins > 10
```

```
            msgbox('Number of Bins must be an integer between 1 and
10.','Input Error');
            result = 0;
        elseif Width <= 0 || Width == inf
            msgbox('Bin Width must be a non negative, non infinite, non
zero value.','Input Error');
            result = 0;
        else
            result = 1;
        end

    else
        msgbox('Number of Bins and Bin Width must be specified.','Input
Error');
        result = 0;
    end
end



% --------------------------------------------------------------------
function FileMenu_Callback(hObject, eventdata, handles)
% hObject    handle to FileMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end

% --------------------------------------------------------------------
function OpenMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to OpenMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
file = uigetfile('*.fig');
if ~isequal(file, 0)
    open(file);
end
end

% --------------------------------------------------------------------
function PrintMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to PrintMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
printdlg(handles.figure1)
end

% --------------------------------------------------------------------
function CloseMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to CloseMenuItem (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
selection = questdlg(['Close ' get(handles.figure1,'Name') '?'],...
                     ['Close ' get(handles.figure1,'Name') '...'],...
                     'Yes','No','Yes');
if strcmp(selection,'No')
    return;
end

delete(handles.figure1)
end



function CurrentTotal_Callback(hObject, eventdata, handles)
% hObject    handle to CurrentTotal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of CurrentTotal as text
%        str2double(get(hObject,'String')) returns contents of
CurrentTotal as a double
end

% --- Executes during object creation, after setting all properties.
function CurrentTotal_CreateFcn(hObject, eventdata, handles)
% hObject    handle to CurrentTotal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


% --- Executes during object creation, after setting all properties.
function pushbutton1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
end
```

```
% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if isequal(get(hObject, 'waitstatus'), 'waiting')
    % The GUI is still in UIWAIT, us UIRESUME
    uiresume(hObject);
else
    % The GUI is no longer waiting, just close it
    delete(hObject);
end
end
```

File 4, SODA_Parameters.m:

```
%{
Data Class for SODA, holds user parameters for each MAR, and includes
 methods to sum the individual distributions. Intended to simplify data
 handling between the 4 available MAR selections, and across the three
 forms used in SODA.
%}

classdef SODA_Parameters < matlab.mixin.SetGet %Allow class to inherit
    properties                                 %MATLAB get and set
methods.
        MAR = [0,0,0,0]
        MAR2 = [0,0,0,0] %2 versions are for the second parameter slot
        DCF = [0,0,0,0]  %on SODA, while the unnumbered version is for
        DCF2 = [0,0,0,0] %the first box. (EX: mean or single point)
        ARF = [0,0,0,0]
        ARF2 = [0,0,0,0]
        RF = [0,0,0,0]
        RF2 = [0,0,0,0]
        Isotope = {'','','',''}
        MARdist = {'','','',''}
        ARFdist = {'','','',''}
        DCFdist = {'','','',''}
        RFdist = {'','','',''}
        AvgCED = [0,0,0,0] %Average CED for each individual MAR
        StdCED = [0,0,0,0] %STDev for each MAR
        MedCED = [0,0,0,0] %Median for each CED
        Ninty_fifth = [0,0,0,0] % ninty fifth Percentile

        XResult1    %Hold graphable results for each MAR to allow for
display of individual MAR result
        XResult2    %Under class methods, and results persistance
between
```

```
        XResult3    %MAR selections when plot is set to MAR[x], not
total.
        XResult4    %These are the xi, and fi results from previous
methods
        YResult1
        YResult2
        YResult3
        YResult4
        CED1
        CED2
        CED3
        CED4
        SumX
        SumY
        SumCED
        SumAvgCED = 0
        SumStdCED = 0
        Sum95CI = 0
        SumMed = 0
        UDtempX     %Temp storage for UDD data coming from the UDD gui.
        UDtempY
        UDD1 = UDDData(); %Mar Specific User defined data for mar 1
        UDD2 = UDDData(); %" " for mar 2
        UDD3 = UDDData(); %etc
        UDD4 = UDDData();
        UDDRX  %User defined data for non mar specific data
        UDDRY
        UDLPFX
        UDLPFY
    end
    methods
        function obj = SumFinal(obj)
            obj.SumCED = obj.CED1 + obj.CED2 + obj.CED3 + obj.CED4;
%Sum indivdual CED distributions
            clearvars CED1 CED2 CED3 CED4;
            obj.SumAvgCED = mean(obj.SumCED);
            obj.SumStdCED = std(obj.SumCED);
            obj.SumMed = median(obj.SumCED);
            obj.Sum95CI = prctile(obj.SumCED,95);
            nbins = max(min(length(obj.SumCED)./10,100),50); %Break
domain up into 100 "bins"
            obj.SumX = linspace(min(obj.SumCED),max(obj.SumCED),nbins);
%Draw a linespace over the range

            dx = mean(diff(obj.SumX));
            obj.SumY = histc(obj.SumCED,obj.SumX-dx); %Count the number
of ced's between a point in xi
                                                    %and the next point.
            obj.SumY = obj.SumY./sum(obj.SumY)./dx;
```

```
        end

        function obj = EnsureDataIntegrity(obj) % A check used in
MAR_Selection
            for i = 1:4
                if obj.MAR(i) == 0 || obj.DCF(i) == 0
                    obj.MAR(i) = 0;
                    obj.DCF(i) = 0;
                    obj.Isotope{i} = '';
                end
            end
        end

        function [obj, msg, flag] = CheckUDD(obj, Param)  %Check a UDD
entry for validity.
            if any(obj.UDtempX) && any(obj.UDtempY)
                step = obj.UDtempX(2) - obj.UDtempX(1);
                s = size(obj.UDtempX);
                s = s(2);
                if ~strcmp(Param, 'DCF')
                    if (obj.UDtempX(s) + step) > 1
                        msg = 'A user defined distribution for a 0 to 1
parameter may not specify a nonzero probability at values greater than
1.';
                        flag = 1;
                    else
                        msg = '';
                        flag = 0;
                    end
                else
                    if (obj.UDtempX(s) + step) > 0.001
                        msg = 'A user defined distribution for DCF may
not specify a nonzero probability at values greater than 1E-3.';
                        flag = 1;
                    else
                        msg = '';
                        flag = 0;
                    end
                end
                return;
            else
                msg = 'Distribution Information not entered. Please try
again.';
                flag = 1;
            end
        end
```

```
        function obj = SaveUDD(obj, CurrentMAR, Param) % Saving user
defined data after entry
            if strcmp(Param, 'DR')
                obj.UDDRX = obj.UDtempX;
                obj.UDDRY = obj.UDtempY;
            elseif strcmp(Param,'LPF')
                obj.UDLPFX = obj.UDtempX;
                obj.UDLPFY = obj.UDtempY;
            else
                switch CurrentMAR
                    case 1
                        obj.UDD1.Save(Param,obj.UDtempX,obj.UDtempY);
                    case 2
                        obj.UDD2.Save(Param,obj.UDtempX,obj.UDtempY);
                    case 3
                        obj.UDD3.Save(Param,obj.UDtempX,obj.UDtempY);
                    case 4
                        obj.UDD4.Save(Param,obj.UDtempX,obj.UDtempY);
                end

            end
            obj.UDtempX = 0;
            obj.UDtempY = 0;
        end
        function [obj,X,Y] = GetUDD(obj,CurrentMAR,Param) %Recall User
defined data
            if strcmp(Param, 'DR')
                X = obj.UDDRX;
                Y = obj.UDDRY;
            elseif strcmp(Param,'LPF')
                X = obj.UDLPFX;
                Y = obj.UDLPFY;
            else
                switch CurrentMAR
                    case 1
                        [obj.UDD1,X,Y] = Recall(obj.UDD1,Param);
                    case 2
                        [obj.UDD2,X,Y] = Recall(obj.UDD2,Param);
                    case 3
                        [obj.UDD3,X,Y] = Recall(obj.UDD3,Param);
                    case 4
                        [obj.UDD4,X,Y] = Recall(obj.UDD4,Param);
                end

            end
            return;
        end
    end
end
```

File 5, UDDData.m:

```
%{
Data Class for SODA, holds user defined values for each possible dist
and
MAR selection. Includes methods to facilitate easy access and setting
of
UDD data, in conjunction with methods in SODA_Parameters. Does not
include
selections which remain consistant across mar selections, namely DR and
LPF
%}

classdef UDDData < handle %Specify handle class
    properties
        ARFX = []
        ARFY = []
        RFX = []
        RFY = []
        DCFX = []
        DCFY = []
    end
    methods
         function obj = Save(obj,Param,UDtempX,UDtempY)
             switch Param
                 case 'ARF'
                     obj.ARFX = UDtempX;
                     obj.ARFY = UDtempY;
                 case 'RF'
                     obj.RFX = UDtempX;
                     obj.RFY = UDtempY;
                 case 'DCF'
                     obj.DCFX = UDtempX;
                     obj.DCFY = UDtempY;
             end
         end
         function [obj,X,Y] = Recall(obj,Param)
              switch Param
                 case 'ARF'
                     X = obj.ARFX;
                     Y = obj.ARFY;
                 case 'RF'
                     X = obj.RFX;
                     Y = obj.RFY;
                 case 'DCF'
                     X = obj.DCFX;
                     Y = obj.DCFY;
              end
             return;
```

```
        end
    end

end
```

Appendix B: SODA Raw Changelog from GitLab Commit Messages

SODA Commit Log: Begins October 2015, Ends September 2016.

1. Added MAR Gui, Set up visuals, back end to come later. Small alterations to SODA gui, to align and match sizes of some text boxes.
2. Renamed GUI Objects for Easy ID, Added Text fields and List fields, as well as export button.
3. Added Dummy Function which fills text in isotope boxes. This will become the actual function once the database is in place. Also added a function which finds the element number of the button selected, and passes this to the other function, the one filling the isotope boxes. This is now called whenever any element button is pressed.
4. Added V1 of Database File. (Used Medium values for all DCF initially) Added check on startup to determine which elements have data, and disable those which do not. Disabled boxes have their color changed to dark gray. Modified function which was setting isotope boxes to actually lookup available isotopes and put them into the isotope selection boxes.
5. MAR Selection Button added to SODA, Location is pending group approval. MAR Selection tool now at basic functionality level. Isotopes can be selected and their quantity and DCF is exported. Tool to be presented at next SODA meeting for group feedback and aesthetic suggestions.
6. Adjusted DCF Values in Database file. Using Adult Inhalation Figures as before, but now using highest value, rather than one absorption rate.
7. Added MAR Selection Radiobuttons, grouped for mutual exclusivity. Detail work on MAR Selection UI, including instructions, SODA logo, and some minor relocating. Detail work on SODA Main, including resize and reposition of main plot, to ensure that nothing is clipping, and sufficient real estate remains. Added SODA main radiobuttons, identical to those in MAR select.
8. Added Comments in MAR_Selection Fixed incorrect truncation in ARF/RF. Added SODA_Parameters class, acting as data struct to simplify the multiple isotope case. (WIP) Changed Title Text position on plots in SODA. (Centered and Standardized across the different distributions)
9. Made Major Changes to MAR_Selection, adding all the nessesary functionality to change to different MAR's and preserve data, and switched all data handling and

saving to the Parameters obj of SODA_Parameters class. SODA_Parameters class modified, keeping more data and added a method to ensure that no incomplete data entries remain before transfering the obj back to SODA Main. As of this Commit, MAR_Selection is in a working state for multiple isotopes. Work to continue on SODA Main.

10. Changes to Soda, and SODA_Parameters to allow for multiple isotope handling. At this point, the program is capable of getting a result for multiple isotopes. Very little condition handling is in place. Functionality somewhat limited as a result. I consider this "limping." Additional condition handling and other detail work to come.
11. Changed Method for Calculating final distribution for multiple isotopes.
12. Fit Distribution now works on the output of the multi isotope run.
13. Added functionality so that mean std etc in a text box will not trigger an error on marstate change. Also added functionality to restore these tips in the text boxes after a change. So if there is no data saved, and a distribution type is selected, the text fields will populate appropriately.
14. Old Isotope selections removed from DCF Single input. Now, clicking Select Isotope in the DCF dropdown takes the user to MAR Selection. Also, after selecting MAR, the selected isotope shows up on the DCF dropdown while in single input mode. Single input mode is now default for DCF. Selection of distribution input works as before, including loading the dcf from MAR selection. Run All now has many checks integrated to ensure that no critical errors occur. Try and Catch statements incorporated to catch errors and report back without crippling the program, or causing consistency errors. The changes in Run All include the ability to send empty CED arrays to the class when an entry is incomplete. This allows the class method to sum the final distribution to work if even 1 MAR has correct input parameters. In that case, the final distribution will be the same as it would be if ran in single MAR mode. The program now waits for the user to click ok when error displays or msgboxes display, to avoid graphical errors. To further eliminate this issue, when plotting commands are issued, they now issue to the specific set of axes that we want to graph to. GetResults now checks both whether ARF/RF are above zero, and below or equal to 1. Program should be fairly usable in this state. Improvements will occur at a later time, presumably, but the core functions now work and the MAR Expansion can be officially considered complete to base functionality requirements.
15. Quick fix, when in single input mode, the program now clears junk from the second text box on the saved parameters rather than saving the junk.
16. KUSHAL BHATTARI: Added Log Normal and Changed CED Plot Title (95%CI etc)

17. Added Show Plot Functionality to parameter distributions for the Log Normal case. Added log normal option to DCF distribution select. Updated the multi mar result printout to match the changes to the single distribution changes (Median and 95 CI values.) Added multimar median and 95CI calculations to the class method to facilitate this.
18. Merge branch 'AMaas' into 'master' AMaas Given the completion of the Multi-Mar feature set, and the verification of a functional installed version, this code revision is ready to be written back to master.
19. Minor Bug Fixes
20. Removed Log Normal from MAR and DCF, pending review. Added Log Normal to GetResults, so that a log normal distribution may be used, and a CED result can be given.
21. Some Features In Progress. Added Check routine in parameter specific show plot routine for mean (or single value) input for DR, ARF, RF, and LPF. Signed-off-by: AMaas <maas.andrew@gmail.com>
22. Work in Progress Performance improvements to the GetResults function Changed Lognormal input to mode and scale parameter Changed reporting of lognormal distribution to avoid confusion with location and scale parameters. Undid first iteration of parameter control feature, needs different implementation.
23. Adjusted Parameter control function in prep for implementation some details still undecided at this point. Fix in MAR Selection, Scientific notation can now be used in the MAR quantity while still controlling the input conditions. New method had to be used.
24. Changed how the programs shuts off and turns on buttons and text fields while running a CED calc. Under new scheme, program will only turn on these fields if they were enabled before running, and will only turn on fit dist if there is a good result. Signed-off-by: AMaas <maas.andrew@gmail.com>
25. MARY TOSTEN: User Defined Distribution GUI file added.
26. Bug fixes, User def GUI rescale, and first pass.
27. Input parameter checking added on button click to ensure correct inputs for bins and bin width prior to attempting to take clicked or manual input. Similar control function in progress for manual entry. Bug fixes for incorrect reporting of sum not being equal to 1. Signed-off-by: AMaas <maas.andrew@gmail.com>
28. Changed minimum bins to 2, since a single bin has only 1 possible distribution that meets the requirements. Made feedback messages modal, so they cannot be clicked off of until dismissed. Changed bar graph to 'histc' style, so that the bars cover intervals, rather than values. Adjustments were made to x1 to accommodate this. Added specific x1 to case 2 plot, so that the displayed interval on the plot corresponds to the actual interval. Adjusted some text for clarity. Disabled Main

Button while ginput is active, to avoid nesting and extra cross hairs appearing when misclicks occur. Signed-off-by: AMaas <maas.andrew@gmail.com>

29. Established data handling between User Defined and Soda main. Introduced class method to save User defined to correct property in the class. Now passing Parameters to User defined, added UIwait to allow for results to be created before return, in the same manner that was used in the MAR selection GUI. Signed-off-by: AMaas <maas.andrew@gmail.com>
30. Added a new class to hold user defined distribution data, and added one instance of this new class for each MAR to the SODA_Parameters class. Added associated methods to get and set UDD data from the inner class, and from SODA_Parameters itself. Data is now saved after setting a UDD. Need to add checks to the UDD data, to make sure it is "good" before saving and add the UDD option to the change MAR stuff etc. Then, allow for the user to see the UDD they made in SODA, and finally, allow the CED calcs to actually use the UDD as a PDF. Signed-off-by: AMaas <maas.andrew@gmail.com>
31. SODA now tracks distribution selection, or single value selection, for each parameter that can be selected independently for each MAR. This change is critical for the UDD, since a UDD for one MAR may not be correct for another MAR. Previously, the distribution selection did not change, but the dist. parameters were saved and changed. The new functionality is, in general, superior, while also being necessitated by the UDD feature. This new functionality needs to be applied with respect to the select MAR feature before it is complete. Signed-off-by: AMaas <maas.andrew@gmail.com>
32. Added the ability to plot a User Defined Distribution. There is no error checking or parameter control in this feature yet, but it will work if the UDD is okay. The plotting algorithm uses the UDD as a pdf and generates random numbers, as opposed to just plotting the points that the user entered in the UDD GUI. This same method will be used to add UDD to the getResults function in the near future. Signed-off-by: AMaas <maas.andrew@gmail.com>
33. User Defined case added for all parameters that we allow, in the get result function. This allows a CED to be calculated when a user defined distribution is selected. Currently not computationally efficient. Hoping to correct with parallel toolbox. Will still remain slow on low core count machines. Advantage to current algorithm is that it works on any selection of a UDD that we allow without excessive use of case statements. Signed-off-by: AMaas <maas.andrew@gmail.com>
34. Good deal of changes. Unhandled cases with select mar were corrected. Select mar no longer overrides a user defined distribution in DCF. User Defined distributions are now checked for unreasonable selections. If an unreasonable selection is made (non-zero probability above 1 for arf for example) the UDD is

not saved, and the user must try again. Parameter control has been implemented in the show plot feature for individual parameters, parameter control when calculating CED will be added next. Several small bug fixes, and additional comments. Signed-off-by: AMaas <maas.andrew@gmail.com>

35. Parameter control implemented for CED calculation, both single and multi MAR. Chi/Q and BR are not checked. Parameter control, with exception to Chi/Q and BR, is fully implemented. This build will be distributed to team members for QA. Signed-off-by: AMaas <maas.andrew@gmail.com>
36. Fixed a bug which would not allow the user to enter a lower limit of 1 for a uniform distribution. Fixed a bug in the user defined dist. where if the UDD GUI was closed before anything was entered, the program would have an unhandled crash. Fixed an additional bug which allowed a subsequent closure of the UDD GUI before input, after a successful entry was made previously, to duplicate the previous distribution, rather than resetting and not saving. Signed-off-by: Andrew Maas <maas.andrew@gmail.com>
37. Changed how SODA saves workspace data, so that this feature is compatible with multi-mar, and the user defined distribution. The class object is now saved in the file. This allows for all data that is needed to be saved and recalled. It should be noted that the file size could become quite large if a large sample count was used, since result data is being saved as well. Need to make a note in the user's manual about this. Signed-off-by: Andrew Maas <maas.andrew@gmail.com>
38. Changed soda show plot button and show all button routines. Once the large results array is done being used in the parameters object, as in, it has been saved in the workspace variable, the copy of the results array is removed from parameters. This keeps the program from keeping unnecessary RAM tied up, and, prevents a workspace save from having issues. It was found that the workspace file will not save the parameter object data when a very large array is present in the object. In addition, the file load function received a bug fix which was causing user defined distribution and Log normal selections in DR and LPF to not load correctly. Signed-off-by: Andrew Maas <maas.andrew@gmail.com>
39. Changed image save routine, so that the final image is of a modern aspect ratio, larger resolution, and able to fit all of the text. The inability to fit all the text was introduced when our output text on a plot changed. Feature was tested in MATLAB, and in the compiled version. Checked zoom and drag features, all seems to be working. Signed-off-by: Andrew Maas <maas.andrew@gmail.com>
40. Platform specific code added for help file opening. This should allow the complied version to work on both Mac and Windows. Additional comments added in several files. New help file replaces old help file. This version of the code base is considered frozen, and is included in the SODA final report. Signed-off-by: Andrew Maas <maas.andrew@gmail.com>

41. Adding INL Logo to files. Can updated, removed DOE logo, added INL logo. About Updated, removed DOE logo, added INL logo. Help file updated, new figures with updated can, and correction of typo in the chi over Q sensitivity discussion, per Jason's comments. Signed-off-by: Andrew Maas <maas.andrew@gmail.com>
42. Fixed critical error in GetResults. When using the UDD option, indexing variable e was overwriting the variable e which was used to keep track of whether an input distribution was single value or distribution input. Changing the index variable to ee from e solved the issue. Signed-off-by: Andrew Maas <maas.andrew@gmail.com>
    Final Commit on October 23, 2016.

Appendix C: RF Parameter Distribution Source Code

```
function varargout = RFGUI(varargin)
% RFGUI MATLAB code for RFGUI.fig
%      RFGUI, by itself, creates a new RFGUI or raises the existing
%      singleton*.
%
%      H = RFGUI returns the handle to a new RFGUI or the handle to
%      the existing singleton*.
%
%      RFGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in RFGUI.M with the given input
arguments.
%
%      RFGUI('Property','Value',...) creates a new RFGUI or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before RFGUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to RFGUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help RFGUI

% Last Modified by GUIDE v2.5 11-Oct-2016 13:57:38

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @RFGUI_OpeningFcn, ...
                   'gui_OutputFcn',  @RFGUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before RFGUI is made visible.
function RFGUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to RFGUI (see VARARGIN)

% Choose default command line output for RFGUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
p = gcp('nocreate');
if isempty(p)
    parpool(7);
end
% UIWAIT makes RFGUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = RFGUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function editMode_Callback(hObject, eventdata, handles)
% hObject    handle to editMode (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editMode as text
%        str2double(get(hObject,'String')) returns contents of editMode
as a double


% --- Executes during object creation, after setting all properties.
function editMode_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editMode (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editIn1_Callback(hObject, eventdata, handles)
% hObject    handle to editIn1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editIn1 as text
%        str2double(get(hObject,'String')) returns contents of editIn1
as a double
```

```
% --- Executes during object creation, after setting all properties.
function editIn1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editIn1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editIn2_Callback(hObject, eventdata, handles)
% hObject    handle to editIn2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editIn2 as text
%        str2double(get(hObject,'String')) returns contents of editIn2
as a double


% --- Executes during object creation, after setting all properties.
function editIn2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editIn2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in plotSPbutton.
function plotSPbutton_Callback(hObject, eventdata, handles)
% hObject    handle to plotSPbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sample = get(handles.editSamples,'String');%grab number from number of
sample box
samplesize = str2double(sample);
```

```
if isnan(samplesize)
   msgbox('Samples');
   return
end
contents = get(handles.popupmenu1,'String');
popupmenuvalue = contents{get(handles.popupmenu1,'Value')};
num1 = str2double(get(handles.editIn1,'string'));
num2 = str2double(get(handles.editIn2,'string'));
axes(handles.axes1)
switch popupmenuvalue
    case 'Normal'
        pd = makedist('Normal','mu',num1,'sigma',num2);
        t = truncate(pd,0,inf);
        %t = pd;
        n = random(t,samplesize,1);
        nbins = max(min(length(n)./10,100),50);
        xi = linspace(min(n),max(n),nbins);
        dx = mean(diff(xi));
        fi = histc(n,xi-dx);
        fi = fi./sum(fi)./dx;
        assignin('base','SP', n);
        %assignin('base','rffi2', fi);
        bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
        axis tight;
        %         hist(n,50);
        ylabel('Probability Density');
        xlabel('Scale Param');
        str = sprintf('\\fontsize{10} Scale Param distribution plot
with Normal distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
            mean(n),std(n));
        title(str,'Units', 'normalized', ...
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')
    case 'Lognormal'
        pd = makedist('Lognormal','mu',log(num1),'sigma',num2);
        t = truncate(pd,0,inf);
        %t = pd;
        n = random(t,samplesize,1);
        nbins = max(min(length(n)./10,100),50);
        xi = linspace(min(n),max(n),nbins);
        dx = mean(diff(xi));
        fi = histc(n,xi-dx);
        fi = fi./sum(fi)./dx;
        assignin('base','SP', n);
        %assignin('base','drfi2', fi);
        bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
        axis tight;
        %         hist(n,50);
```

```
        axis tight;
        ylabel('Probability Density');
        xlabel('Scale Param');
        str = sprintf('\\fontsize{10} Scale Param distribution plot
with Log Normal distribution with Mean=%0.2e , Stdev=%0.2e',...
            mean(n),std(n));
        title(str,'Units', 'normalized', ...
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')

    case 'Uniform'
        if num1 < num2;
            % In unifrom distribution upper limt must be greater than
lower
            % limit, if not show the error message
            errordlg('Upper Limit is less than lower limt','Uniform
Distribution','modal')
            return;
        else
            pd = makedist('Uniform','Upper',num1,'Lower',num2);
            t = truncate(pd,0,inf);
            %t = pd;
            n = random(t,samplesize,1);
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','SP', n);
            %assignin('base','rffi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            % %             hist(n,50);

            ylabel('Probability Density');
            xlabel('Scale Param');
           str = sprintf('\\fontsize{10} Scale Param distribution plot
with Uniform distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
            mean(n),std(n));
        title(str,'Units', 'normalized', ...
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')
        end

    case 'Single Value'
        assignin('base','SP', num1);
        cla(handles.axes1);
        return

end
```

```
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function editSamples_Callback(hObject, eventdata, handles)
% hObject    handle to editSamples (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editSamples as text
%        str2double(get(hObject,'String')) returns contents of
editSamples as a double


% --- Executes during object creation, after setting all properties.
function editSamples_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editSamples (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in plotParticle.
function plotParticle_Callback(hObject, eventdata, handles)
% hObject    handle to plotParticle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sample = get(handles.editSamples,'String');%grab number from number of
sample box
samplesize = str2double(sample);
intervl = get(handles.editIntervals,'String');
intervals = str2double(intervl);
if isnan(samplesize)
   msgbox('Samples');
   return
end
```

```
plotMEDIANbutton_Callback(handles.plotMEDIANbutton, 1 ,handles);
plotSPbutton_Callback(handles.plotSPbutton, 1 ,handles);
axes(handles.axes1);
MediaN = evalin('base','MediaN');
sp = evalin('base','SP');
RFDist = zeros(1,samplesize);
tic;
parfor i=1:length(RFDist)
    if size(MediaN) == 1
        num1 = MediaN;
    else
        num1 = MediaN(i);
    end
    if size(sp) == 1
        num2 = sp;
    else
        num2 = sp(i);
    end
    pd = makedist('Lognormal','mu',log(num1),'sigma',num2);
    t = truncate(pd,0,inf);
    %t = pd;
    n = random(t,samplesize,1);
    n = n./2;
    n = n.^3;
    xi = linspace(min(n),max(n),intervals);
    dx = mean(diff(xi));
    fi = histc(n,xi-dx);
    fi = fi./sum(fi)./dx;
    low = 1;
    high = intervals;
    lowisntset = 1;
    highisntset = 1;
    if (min(n) <= ((1e-6)/2)^3) && (max(n) >= ((1e-5)/2)^3)
        for j=1:intervals
            if (xi(j) >= ((1e-6)/2)^3) && (lowisntset == 1)
                low = j;
                lowisntset = 0;
            end
            if (xi(j) >= ((1e-5)/2)^3) && (highisntset == 1)
                high = j;
                highisntset = 0;
            end
        end
        fitrunct = fi(low:high);
        RFDist(i) = trapz(fitrunct)/trapz(fi);
    elseif (min(n) > ((1e-5)/2)^3) || (max(n) < ((1e-6)/2)^3)
        RFDist(i) = 0;
        disp([num1,num2]);
        disp('Med. Scale');
```

```
    elseif min(n) > ((1e-6)/2)^3
        for j=1:intervals
            if (xi(j) >= ((1e-5)/2)^3) && (highisntset == 1)
                high = j;
                highisntset = 0;
            end
        end
        low = 1;
        fitrunct = fi(low:high);
        RFDist(i) = trapz(fitrunct)/trapz(fi);
    elseif max(n) < ((1e-5)/2)^3
        for j=1:intervals
            if (xi(j) >= ((1e-6)/2)^3) && (lowisntset == 1)
                low = j;
                lowisntset = 0;
            end
        end
        high = intervals;
        fitrunct = fi(low:high);
        RFDist(i) = trapz(fitrunct)/trapz(fi);
    end
    %assignin('base','MediaN', n);
    %assignin('base','drfi2', fi);
end
binselect = [100, intervals/10];
%nbins = min(max(binselect),10000);
nbins = 200;
xi2 = linspace(min(RFDist),max(RFDist),nbins);
dx2 = xi2(2)-xi2(1);
fi2 = histc(RFDist,xi2-dx2);
fi2 = fi2./sum(fi2)./dx2;
bar(xi2,fi2,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
assignin('base','RFDist', RFDist);
axis tight;
%         hist(n,50);
axis tight;
ylabel('Probability Density');
xlabel('Respirable Fraction');
str = sprintf('\\fontsize{10} 1:10 Respirable Fraction plot with
Mean=%0.2e , Stdev=%0.2e',...
    mean(RFDist),std(RFDist));
title(str,'Units', 'normalized', ...
    'Position', [0.5 1.02], 'HorizontalAlignment', 'center')

caption = sprintf('Iteration: %d', samplesize);
% Print to static text control on a GUI.
set(handles.iterText, 'String', caption);
caption = sprintf('Elapsed Time: %d', toc);
```

```
% Print to static text control on a GUI.
set(handles.textTime, 'String', caption);
drawnow;

% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu2
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu2


% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editIn3_Callback(hObject, eventdata, handles)
% hObject    handle to editIn3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editIn3 as text
%        str2double(get(hObject,'String')) returns contents of editIn3
as a double


% --- Executes during object creation, after setting all properties.
function editIn3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editIn3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editIn4_Callback(hObject, eventdata, handles)
% hObject    handle to editIn4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editIn4 as text
%        str2double(get(hObject,'String')) returns contents of editIn4
as a double


% --- Executes during object creation, after setting all properties.
function editIn4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editIn4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in plotMEDIANbutton.
function plotMEDIANbutton_Callback(hObject, eventdata, handles)
% hObject    handle to plotMEDIANbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sample = get(handles.editSamples,'String');%grab number from number of
sample box
samplesize = str2double(sample);
if isnan(samplesize)
   msgbox('Samples');
   return
end
contents = get(handles.popupmenu2,'String');
popupmenuvalue = contents{get(handles.popupmenu2,'Value')};
num1 = str2double(get(handles.editIn3,'string'));
```

```
num2 = str2double(get(handles.editIn4,'string'));
axes(handles.axes1)
switch popupmenuvalue
    case 'Normal'
        pd = makedist('Normal','mu',num1,'sigma',num2);
        t = truncate(pd,0,inf);
        %t = pd;
        n = random(t,samplesize,1);
        nbins = max(min(length(n)./10,100),50);
        xi = linspace(min(n),max(n),nbins);
        dx = mean(diff(xi));
        fi = histc(n,xi-dx);
        fi = fi./sum(fi)./dx;
        assignin('base','MediaN', n);
        %assignin('base','rffi2', fi);
        bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
        axis tight;
        %         hist(n,50);
        ylabel('Probability Density');
        xlabel('Scale Param');
        str = sprintf('\\fontsize{10} Median distribution plot with
Normal distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
            mean(n),std(n));
        title(str,'Units', 'normalized', ...
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')
    case 'Lognormal'
        pd = makedist('Lognormal','mu',log(num1),'sigma',num2);
        t = truncate(pd,0,inf);
        %t = pd;
        n = random(t,samplesize,1);
        nbins = max(min(length(n)./10,100),50);
        xi = linspace(min(n),max(n),nbins);
        dx = mean(diff(xi));
        fi = histc(n,xi-dx);
        fi = fi./sum(fi)./dx;
        assignin('base','MediaN', n);
        %assignin('base','drfi2', fi);
        bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
        axis tight;
        %         hist(n,50);
        axis tight;
        ylabel('Probability Density');
        xlabel('Scale Param');
        str = sprintf('\\fontsize{10} Median distribution plot with Log
Normal distribution with Mean=%0.2e , Stdev=%0.2e',...
            mean(n),std(n));
        title(str,'Units', 'normalized', ...
```

```
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')

    case 'Uniform'
        if num1 < num2;
            % In unifrom distribution upper limt must be greater than
lower
            % limit, if not show the error message
            errordlg('Upper Limit is less than lower limt','Uniform
Distribution','modal')
            return;
        else
            pd = makedist('Uniform','Upper',num1,'Lower',num2);
            t = truncate(pd,0,inf);
            %t = pd;
            n = random(t,samplesize,1);
            nbins = max(min(length(n)./10,100),50);
            xi = linspace(min(n),max(n),nbins);
            dx = mean(diff(xi));
            fi = histc(n,xi-dx);
            fi = fi./sum(fi)./dx;
            assignin('base','MediaN', n);
            %assignin('base','rffi2', fi);
            bar(xi,fi,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
            axis tight;
            % %             hist(n,50);

            ylabel('Probability Density');
            xlabel('Scale Param');
           str = sprintf('\\fontsize{10} Median distribution plot with
Uniform distribution with\\mu=%0.2e ,\\sigma =%0.2e',...
            mean(n),std(n));
        title(str,'Units', 'normalized', ...
            'Position', [0.5 1.02], 'HorizontalAlignment', 'center')
        end

    case 'Single Value'
        assignin('base','MediaN', num1);
        cla(handles.axes1);
        return

end



function editIntervals_Callback(hObject, eventdata, handles)
% hObject    handle to editIntervals (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of editIntervals as
text
%        str2double(get(hObject,'String')) returns contents of
editIntervals as a double


% --- Executes during object creation, after setting all properties.
function editIntervals_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editIntervals (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in plotzerobutton.
function plotzerobutton_Callback(hObject, eventdata, handles)
% hObject    handle to plotzerobutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sample = get(handles.editSamples,'String');%grab number from number of
sample box
samplesize = str2double(sample);
intervl = get(handles.editIntervals,'String');
intervals = str2double(intervl);
if isnan(samplesize)
   msgbox('Samples');
   return
end
plotMEDIANbutton_Callback(handles.plotMEDIANbutton, 1 ,handles);
plotSPbutton_Callback(handles.plotSPbutton, 1 ,handles);
axes(handles.axes1);
MediaN = evalin('base','MediaN');
sp = evalin('base','SP');
RFDist = zeros(1,samplesize);
tic;
parfor i=1:length(RFDist)
    if size(MediaN) == 1
        num1 = MediaN;
    else
        num1 = MediaN(i);
    end
```

```
if size(sp) == 1
    num2 = sp;
else
    num2 = sp(i);
end
pd = makedist('Lognormal','mu',log(num1),'sigma',num2);
t = truncate(pd,0,inf);
%t = pd;
n = random(t,samplesize,1);
n = n./2;
n = n.^3;
xi = linspace(min(n),max(n),intervals);
dx = mean(diff(xi));
fi = histc(n,xi-dx);
fi = fi./sum(fi)./dx;
low = 1;
high = intervals;
lowisntset = 1;
highisntset = 1;
if (max(n) >= ((1e-5)/2)^3)
    for j=1:intervals
        if (xi(j) >= ((1e-5)/2)^3) && (highisntset == 1)
            high = j;
            highisntset = 0;
        end
    end
    fitrunct = fi(1:high);
    RFDist(i) = trapz(fitrunct)/trapz(fi);
elseif min(n) > ((1e-5)/2)^3
    RFDist(i) = 0;
    disp([num1,num2]);
    disp('Med. Scale');
elseif min(n) > ((1e-6)/2)^3
    for j=1:intervals
        if (xi(j) >= ((1e-5)/2)^3) && (highisntset == 1)
            high = j;
            highisntset = 0;
        end
    end
    low = 1;
    fitrunct = fi(low:high);
    RFDist(i) = trapz(fitrunct)/trapz(fi);
elseif max(n) < ((1e-5)/2)^3
    for j=1:intervals
        if (xi(j) >= ((1e-6)/2)^3) && (lowisntset == 1)
            low = j;
            lowisntset = 0;
        end
    end
```

```
        high = intervals;
        fitrunct = fi(low:high);
        RFDist(i) = trapz(fitrunct)/trapz(fi);
    end
    %assignin('base','Median', n);
    %assignin('base','drfi2', fi);

end
binselect = [100, intervals/10];
%nbins = min(max(binselect),10000);
nbins = 200;
xi2 = linspace(min(RFDist),max(RFDist),nbins);
dx2 = xi2(2)-xi2(1);
fi2 = histc(RFDist,xi2-dx2);
fi2 = fi2./sum(fi2)./dx2;
bar(xi2,fi2,'FaceColor',[.2 .6 .6],'EdgeColor',[.2 .6 .6],
'BarWidth',1);
assignin('base','RFDist', RFDist);
axis tight;
%         hist(n,50);
axis tight;
ylabel('Probability Density');
xlabel('Respirable Fraction');
str = sprintf('\\fontsize{10} 0:10 Respirable Fraction plot with
Mean=%0.2e , Stdev=%0.2e',...
    mean(RFDist),std(RFDist));
title(str,'Units', 'normalized', ...
    'Position', [0.5 1.02], 'HorizontalAlignment', 'center')

caption = sprintf('Iteration: %d', samplesize);
% Print to static text control on a GUI.
set(handles.iterText, 'String', caption);
caption = sprintf('Elapsed Time: %d', toc);
% Print to static text control on a GUI.
set(handles.textTime, 'String', caption);
drawnow;



% --- Executes on button press in radioRun.
function radioRun_Callback(hObject, eventdata, handles)
% hObject    handle to radioRun (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radioRun


% --- Executes on button press in radioPause.
```

```
function radioPause_Callback(hObject, eventdata, handles)
% hObject    handle to radioPause (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radioPause
```

Appendix D: ANS Powerpoint Presentation

Background

- US DOE non-reactor nuclear facilities use unmitigated and mitigated hazard evaluations
- If radiological doses from design basis events challenge or exceed dose evaluation guidelines, controls must be identified to mitigate the hazard
- Unnecessarily selecting controls can be cost or mission prohibitive
- Failure to select controls can subject the public and workers to inappropriate risk

Background

- Sometimes a calculated dose does not paint a clear picture
- Does an 19 rem dose to the public merit a Safety SSC selection? Always, Never, Sometimes
- A proposal was made to develop a software tool that can aid decision makers

Project Objectives

- Assist decision makers
- Visualize the calculated dose consequence distribution
- Easy to use portable computer application

Not Project Objectives

- Replace or compete with MACCS or RSAC
- Tell decision makers the "answer"
- Advocating alternate methods of performing required dose calculations!

Timeline

- The first year of the project ran from Fall of 2014 to Fall 2015, covered in previous ANS Summary
- The second year tasks were completed in September 2016
- 2016 Tasks:
  - Expanded Material at Risk Database
  - Multi Material CED Calculations
  - Additional Distribution Options
  - User Defined Distribution
  - Final Report

Pocatello | Idaho Falls | Meridian | Twin Falls
Idaho State UNIVERSITY

Dose Consequence Calculation

Traditional five-factor source term formula

$$ST = MAR \bullet DR \bullet ARF \bullet RF \bullet LPF$$

ST - source term (Bq)
MAR - total available material-at-risk (Bq)
DR - damage ratio (no units)
ARF - airborne release fraction (no units)
RF – respirable fraction (no units)
LPF – leak path factor (no units)

Pocatello | Idaho Falls | Meridian | Twin Falls

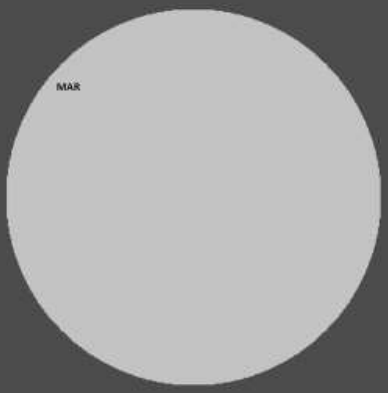
Idaho State UNIVERSITY

Dose Consequence Calculation

Traditional five-factor source term formula

$$ST = MAR \bullet DR \bullet ARF \bullet RF \bullet LPF$$

ST - source term (Bq)
MAR - total available material-at-risk (Bq)
DR - damage ratio (no units)
ARF - airborne release fraction (no units)
RF – respirable fraction (no units)
LPF – leak path factor (no units)



Pocatello | Idaho Falls | Meridian | Twin Falls

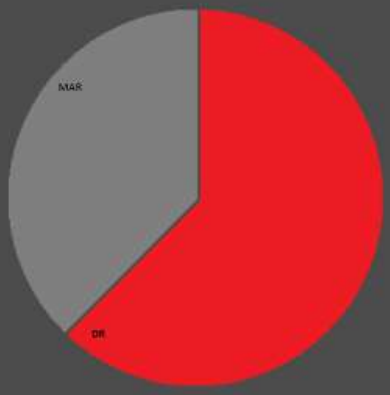
Idaho State UNIVERSITY

Dose Consequence Calculation

Traditional five-factor source term formula

$$ST = MAR \bullet DR \bullet ARF \bullet RF \bullet LPF$$

ST - source term (Bq)
MAR - total available material-at-risk (Bq)
DR - damage ratio (no units)
ARF - airborne release fraction (no units)
RF – respirable fraction (no units)
LPF – leak path factor (no units)



Pocatello | Idaho Falls | Meridian | Twin Falls

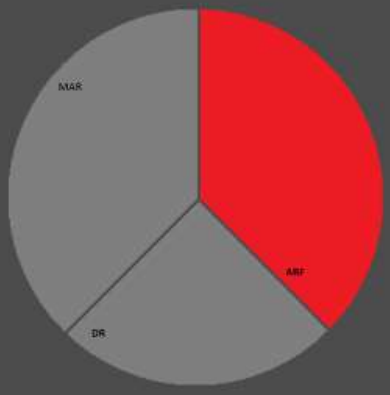
Idaho State UNIVERSITY

Dose Consequence Calculation

Traditional five-factor source term formula

$$ST = MAR \bullet DR \bullet ARF \bullet RF \bullet LPF$$

ST - source term (Bq)
MAR - total available material-at-risk (Bq)
DR - damage ratio (no units)
ARF - airborne release fraction (no units)
RF – respirable fraction (no units)
LPF – leak path factor (no units)



Pocatello | Idaho Falls | Meridian | Twin Falls

Idaho State UNIVERSITY

Dose Consequence Calculation

Traditional five-factor source term formula

$$ST = MAR \bullet DR \bullet ARF \bullet RF \bullet LPF$$

ST - source term (Bq)
MAR - total available material-at-risk (Bq)
DR - damage ratio (no units)
ARF - airborne release fraction (no units)
RF – respirable fraction (no units)
LPF – leak path factor (no units)

Pocatello | Idaho Falls | Meridian | Twin Falls

Idaho State UNIVERSITY

Dose Consequence Calculation

Traditional five-factor source term formula

$$ST = MAR \bullet DR \bullet ARF \bullet RF \bullet LPF$$

ST - source term (Bq)
MAR - total available material-at-risk (Bq)
DR - damage ratio (no units)
ARF - airborne release fraction (no units)
RF – respirable fraction (no units)
LPF – leak path factor (no units)

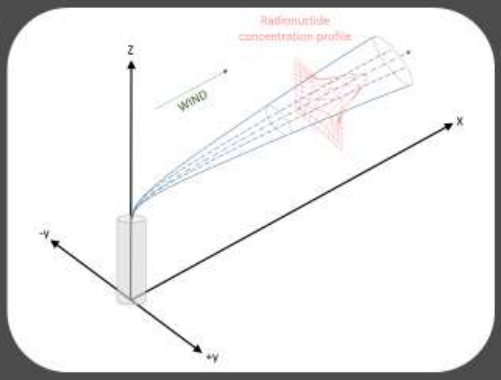
Pocatello | Idaho Falls | Meridian | Twin Falls

Idaho State UNIVERSITY

Dose Consequence Calculation

Once the source term is known, the dose can be calculated

$$CED = \chi/Q \bullet ST \bullet BR \bullet DCF$$



CED – committed effective dose (Sv)
χ/Q – plume dispersion (s/m^3)
ST - source term (Bq)
BR – breathing rate (m^3/s)
DCF – dose conversion factor (Sv/Bq)

Computer Application Concept

- Traditionally each term in the ST and CED calculation is represented as a point estimate (typically a 95% CI value)
- Allow users to represent any (or all) of the terms with an underlying distribution
- Calculate the resulting dose distribution using Monte Carlo sampling of the input distributions
- Provide a visual display of the dose distribution

Computer Application Concept

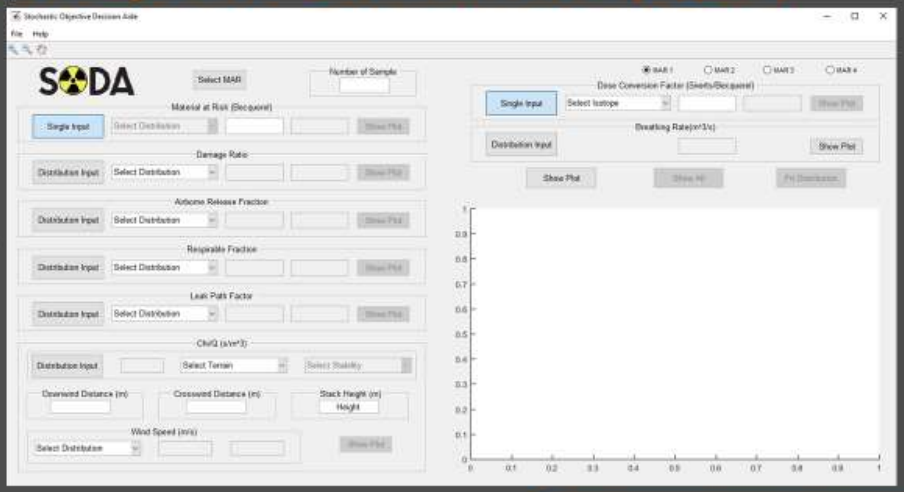
- Simple to use (GUI driven)
- Portable (no supporting software required)
- Automatic Monte Carlo sampling of distributions
- Develop the code using MATLAB
- Compiled version must not require MATLAB
- Compiled version must run on Windows or Mac

Pocatello | Idaho Falls | Meridian | Twin Falls

Idaho State UNIVERSITY

Stochastic Objective Decision Aide (SODA)

- Single screen GUI
- Each input can be entered as a fixed point or as a distribution
- Built-in distributions
- Distribution plots can be generated for each input parameter
- Dose distribution is automatically plotted

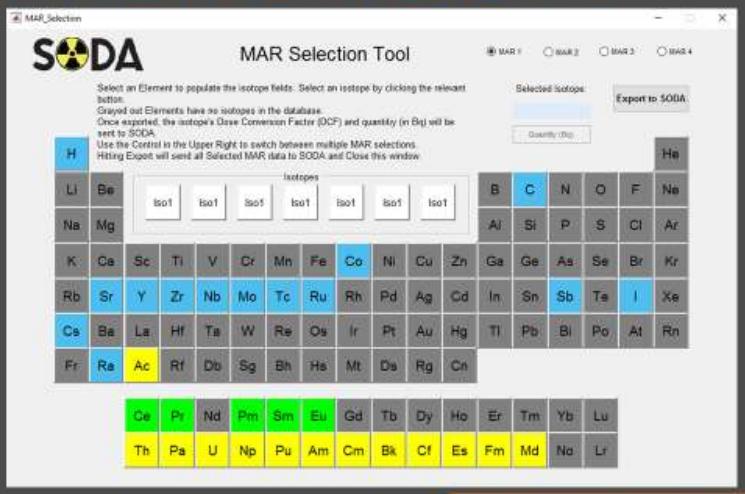


Pocatello | Idaho Falls | Meridian | Twin Falls

Idaho State UNIVERSITY

Expanded Material at Risk Database

- Provide easy to access MAR Database
- User enters quantity, DCF automatically entered
- Common Isotope data provided
- Color coded for ease of use
- Easy to edit .csv database file, can use Excel

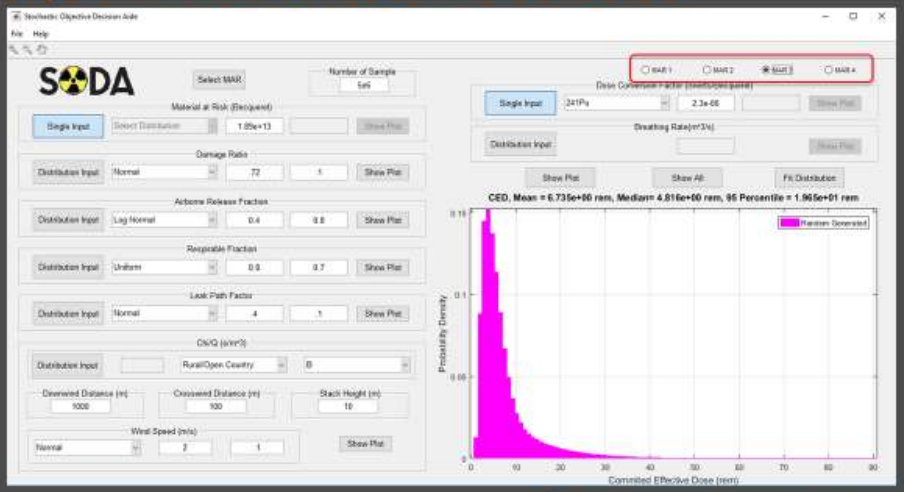


Pocatello | Idaho Falls | Meridian | Twin Falls

Idaho State University

Multiple Material CED Calculation

- Switch between selected MARs with radio buttons
- Specification of MAR quantity, DCF, ARF, RF is material specific
- Each material specification can be saved and recalled
- Each material CED calculated, then summed for total CED

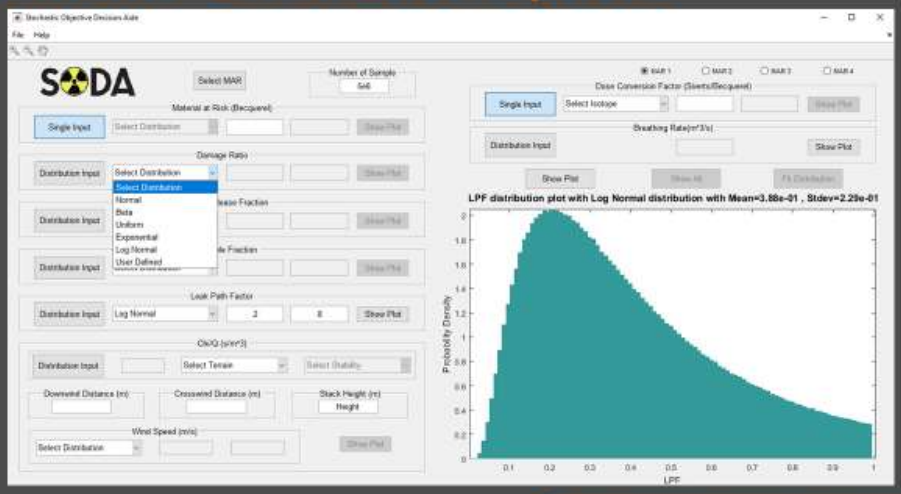


Pocatello | Idaho Falls | Meridian | Twin Falls

Idaho State University

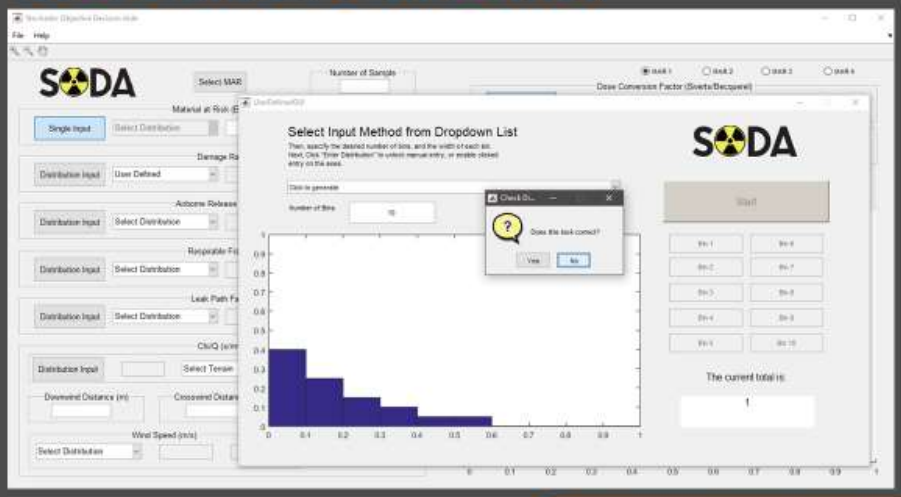
Additional Distribution Options

- Log Normal option added to distribution selection
- Distribution is localized by specification of its most probable value
- Selections include: Normal, Beta, Uniform, Exponential, and Log Normal



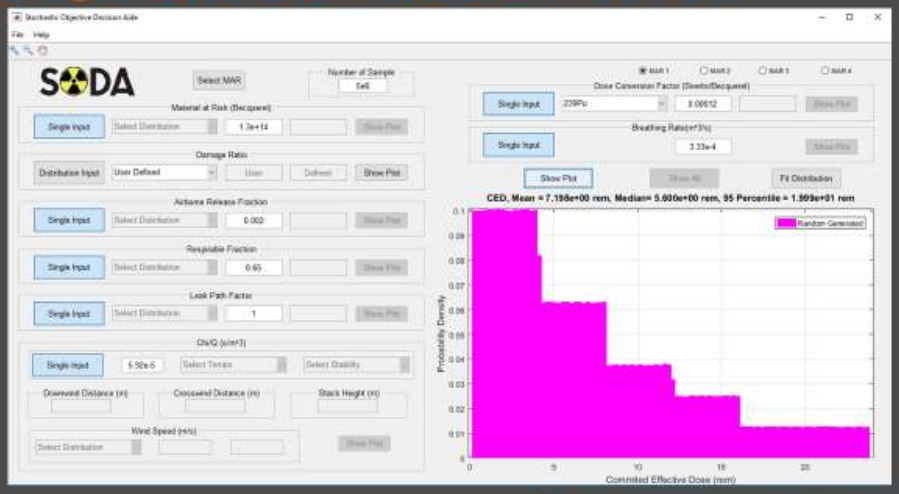
User Defined Distribution

- Easy to use tool for custom distribution specification
- Click on the axes, or manually enter values
- Check for normalized total probability



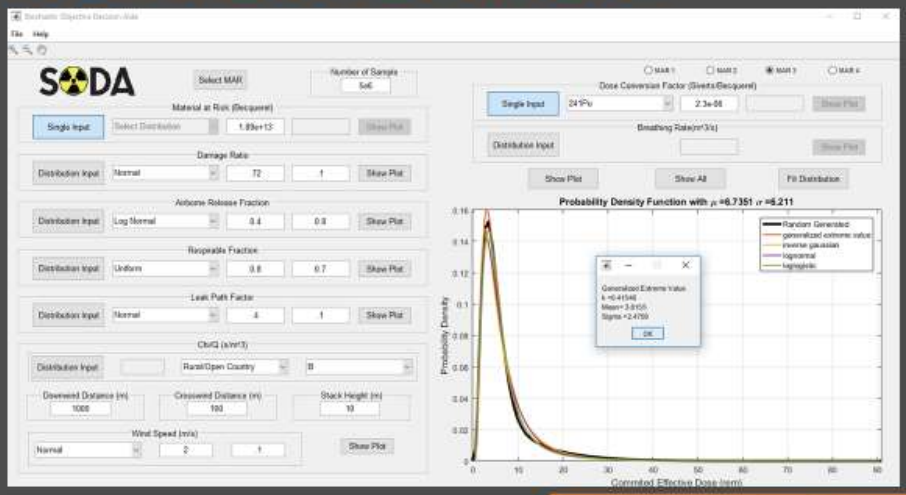
Sampling a User Defined Distribution

- SODA can sample single or multiple UDDs in a CED calculation
- Each bin is sampled uniformly between the bin limits



Dose Distribution Identification

- Newly introduced Bayesian Information Criterion methodology
- Testing of the resulting distribution to identify a likely fit
- Goal is to identify the distribution so that the resulting uncertainty can be better quantified

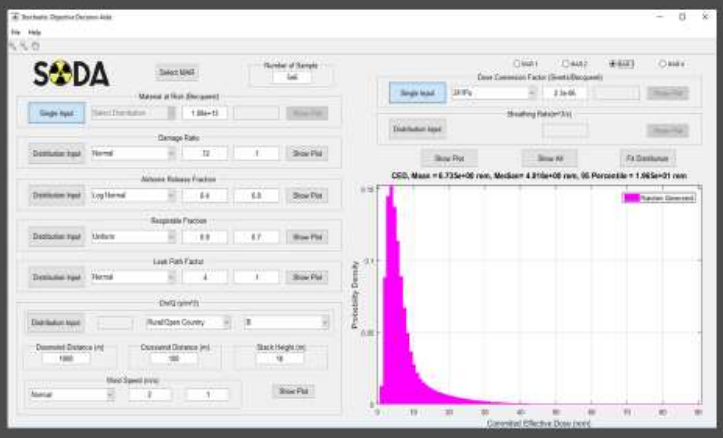


Future Improvements

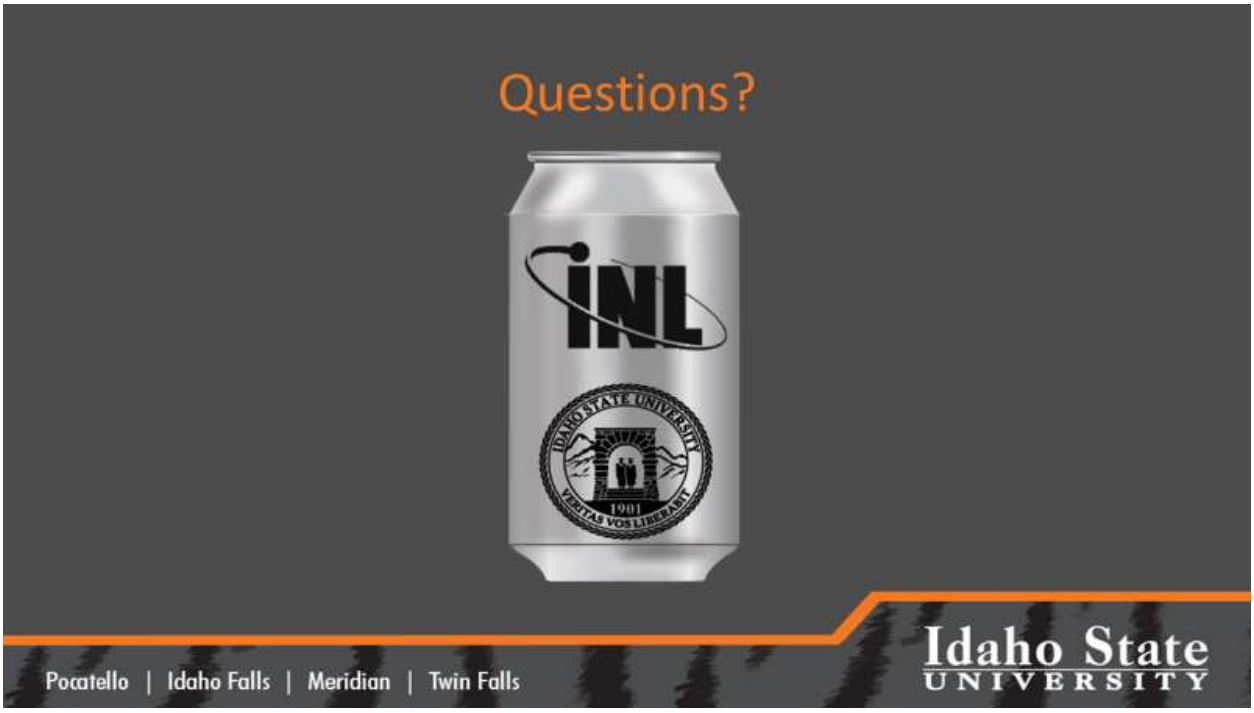
- Code Optimizations
- Computer specific sample count limits
- Evaluation guideline specification
- DCF/ARF/RF specific distributions
- Scripting

Conclusion

- SODA was developed to provide a tool to help decision makers
- The code allows users to examine the impact of distributions on the calculated dose
- The code is portable and does not require a MATLAB license



Pocatello | Idaho Falls | Meridian | Twin Falls
Idaho State UNIVERSITY

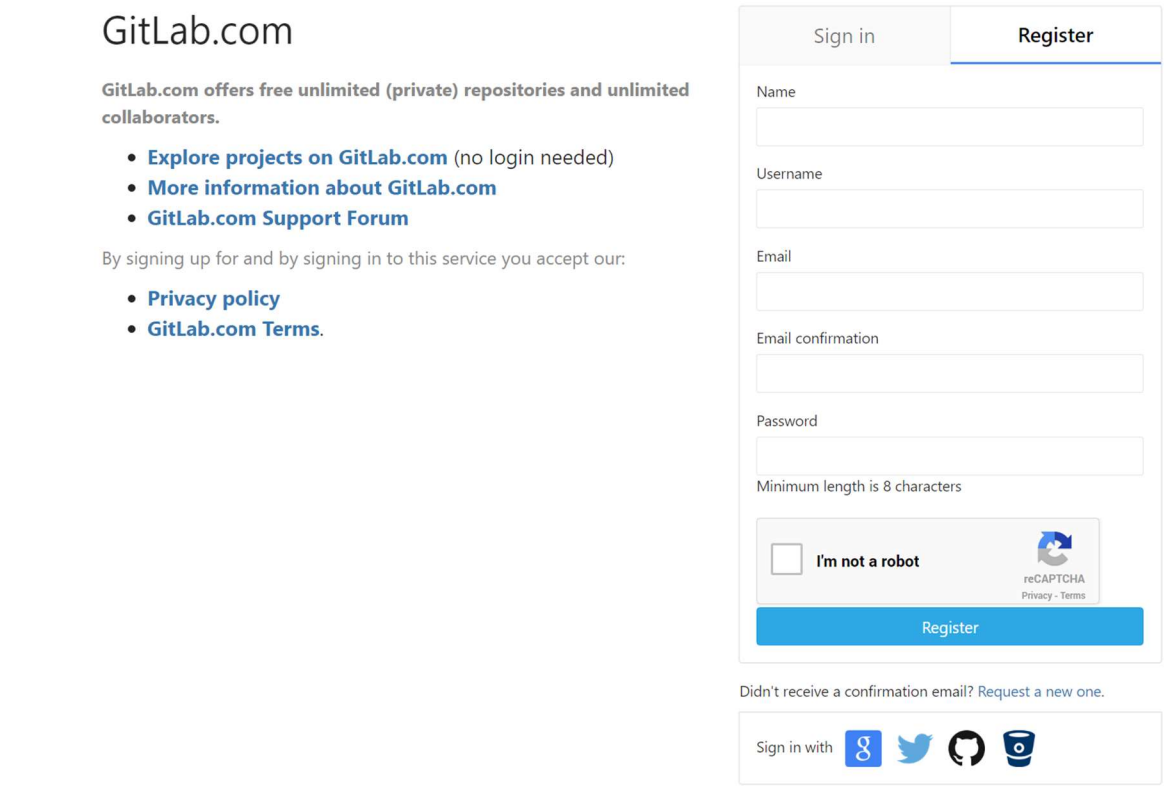

Questions?
INL
IDAHO STATE UNIVERSITY
1901
VERITAS VOS LIBERABIT
Pocatello | Idaho Falls | Meridian | Twin Falls
Idaho State
UNIVERSITY

Appendix E: Git/GitLab Setup and Instructions

For future research projects under Dr. Pope which involve software development, it is expected to become common practice to use Git, and host on GitLab. To facilitate this, a group on GitLab has been formed, with the title "Pope Research." In the following section, instructions will be provided on how to get into this group, create a project, and perform basic tasks.

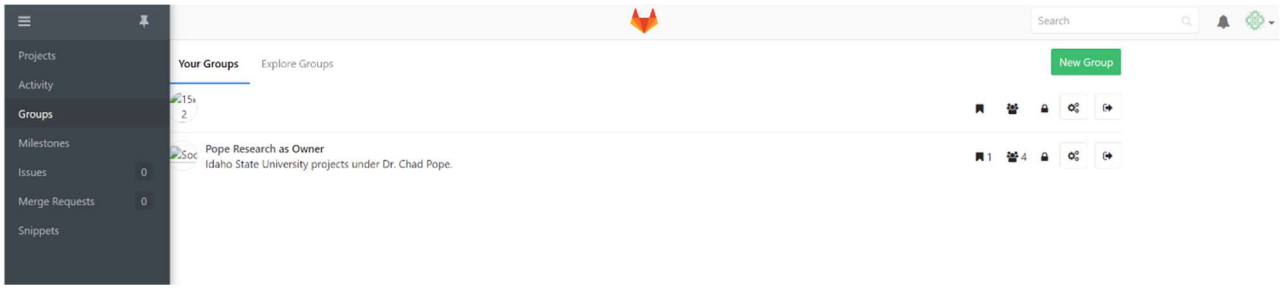
The first step a perspective individual should perform is getting an account setup on GitLab. Since acceptance to the Pope Research group can only be accomplished by invitation, a new user will need to wait on a colleague to gain acceptance. Thus, this step should be completed first so that this process can begin.

Begin by navigating your internet browser to www.gitlab.com. Before you will be a screen with tabs that read Sign In and Register. Select Register, and what appears then can be seen below.

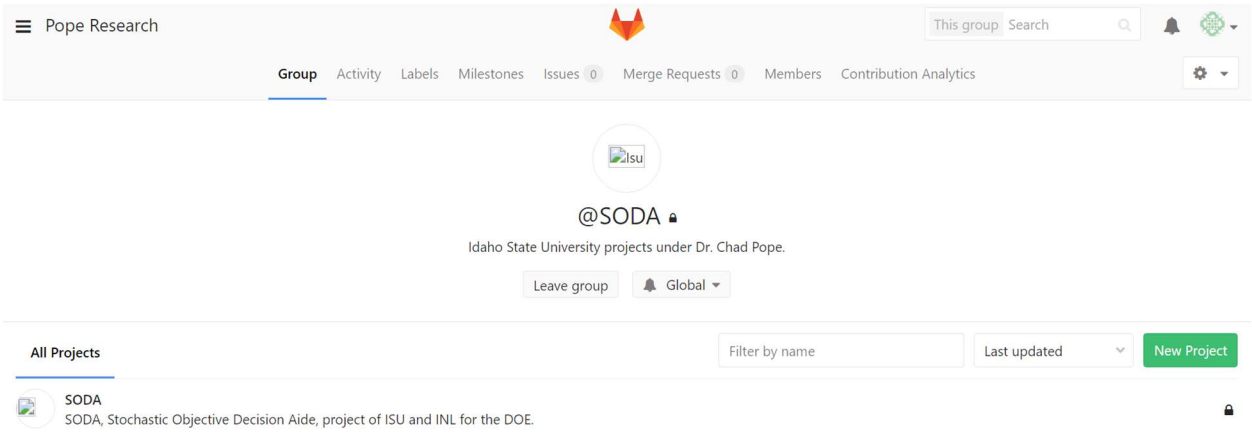


Fill out the information that is requested, taking note of your username. It is necessary to provide this user name to a colleague who is already in the Pope Research group to get an invitation. Email your contact with this information as soon as the registration process is finished. It is also necessary to confirm your email address with GitLab after completing registration.

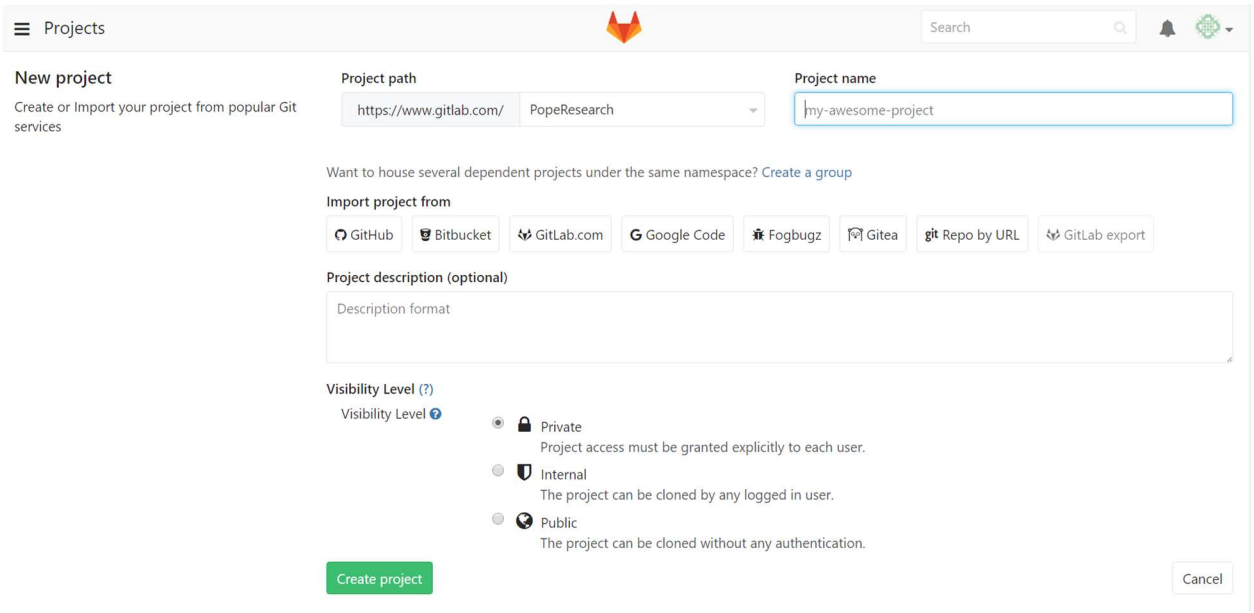
Once the account has been created, and your email has been confirmed, a new user will be given access to Pope Research in short order. Once the user has access to Pope Research, they can navigate to the group on the website as shown below.



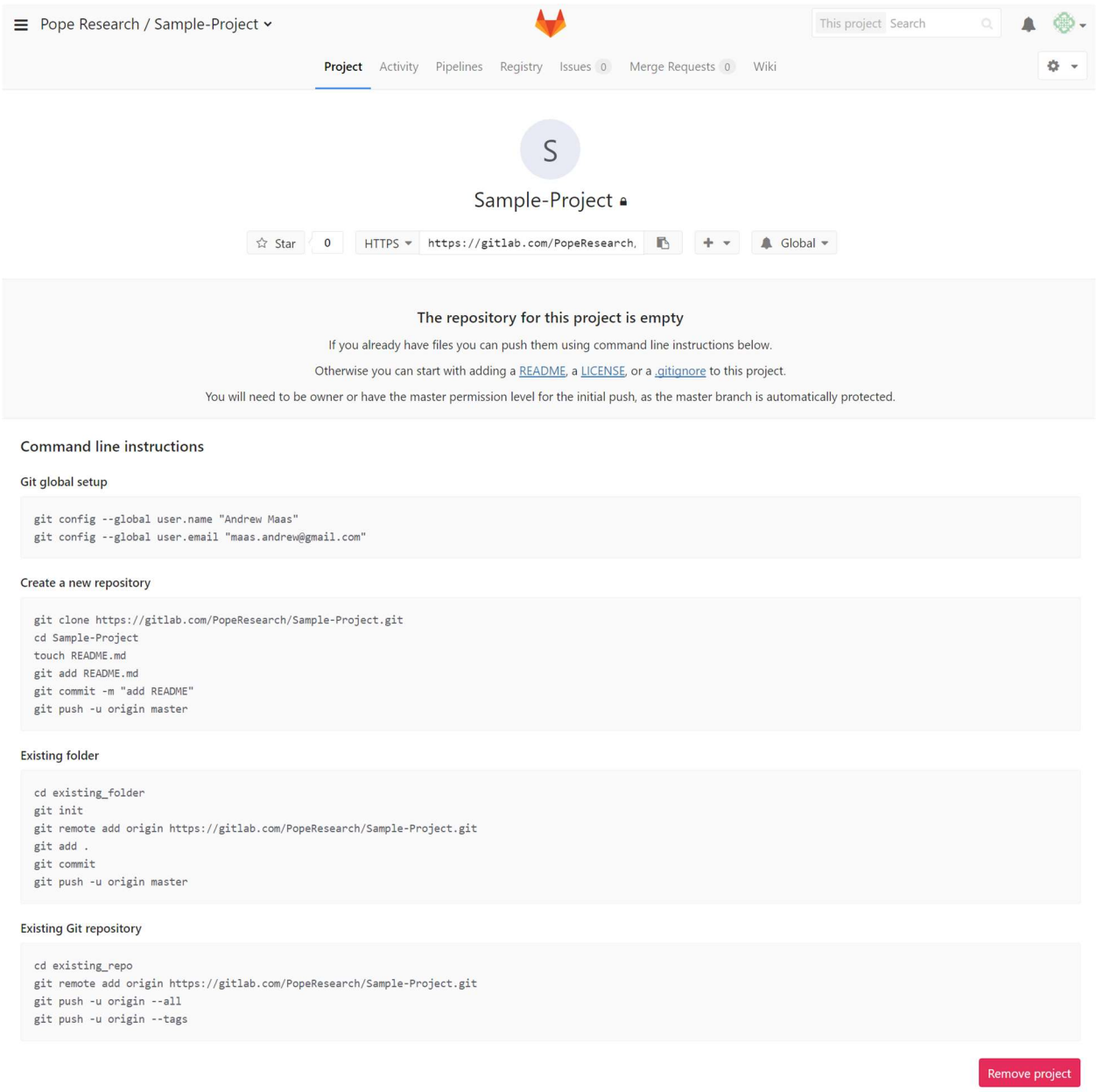
From here, select the Pope Research Group. Note that in most cases, a new user will not have Owner status on the group. Once in the group, the user will be given the option to create a new project, or access existing ones they have been given access to. See below:



In this case, the only project that is currently in the Pope Research group is SODA. There will be more added in the future. If the user needs to add a new project to the group, select the green New Project button, as shown in the above image. When the page loads, enter the relevant information. An example is shown below:



Typically, as a user under Pope Research, it is recommended to have the visibility set to private (Shown above). This setting allows the hosted repository to only be available to users that are a part of that project, or those who have been given access to the project. Do not select any of the import options, unless the code base that is desired to form this new project is already hosted on one of the indicated sites. Once the new project has been created, navigating to the project will present the user with a screen as shown below:

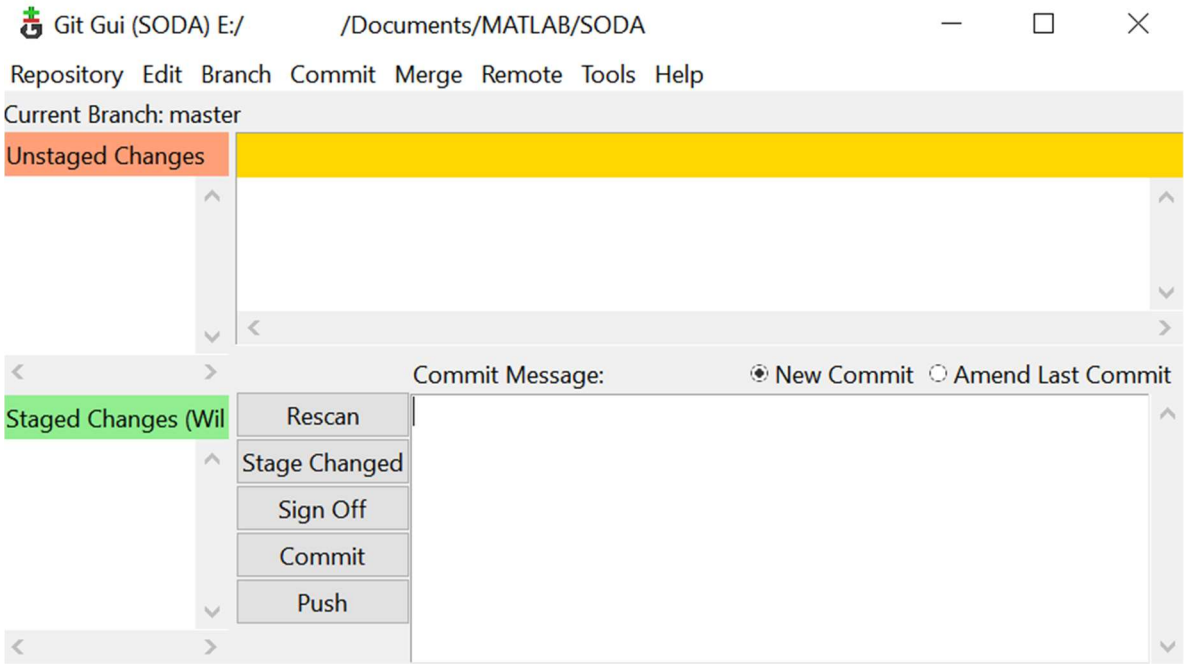


As can be seen in the above screenshot, instructions are provided to upload files to the repository, and interface with Git. At this point, the Git software is needed to continue. To acquire the Git source code control software, navigate your web browser to https://git-scm.com/downloads and select the download option that matches your operating system. Another useful link, when getting started is https://git-scm.com/doc which contains the Git

manual. This manual will contain all of the information needed to run the Git software, although the instructions provided on GitLab are sufficient to get started.

Once Git is installed on a computer, the available options for how to use Git will depend on the operating system which is running it. In every case, a terminal-based Git (Git-Bash) will be available. The commands shown in the GitLab quick start guide are Git-Bash commands, since Git-Bash will be universally effective on any OS.

For Windows™ users who are not comfortable with the use of a terminal, there is also the option to use Git GUI (Graphical User Interface). This is a graphical user interface program which allows the user to perform most Git commands. For general purpose use, this is often sufficient. The Git GUI can be seen below:



For additional tips and instructions, consult the GitLab quick start guide that was shown above, or the Git manual, for which a link was provided. [3]