INL/EXT-16-40755
Revision 0

NSR&D Program Fiscal Year 2015

Funded Research

Stochastic Modeling of Radioactive Material Releases Final Report

Idaho National Laboratory & Idaho State University

December 2016

INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

NSR&D Program Fiscal Year 2015 Funded Research

Stochastic Modeling of Radioactive Material Releases

Final Report

Revision 0
December 2016

Idaho National Laboratory & Idaho State University

Prepared for the
U.S. Department of Energy
Assistant Secretary for Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

EXECUTIVE SUMMARY

Nonreactor nuclear facilities operating under the approval authority of the U.S. Department of Energy use unmitigated hazard evaluations to determine if potential radiological doses associated with design basis events challenge or exceed dose evaluation guidelines. Unmitigated design basis events that sufficiently challenge dose evaluation guidelines or exceed the guidelines for members of the public or workers, merit selection of safety structures, systems, or components or other controls to prevent or mitigate the hazard.

Idaho State University, in collaboration with Idaho National Laboratory, has developed a portable and simple to use software application called SODA (Stochastic Objective Decision-Aide) that stochastically calculates the radiation dose distribution associated with hypothetical radiological material release scenarios. Rather than producing a point estimate of the dose, SODA produces a dose distribution result to allow a deeper understanding of the dose potential. SODA allows users to select the distribution type and parameter values for all of the input variables used to perform the dose calculation. Users can also specify custom distributions through a user defined distribution option. SODA then randomly samples each distribution input variable and calculates the overall resulting dose distribution. In cases where an input variable distribution is unknown, a traditional single point value can be used. SODA, developed using the MATLAB coding framework, has a graphical user interface and can be installed on both Windows and Mac computers. SODA is a standalone software application and does not require MATLAB to function.

SODA provides improved risk understanding leading to better informed decision making associated with establishing nuclear facility material-at-risk limits and safety structure, system, or component selection. It is important to note that SODA does not replace or compete with codes such as MACCS or RSAC; rather it is viewed as an easy to use supplemental tool to help improve risk understanding and support better informed decisions. The SODA development project was funded through a grant from the DOE Nuclear Safety Research and Development Program.

Contents

Figures

Tables

ACROYNMS

ARF	Airborne Release Fraction
BIC	Bayesian Information Criterion
BR	Breathing Rate
CED	Committed Effective Dose
DBE	Design Basis Event
DCF	Dose Conversion Factor
DOE	Department of Energy
DR	Damage Ratio
GUI	Graphical User Interface
GUIDE	Graphical User Interface Development Environment
ICRP	International Commission on Radiological Protection
IDE	Integrated Development Environment
INL	Idaho National Laboratory
ISU	Idaho State University
LPF	Leak Path Factor
MACCS	MELCOR Accident Consequence Code System
MAR	Material-at-Risk
NSRD	Nuclear Safety Research and Development
PDC	Probability Density Curve
PDF	Probability Density Function
RF	Respirable Fraction
RSAC	Radiological Safety Analysis Computer
SODA	Stochastic Objective Decision Aide
SSC	System, Structure, or Component
ST	Source Term

1. INTRODUCTION

Nonreactor nuclear facilities operating under the approval authority of the U.S. Department of Energy (DOE) use unmitigated hazard evaluations to determine if potential radiological doses associated with design basis events (DBEs) challenge dose evaluation guidelines. Unmitigated DBEs that sufficiently challenge dose evaluation guidelines for members of the public or workers merit selection of safety structures, systems, or components (SSCs) or other controls to prevent or mitigate the hazard.

Idaho State University (ISU), in collaboration with Idaho National Laboratory (INL), has developed a portable and simple to use software application called SODA (Stochastic Objective Decision-Aide) that utilizes a Monte Carlo based code system to stochastically calculate the radiation dose distribution of hypothetical radiological material release scenarios. Rather than producing a point estimate of the dose, SODA produces a dose distribution to allow a deeper understanding of the dose potential. Thus, SODA provides improved risk understanding leading to better informed decision making associated with establishing material-at-risk (MAR) limits and safety SSC selection. It is important to note that SODA does not replace or compete with codes such as MACCS or RSAC, rather it is viewed as an easy to use supplemental tool to help improve risk understanding and support better informed decisions.

2. BACKGROUND

Radioactive material release modeling codes typically provide a bounding single point estimate of receptor dose. While this approach attempts to bound the dose estimate, at least in the context of the atmospheric dispersion model, it falls short in providing quantification of the expected value and the uncertainty associated with the dose estimate. This is particularly problematic when one considers the lack of governing distribution identification for input parameters. Thus, potential doses to workers and members of the public can be overstated, leading to potentially excessive MAR limits and over selection of safety SSCs.

DBE dose consequence calculations traditionally use the “five-factor” formula:

$$ST = MAR \cdot DR \cdot ARF \cdot RF \cdot LPF \quad (1)$$

where ST is the source term (Bq), MAR is the total available material-at-risk (Bq), DR is the damage ratio (no units), ARF is the airborne release fraction (no units), RF is the respirable fraction (no units), and LPF is the leak path factor (no units).

Potential radiation doses are then calculated using:

$$CED = \chi / Q \cdot BR \cdot ST \cdot DCF \quad (2)$$

where CED is the committed effective dose (Sv), χ/Q is the plume dispersion (s/m^3), BR is the breathing rate (m^3/s), ST is the source term (Bq), and DCF is the dose conversion factor (Sv/Bq).

The ST is defined as the amount of radioactive or hazardous material released to the environment following an accident. Quantifying the radiation source term goes beyond the determination of the different radionuclides involved. Understanding how the radionuclide reacts with the environment is vital if the source term is to be accurate. The effect of barriers and containers plays a significant role in decreasing the uncertainties. Considering the uncertainties involved, obtaining an accurate value for the ST is challenging. Most ST values are usually bounding or worst case scenarios; this means that MAR, DR, ARF, RF and LPF are selected based on bounding estimates to produce a very bounding ST. This method of estimation, while conservative, lacks detailed information about the overall dose distribution.

MAR is the amount of radioactive material available to be acted upon as a result of a physical disturbance such as a spill, shock or fire. The MAR can also be defined as the value representing a maximum quantity of radioactive material present or anticipated to be effected from the analysis of a structure. Thus, the MAR associated with a facility explosion would be different from the MAR during the spill of a radioactive material powder. For instance, if an earthquake were to occur at a nuclear facility the MAR would be everything in the nuclear facility, because the earthquake impacts the entire facility.

DR is the fraction of the MAR that is effected by the accident scenario. For example, if a facility holds many containers of radioactive material and a seismic event occurs, a DR could be applied indicating that only a portion of the containers are subjected to enough seismic impact to result in the release of radioactive material.

ARF is employed in the estimation of the fraction of radioactive materials suspended in air as an aerosol, thus available for transport due to physical stress from a specific accident.

RF refers to the quantity of released material that has an aerosol particle size such that it can penetrate into the alveolar region of the lungs and be deposited. Large particles tend to deposit before they get deep into the lung, where they can be expelled through normal body processes. Very small particles

will get into the alveolar region of the lung, but they tend to remain in air suspension rather than being deposited into the lung. These particles are typically expelled in later breaths. Particles of intermediate size are able to get into the alveolar region, but are heavy enough to deposit deep in the lung. The fraction of the material that is suspended in air, having this particle size, is given by the RF.

Chi over Q (χ / Q) is a normalized air concentration term, expressed in seconds per cubic meter. This is used to quantify the effect of diffusion on a plume as it propagates downwind. As calculated, it is an expression of radioactivity per cubic meter, per radioactivity per second. Q is the material release rate term (radioactivity per second). Since Chi depends on Q, dividing out Q normalizes to the release rate, allowing the term to describe the rate of plume dispersion, independent of release rate. Figure 1 shows how radioactive material can disperse as it transports away from the event site. Independent parameters affecting the concentration profile can be stochastically sampled and used in the χ/Q portion of the dose calculation.

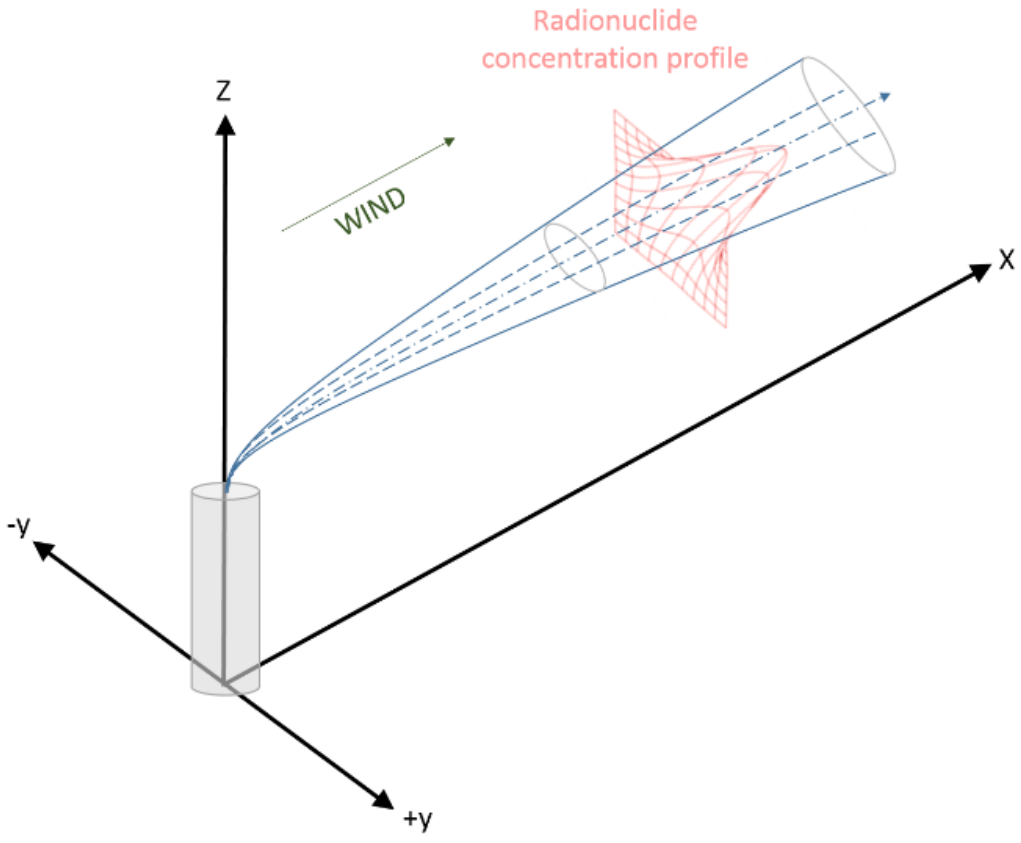


Figure 1. Plume dispersion.

BR allows the model to account for the difference in respiration rate of a person who is exposed depending on their activity state. Over a day, a given person spends some time sleeping, some time awake and inactive, and some time active. Using this 24-hour data for typical respiration rate, and

sampling this data, the distribution of the resulting dose is influenced. This allows decision makers to ascertain the full range of potential doses to a population.

DCF is the multiplicative factor relating activity to absorbed dose. A person exposed to material of some activity is going to receive a different absorbed dose dependent upon the type of radiation, where in or on the body that the exposure is occurring, and other factors. The traditional point values used are obtained from the ICRP.

Conservative single value input parameters are typically used to represent ARF, RF, LPF and BR. The traditional methodology, while conservative, can lead to skewed conclusions in the balance between cost and risk reduction resulting in over engineered systems with greater design, construction and operating costs. Rather than using a bounding single point value for each parameter in the dose consequence calculation, distributions for some or all of the parameters can be used.

Each parameter distribution can be stochastically sampled and the resulting dose consequence calculation can be repeated many times to develop a dose consequence distribution. The resulting dose distribution can then be used by decision makers to make a better informed decision about how a particular DBE challenges dose evaluation guidelines.

3. SODA

The SODA software application is not intended to replace or even compete with traditional radioactive material release modeling codes such as the MELCOR Accident Analysis Code System (MACCS) or Radiological Safety Analysis Computer (RSAC) code, rather it is viewed as a simple to use supplement to help improve risk understanding. The application was developed using MATLAB computing environment and programming language and it incorporates use of Monte Carlo techniques as well as a graphical user interface (GUI). The code system also utilizes MATLAB vectorization to provide execution time reduction. The application includes user selection of the governing distribution for parameters; MAR, DR, ARF, RF, LPF, BR, and DCF. While MATLAB was used for the development work, the application is distributed as a self-contained executable program. As seen in Figure 2, features of the application include pull down menus with available distributions for the various parameters.

Figure 2. SODA input options.

The user has the option to plot the input parameter distribution that results from the random sampling process. For example, Figure 3 shows a sample distribution of χ/Q values resulting from five million random samples. Plume dispersion is the concept that as particles are transferred through the air, their concentration decreases. As the concentration is reduced, the size of the area reached by the particles will increase, but the number of particles reaching a specific area will be lower than the original number of particles released due to the dispersion. The χ/Q value depends on the terrain, the atmospheric stability class, the downwind and crosswind distances, the stack height, and the wind speed. The dependency of the plume dispersion on each of these parameters varies greatly. Parametric study of these parameters is included in the User Manual (Appendix A) to help guide users in the selection of input parameters.

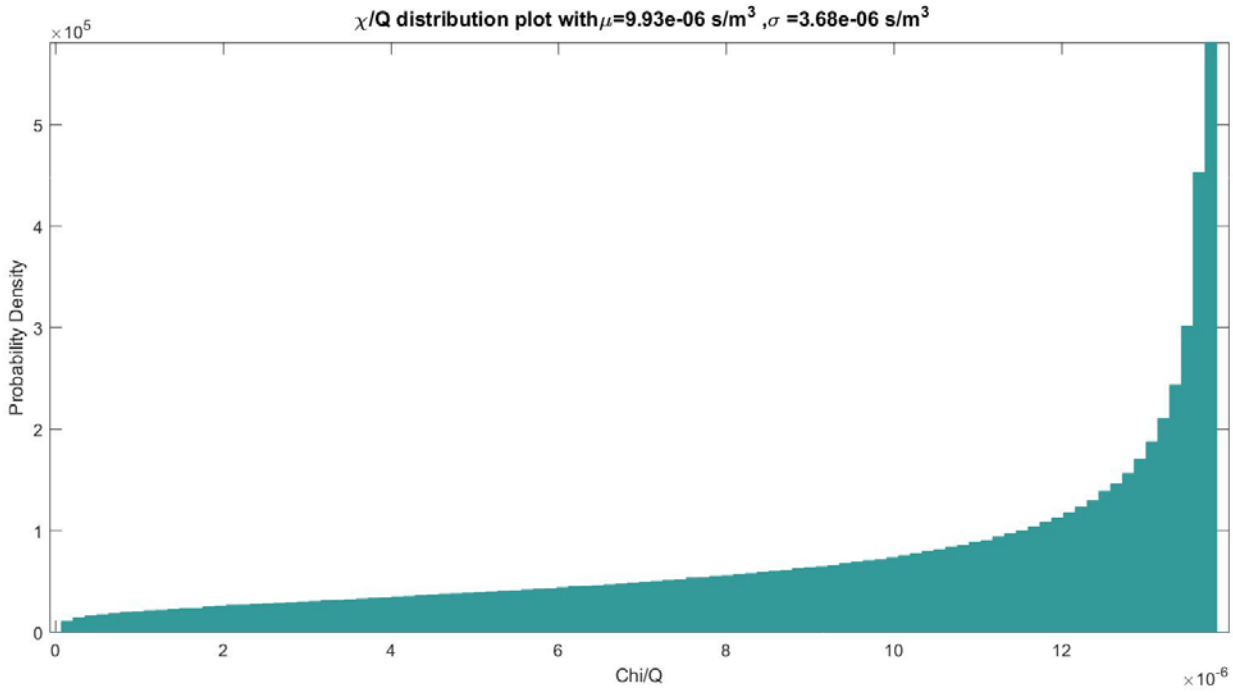


Figure 3. χ/Q values from random sampling.

Selecting the appropriate input parameter distribution is the most challenging aspect of performing a Monte Carlo based calculation of the potential dose. For example, Figure 4 shows the breathing rate distribution for an adult male over a hypothetical 24-hour period with breathing rates for sleep, sedentary activity, light physical activity, and heavy physical activity (ICRP-89). The traditional approach suggested by DOE-STD-3009-2014, is to select a value of 3.3 x 10^{-2} m^3/hr, corresponding to light activity for an adult male. However, it is clear that there are periods where the breathing rate is significantly lower and higher. SODA can randomly sample the distribution shown in Figure 4 when performing the dose calculation, thereby influencing the distribution of the resulting dose which helps inform decision makers by more clearly showing the range of potential dose result values.

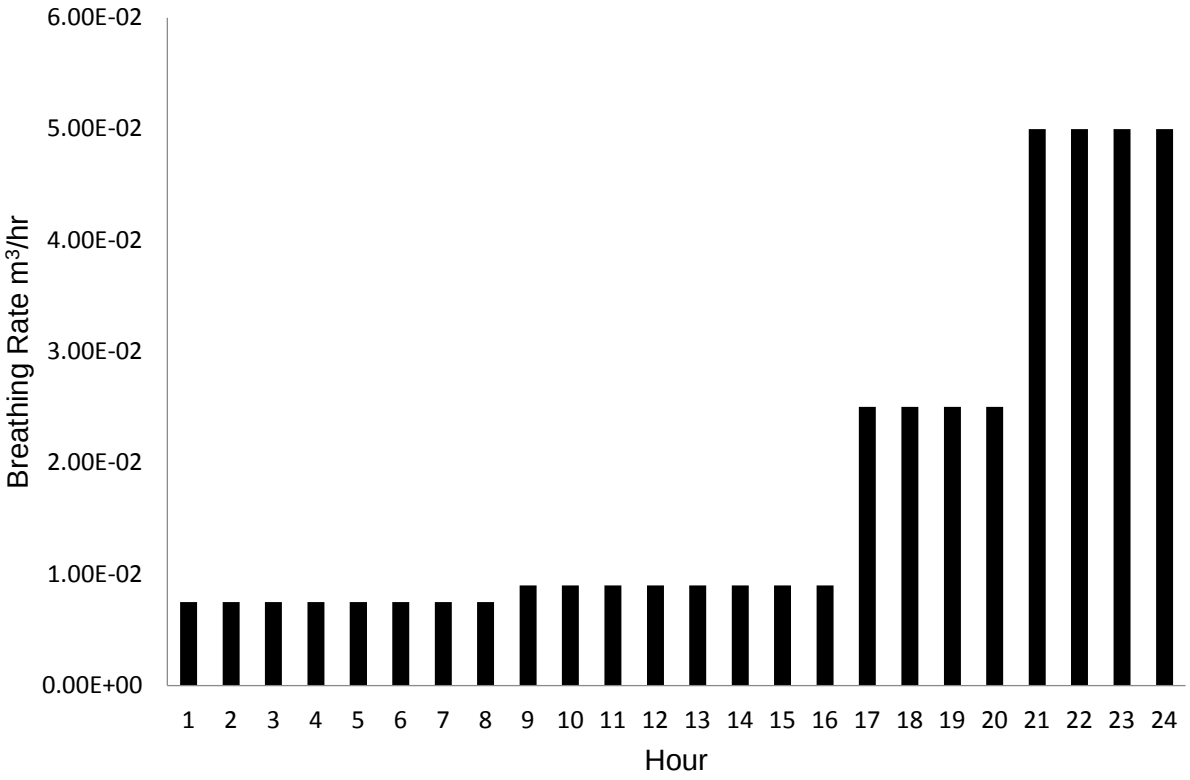


Figure 4. Adult male breathing rate.

The SODA GUI interface with an accompanying χ/Q distribution plot is shown in Figure 5. The user can select the pre-defined distribution for each input variable along with the associated distribution key parameters such as the mean value and variance. Individual input parameter sampled distribution plots can be requested. The user also selects the number of Monte Carlo calculations to execute. The resulting dose distribution is then displayed for the user. SODA allows the user to save the input selections as well as export the resulting dose distribution plot.

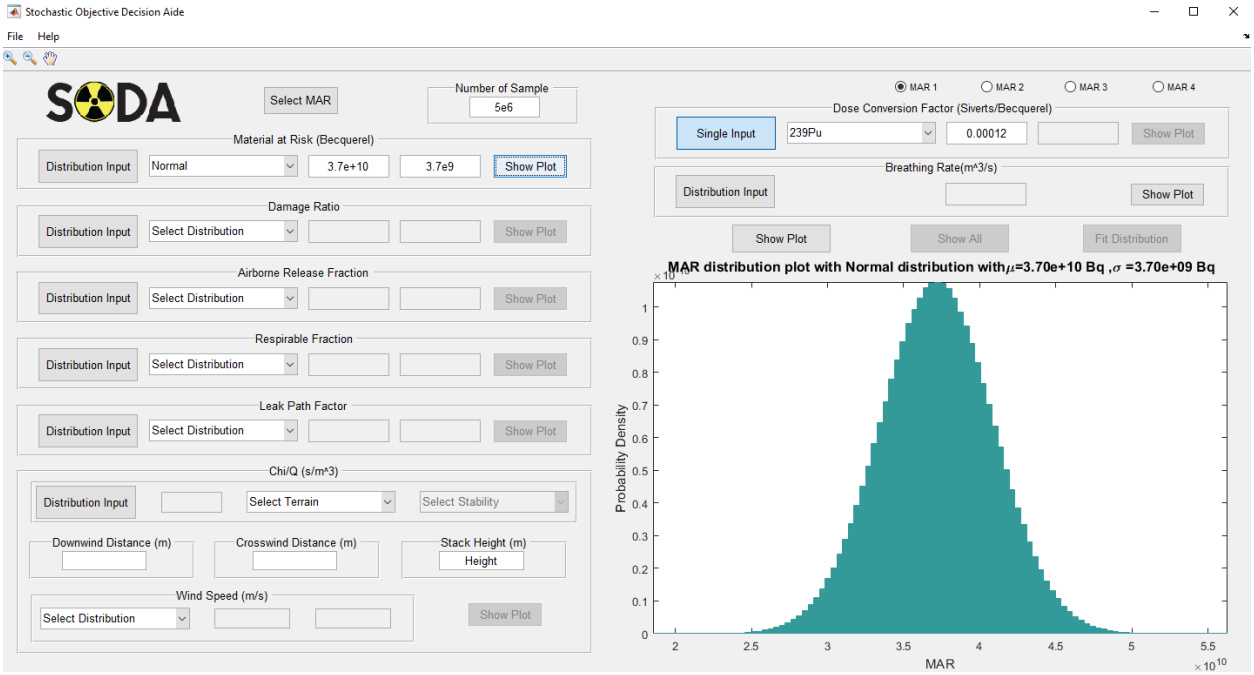


Figure 5. SODA GUI Screen.

3.1. Coding Framework

One objective for this project was to develop a GUI based on intuitive functions. GUI applications are easy to use and require little memorization of command or syntax structure. As such, the SODA application needs little training to use. Help File instructions are provided to further support rapid use. Furthermore, placing the mouse pointer over a button will display tips. The application was prepared in MATLAB, created by Math Works Inc. MATLAB contains a large standard library with an easy to use Integrated Development Environment (IDE). MATLAB is a high level programming language; thus it is easy to use. For example, a simple addition of variables in MATLAB is straightforward whereas in low level language the user has to declare the variable type before doing any mathematical operations. High level languages such as MATLAB are prevalent, because they allow focus on the problem rather than the command syntax and code. With the advent of high speed processors, and ample memory sizes, high level languages are convenient for developing prototype codes.

MATLAB was the choice for the application development because of the many built-in functions and libraries that come standard with the MATLAB package; the use of which greatly reduced the time needed to create the application. Importantly, the MATLAB statistical toolbox as well as the MATLAB compiler toolbox was used to help develop the application. The statistical toolbox is used to fit probability distributions to data and generate random numbers for Monte Carlo simulations of various probability distributions. The MATLAB compiler toolbox is used for compilation of the code to a

standalone application for Mac and Windows computers. The entire MATLAB code for SODA is provided in Appendix B.

3.2. GUI

MATLAB utilizes the Graphical User Interface Development Environment (GUIDE). GUIDE was used to design SODA. Using the GUIDE layout editor, the user can graphically design the User Interface. GUIDE then automatically generates the MATLAB code for constructing the user interface, and the event driven functions which are called when user input is made. This allows the programmer to focus on the code that solves the problem rather than visual cosmetics of the application. GUIDE is accessible by typing *guide* in the MATLAB command window which activated the quick start menu shown in Figure 6.

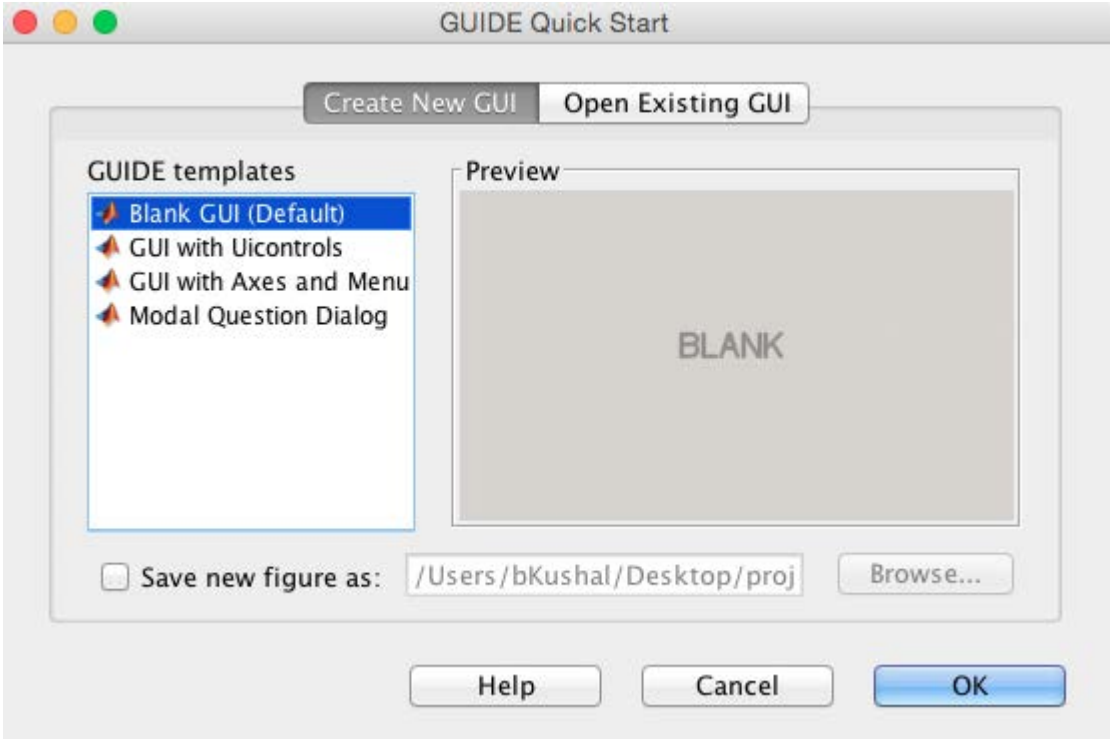


Figure 6. MATLAB GUI Quick Start.

The user has a choice to select from various GUI templates. To create SODA, a blank GUI (Default) was selected which reveals the window shown in Figure 7 where the user can drag and drop all the necessary tools required in the application. Here, the user is creating the visual and cosmetic aspects of the application by deciding which tools are required for the application.

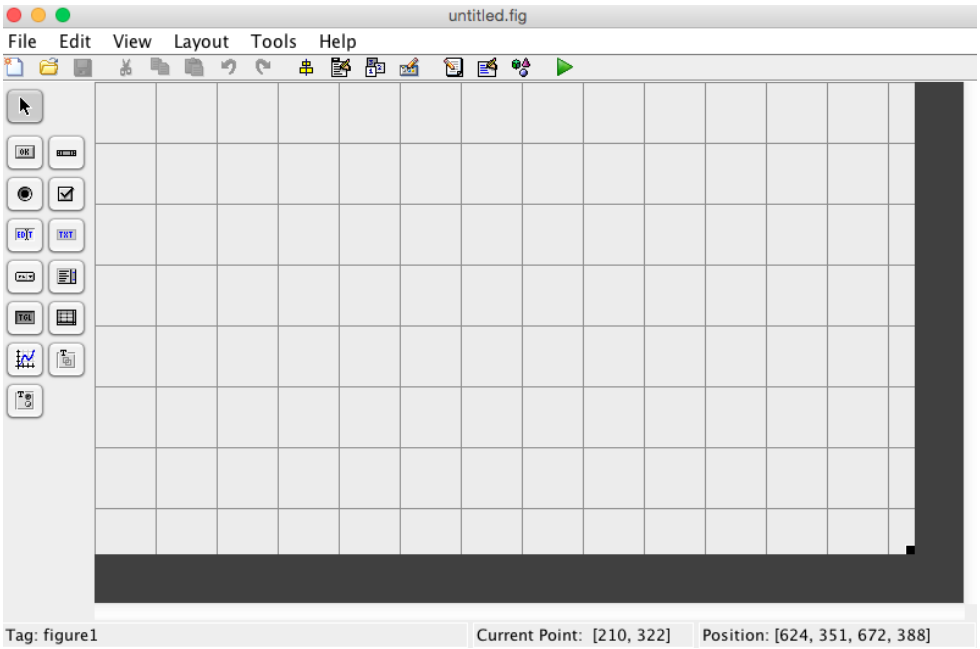


Figure 7. Default GUI figure window in MATLAB.

On the side panel shown in Figure 7, there are list of tools that can be used in the layout window. These includes *Push Button, Slider, Radio Button, Checkbox, Edit Text, Static Text, Pop-up Menu, List Box, Toggle Button, Table, Axes, Panel,* and *Button Group.* In the SODA application, the *Push Button, Edit Text, Static Text, Pop-up Menu, Toggle Button, Axes, Radio Button* and *Panel* tools were used. The SODA application uses four figures created using the GUI figure window: the Main Window, the About Window, The MAR Selection window, and the User Defined Distribution window. First the About Window, shown in Figure 8, provides the logo of DOE and ISU as well as static text identifying the names of the people involved in the project.

Figure 8. SODA *About* window.

The main SODA window, shown in Figure 9, uses the *Edit Text* option for user input. All user input on the SODA application is numeric. For example, 1.5*10^5, 1.5E5, 1.5e5 are acceptable inputs for the same value. The *Panel* option is used to group input parameters. The *Toggle Button* option is used to switch between single value input and distribution input. The *Popup Menu* option is used to select from the list of various probability distributions. The *Push Button* option makes it possible to execute code when the user presses those buttons. A *Push Button* is also used to run the program as well as to plot each parameter probability distribution plot. An *Axes* option is used to display the Probability Density Curve (PDC) of parameters as well as compute the PDC of the CED. In addition, a push button is used to find the best fit for the resultant distribution of CED using the Bayesian Information Criterion (BIC). *Radio Buttons* are used to select between available MARs.

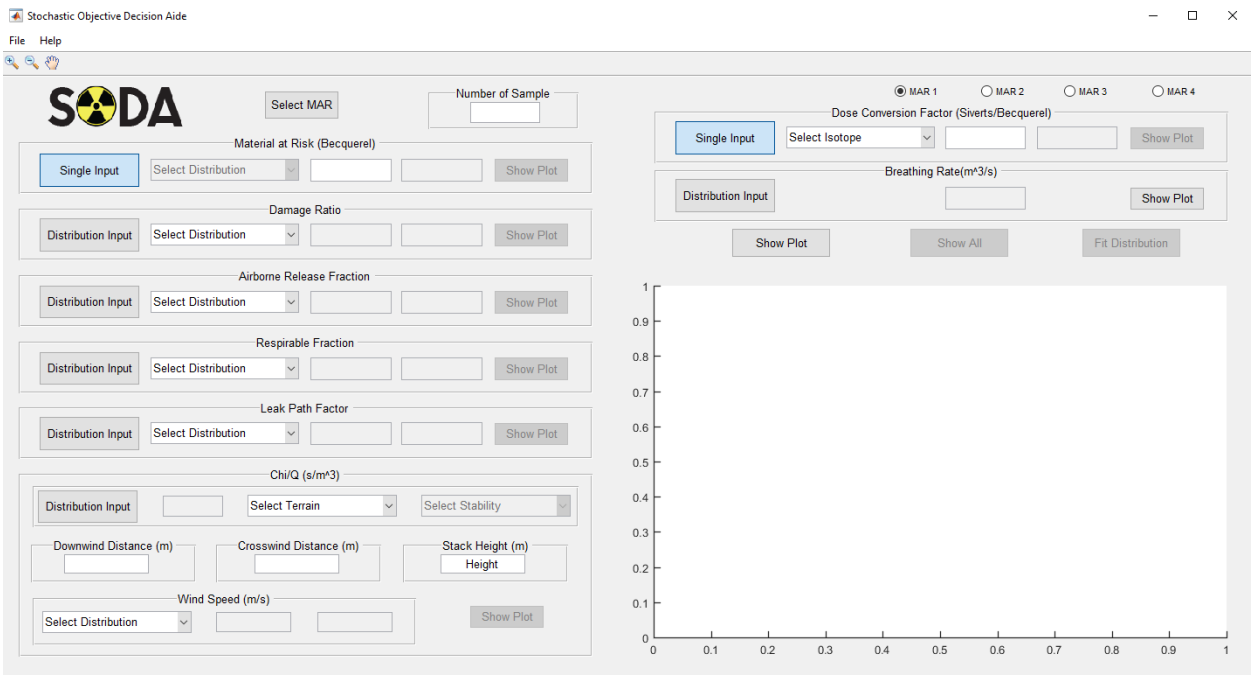


Figure 9. SODA main window.

The SODA application includes a menu and toolbar. In the toolbar there are three functions, zoom in, zoom out and pan. In the main menu there are two options, File and Help. Each of them has a further detailed sub-menu. The File menu consists of five sub-menu options which are Load Workspace, Save Workspace, Save Image, Reset Random Number and Exit. Load Workspace lets the user load a file which contains all the input parameters from a *.mat file. Save Workspace is used to save all the input parameters so that the user does not have to re-type the input data to run the program. Save Image will let the user save the plot image into a *.jpg, *.tif, or *.png image file. Reset Random Number menu will reset the Mersenne Twister random number algorithm so that calculations can be performed using different computers (or repeated) to get same result. The Exit menu will let user exit the application with a confirmation to make sure that the user is ready to exit the application.

The MAR Selection window, shown in Figure 10, gives the user the option to select MAR quantities and their associated DCF values from an easy to use GUI interface. Styled *Push Buttons* are used in a periodic table arrangement, allowing the user to select an element which has available isotope data. Once again, *Radio Buttons* let the user select between available MARs. Additional push buttons are used to export the entered MAR data to the main SODA window.

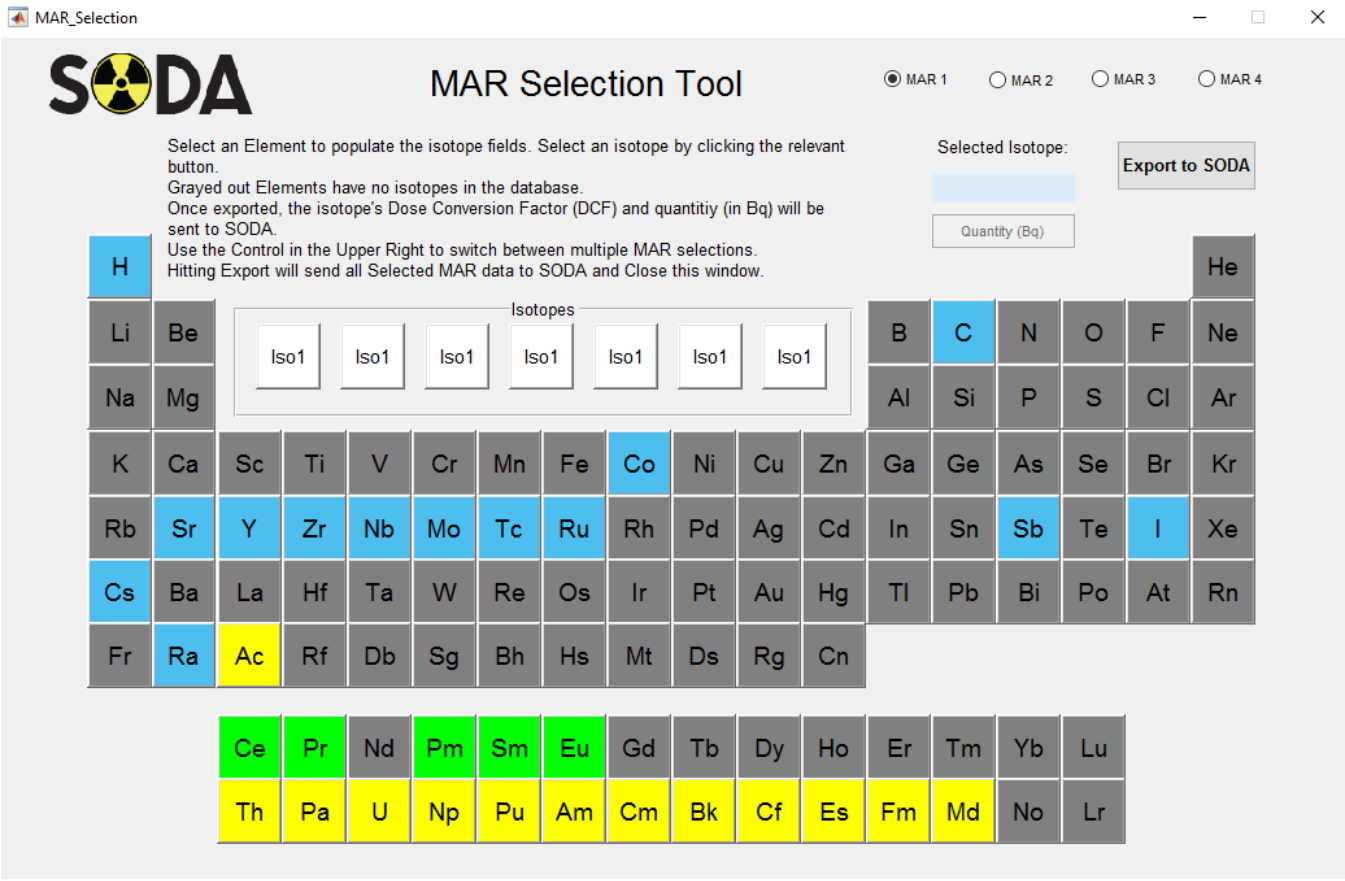


Figure 10. The MAR selection tool.

The MAR selection tool is easily accessed from the SODA main window (Figure 9 for reference) by clicking Select MAR, or using the dropdown menu in the DCF section of SODA and clicking Select Isotope. This tool allows a user to select an element, of those not grayed out, and display the available isotope data. These available isotopes will populate the buttons in the Isotopes panel, seen in Figure 10. Once an isotope is selected, this is reflected in the selected isotope field in the upper right hand region of the MAR selection tool. Once this is selected, the user can input the MAR quantity and export the data to the SODA main window. The option to select multiple MARs is present, and is accessible from the radio buttons in the top right hand corner of the tool.

3.3. Monte Carlo Method

SODA uses Monte Carlo techniques in which the application uses repeated random sampling to compute the CED distribution. SODA can handle up to 10^8 samples depending on the available computer memory (additional information on this topic is contained in the User's Manual, Appendix A). For each input parameter, the user can select from up to five different probability distributions; Normal,

Uniform, Beta, Exponential, and Log Normal, or the user can define their own custom distribution. The user can verify the resulting input parameter distribution by clicking the show plot button for each parameter. Based on the distribution selection, different user input values are required. For a normal distribution, the user is required to input the mean and standard deviation, for the Beta distribution, the user is required to input the alpha and beta parameter. For a uniform distribution, the user is required to input the upper and lower limit. For the exponential distribution, the user is required to input the mean. Finally, for the Log Normal distribution, the user is required to input the mode and scale parameter (also known as the normal standard deviation.) Once user provides the input parameter values, SODA creates a string of random numbers to sample the specified distributions.

3.4. Distribution Mathematics

The normal distribution, also known as Gaussian distribution, is a common distribution in which the probability density function (PDF) has a bell shaped curve. The PDF equation of a normal distribution is

$$f(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3}$$

where μ is the mean or expected value and σ is standard deviation. Examples of normal distributions using different mean and standard deviation values are provided in Figure 11.

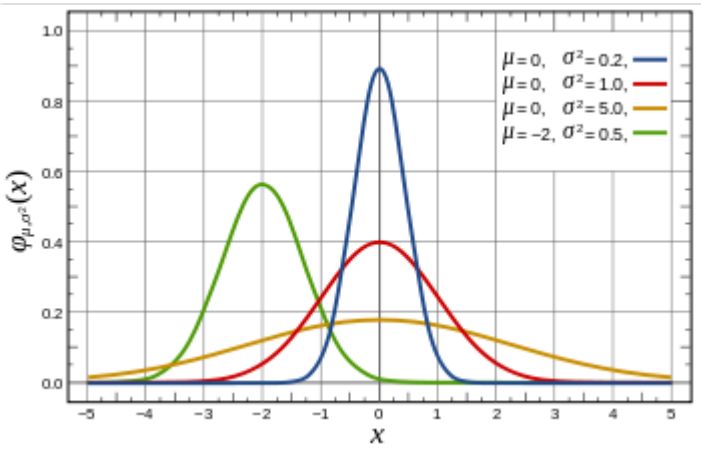


Figure 11. Normal distributions.

The probability distribution in which the logarithm of a variable is normally distributed is log normal distribution. The PDF equation of a log normal distribution is:

$$f(x\,|,\mu,\sigma) = \begin{cases} \frac{1}{\sigma x\sqrt{2\pi}} e^{-\frac{(log(x)-\mu)^2}{2\sigma^2}} & x > 0 \\ 0 & \text{Otherwise} \end{cases}$$

A Log-Normal distribution plot is shown in Figure 12. Discussion of the SODA specific treatment of this distribution (use of mode instead of location parameter etc.) is made in the user's manual.

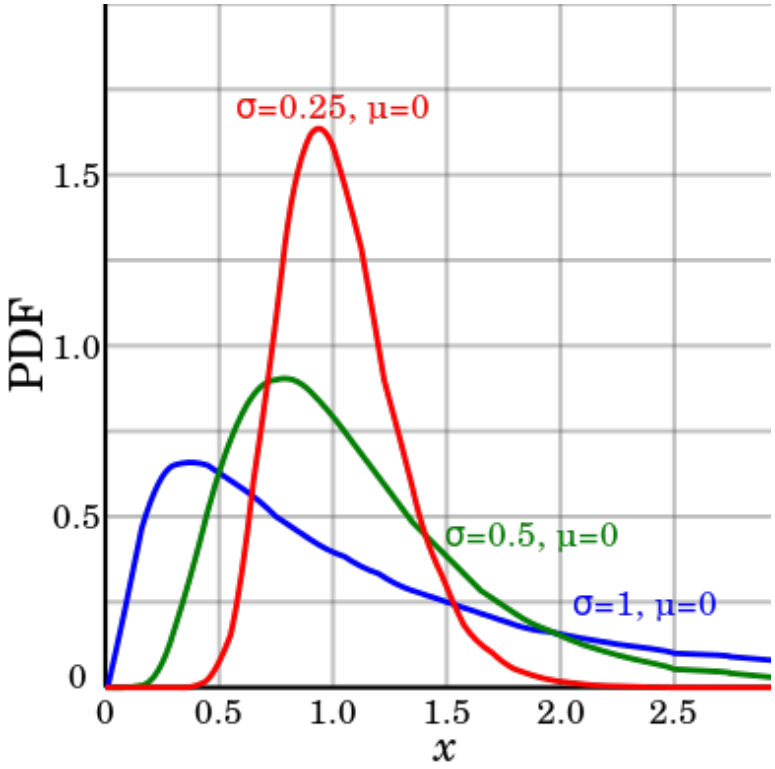


Figure 12. Log-Normal distribution.

The beta distribution, plotted in Figure 13, is defined over the interval of zero to one with two positive shape parameters, alpha and beta. The PDF equation of a beta distribution is

$$\frac{x^{\alpha-1}(1-x)^{\beta-1}}{\mathrm{B}(\alpha,\beta)} \quad (4)$$

where B is beta function. Since beta distribution is defined between zero and one it can be used for damage ratio and leak path factor.

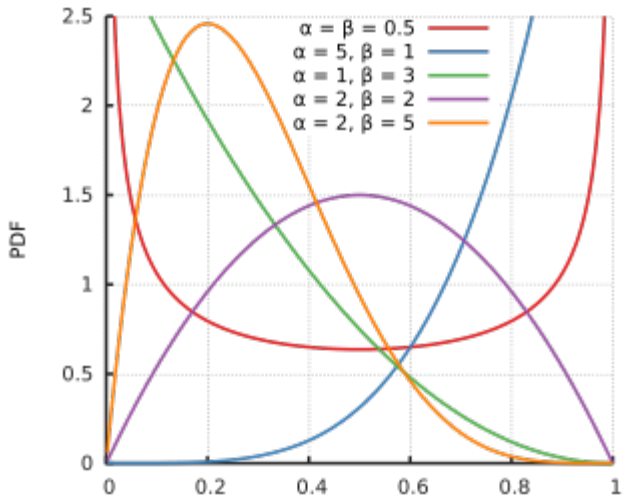


Figure 13. Beta distribution.

The uniform distribution is uniform continuous distribution in which all possible values between the minimum and maximum value are equally probable. The PDF equation of a uniform distribution is

$$\begin{cases} \frac{1}{b-a} & \text{for } x \in [a, b] \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where b is the maximum value and a is minimum value. A uniform distribution plot is shown in Figure 14

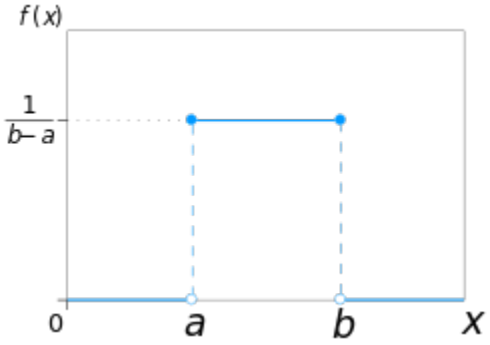


Figure 14. Uniform distribution.

The exponential distribution describes a process in which events occur continuously and independently at a constant average rate lambda, see Figure 15. The exponential distribution equation is

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases} \tag{6}$$

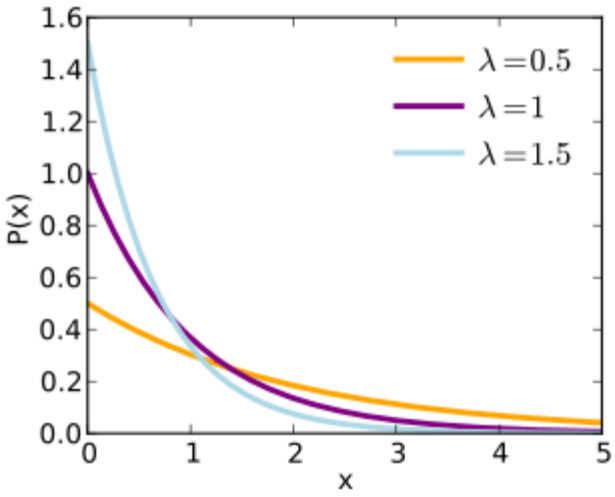


Figure 15. Exponential distribution.

In addition to the distributions described above, a specialized BR distribution is available in SODA. The distribution is intended to be more representative than a single point value. It is applicable for scenarios where the event can occur at any time throughout a 24-hour period. A 24-hour period was used with an adult male BR consisting of eight hours per day resting, eight hours per day sitting, four hours per day of light exercise, and four hours per day of heavy exercise. The distribution is sampled by scaling the random number generator and selecting the breathing rate reflected in Table 1 which uses data from ICRP 89.

Table 1. Breathing rate data.

Status	Breathing rate(m^3/s)
Resting	1.25e-4
Sitting	1.5e-4
Light Exercise	4.17e-4
Heavy Exercise	8.33e-4

The resting BR and sitting breathing rate are sufficiently similar to allow using the sitting BR for both. The sampling strategy results in the 1.5e-4 value being selected 66% of the time and the light exercise value, 4.17e-4, and the heavy exercise vale, 8.33e-4, each being selected 17% of the time.

A simplified damage ratio experiment was performed to support a rudimentary estimate for the damage ratio distribution that could be applied to certain scenarios. The experiment investigated the damage

ratio for dropped radioactive material containers. The damage ratio experiment involved both 1-m and 3-m drop tests onto a concrete surface. The containers involved in the experiment were one-pint capacity and one-quart capacity. The containers used press fit lids similar to paint cans. Each container was filled 3/4 full with rock salt to simulate radioactive material. Twelve containers of each capacity were used in the experiment. The one-pint capacity containers were initially dropped from the 1 m height. A total of 100 drop tests with one-pint capacity containers was performed from the 1 m height. After each container was dropped, it was visually examined to qualitatively determine if the container had breached. Containers that were not breached were subjected to additional drop testing. Containers that breached were photographed, the amount of salt that escaped from the container was quantified, the salt was returned to the container, the lid was reinstalled, and the container was reused for subsequent drop testing.

A similar approach was used for the 3 m drop testing. The 3 m drop testing involved both the one-pint capacity containers and the one-quart capacity containers. A total of 300 container drop test were performed. No container failure mode other than lid failure was observed. Approximately 20% of the container drops resulted in damage sufficient to allow release of radioactive material.

The drop testing experiment provides a rudimentary understanding of the likelihood of container breaching due to a drop event which can help understand the appropriate damage ratio distribution to select for a dose consequence calculation associated with a particular accident scenario. Since the containers used in the experiment have no quality assurance pedigree and are very thin walled, the experiment is intended to provide some rudimentary insight into a reasonable, but conservative, damage ratio distribution for accident scenarios involving multiple radioactive material containers. Appendix C contains the drop test results data.

To make SODA more user friendly and applicable in more situations, a user defined distribution option was added. SODA offers a clickable plot for the user to “draw” the desired distribution and also allows a typed entry option. This feature was created using GUIDE in MATLAB. A GUI allows the user to select the entry method to be used for inputting the distribution values. The selection is made from a drop down menu. More information and instruction for use of the user defined distribution feature can be found in the SODA User Manual (Appendix A). The user defined distribution GUI can be seen in Figure 16.

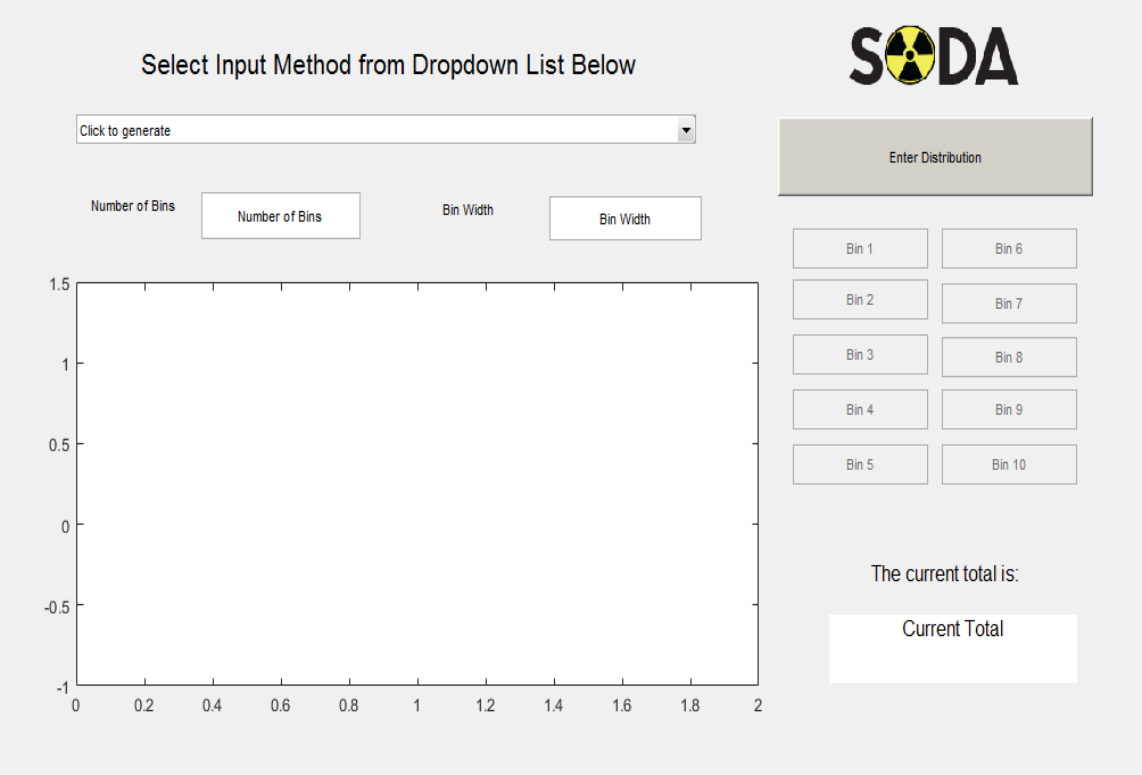


Figure 16. SODA user defined distribution GUI.

With the method selected, the user then must enter the number of bins to be used and the bin width. The user defined distribution feature currently only allows constant width bins, and variation in bin size is not permitted. If the entry method selected is the clickable plot option, the user may then begin to click the positions on the axes provided on the GUI. The application sums the values clicked and determines if the total probability for the entire distribution sums to one. If the sum does not equal one, the feature will give the user a notice indicating that the probabilities must sum to one and the axes will clear allowing the user to make another attempt. Four attempts are allowed before the user will be returned to the main SODA screen. The notice for a distribution not summing to one can be seen in Figure 17.

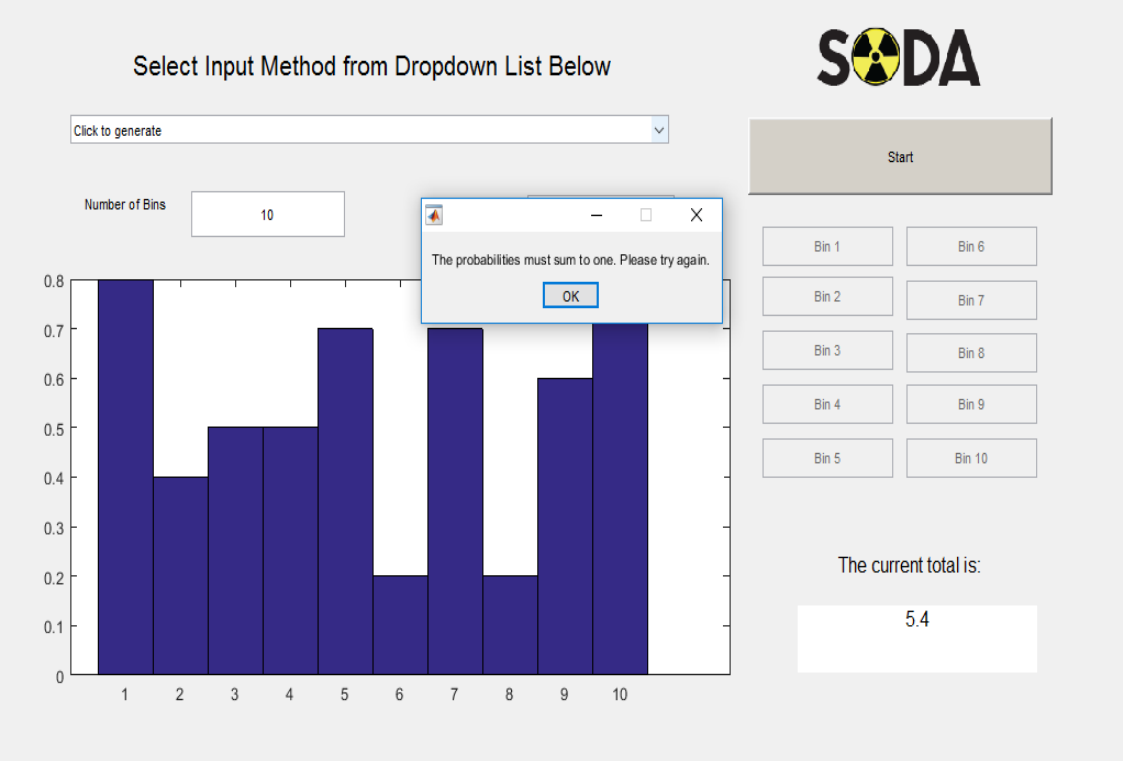


Figure 17. Notice received when user defined distribution is inappropriate.

If the distribution does sum to one, then the application will show the user a plot of what they have clicked and ask if it is correct, see Figure 18.

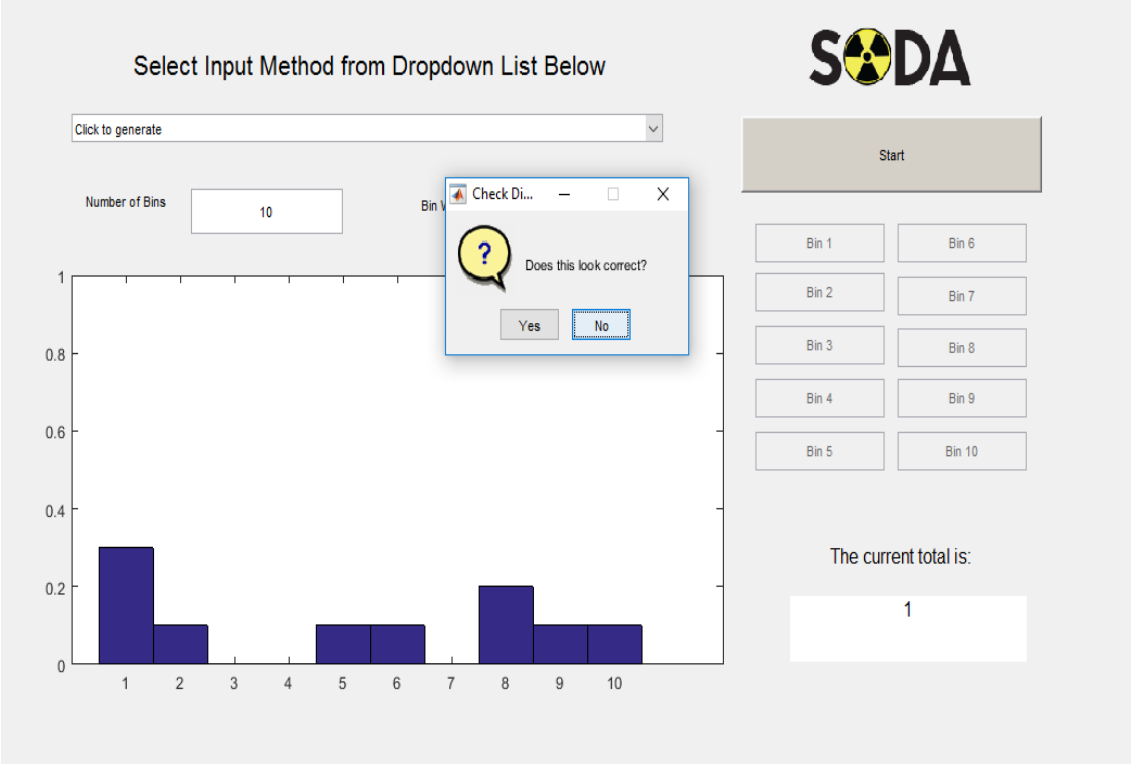


Figure 18. User defined distribution verification.

If correct, then the distribution entered is passed back to SODA. If the user determines the distribution is not correct, then they are allowed another chance to enter the distribution. This situation also allows four attempts before returning to the main SODA screen.

If the typed entry method is selected, then the user still must enter the number of bins and the bin width. If the number of bins entered is greater than 10, the application will give the user a message indicating the most bins allowed for this distribution entry method is 10 and the bin number will be set to 10. Again, as with the clickable entry option, variations in bin width cannot be accounted for in this distribution option. At this point, the GUI will enable the textboxes for the distribution typed entry. The application is set up to only enable a specific number of textboxes based on the number of bins entered in the number of bins textbox. This is a safeguard to prevent the user from entering the distribution numbers for bins not being used and is used as check to ensure the user has entered the correct number of bins for the number of distribution values that will be entered. Once all of the distribution values have been typed into the respective textboxes, the user will select the start button. The start button will generate a plot of the distribution entered by the user in the textboxes and the application will perform a check to ensure the distribution values sum to one, see Figure 19.

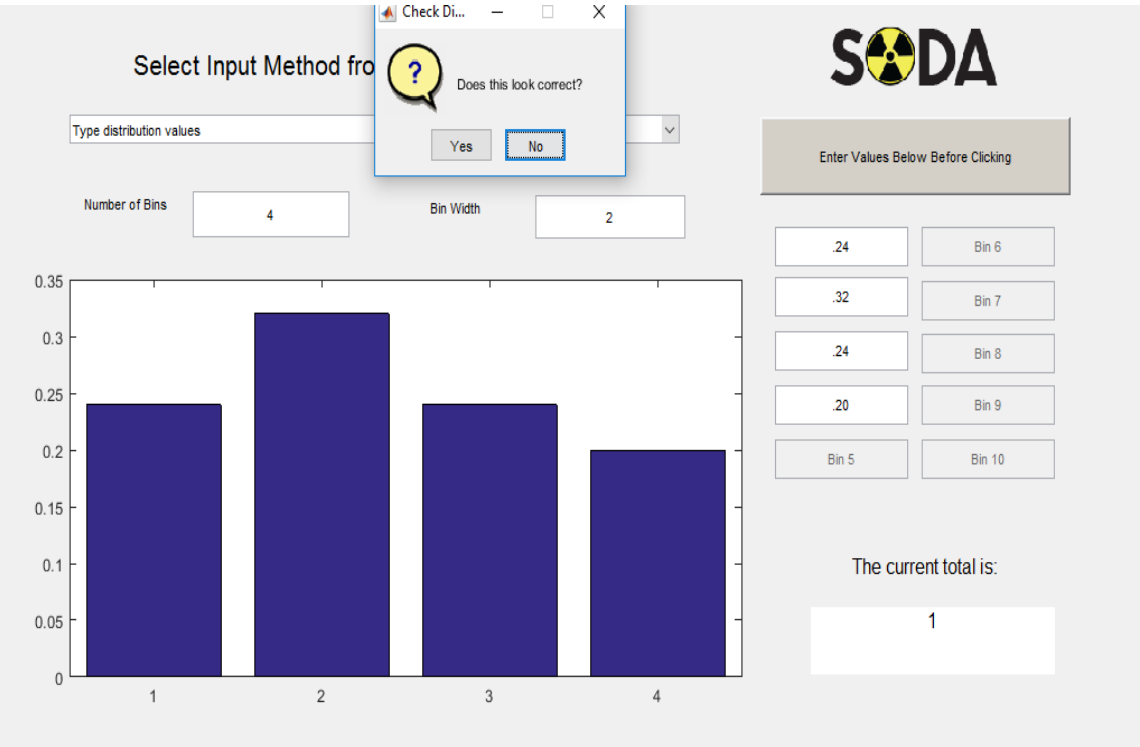


Figure 19. User defined distribution typed entry.

If the sum is one, then the application will ask the user if the plot looks correct, as in the figure above, and if yes, the data will be stored as the distribution. If it does not look correct, or if the distribution values do not sum to one, then the user will be notified of the issue and given a chance to fix the typed distribution values.

3.5. Bayesian Information Criterion

The Bayesian Information Criterion (BIC) method is used by SODA to find the best possible probability distribution for the resultant CED distribution. The BIC value is computed using a likelihood function of the estimated model. A likelihood function is the probability or probability density for the occurrence of a sample configuration x1, x2, x3,..., xN given that the probability density f(x , alpha) with parameter alpha is known. The smaller the BIC value indicates a better model fit. An example best fit plot is provided in Figure 20.

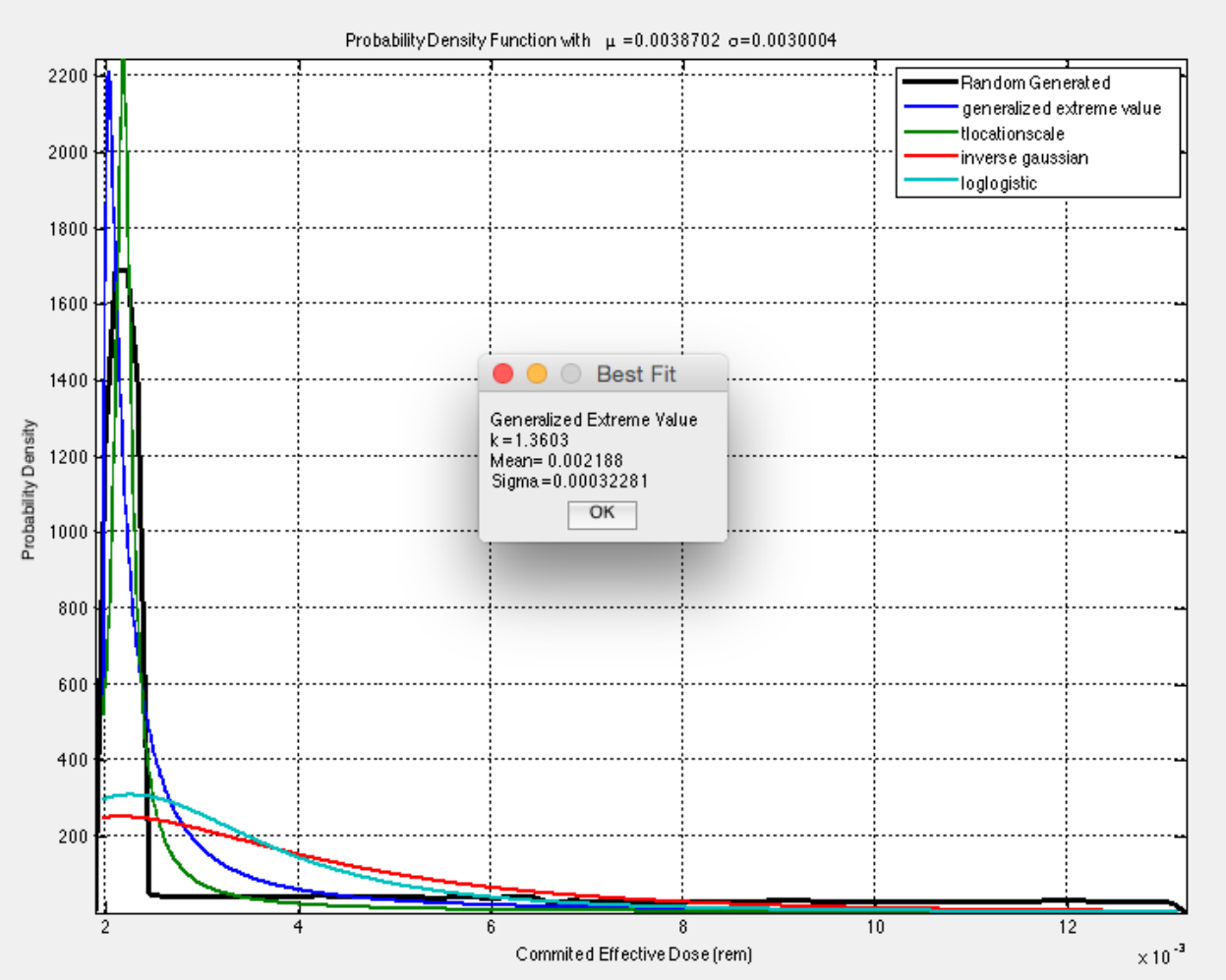


Figure 20. Best fit example.

4. COMPARISON WITH RSAC

SODA was verified by comparing results obtained using RSAC 7.2.0 calculations for a simplified scenario involving the release of 1 Ci of Pu-239. Fixed parameters used for the verification case are provided in Table 2. Results obtained from RSAC, SODA, and a hand calculation are provided in Table 3 and show good agreement using fixed parameters in SODA.

Table 2. Verification case input data.

Parameter	Value
ARF	2.00e-3
RF	0.3
Wind Speed, mixing height, release height	1 m/s, 400 m, 0 m
Pasquill Class	F
Meteorology	Hillsmier-Gifford Sigmas
Downwind distance	5000 m
Calculated *X*/Q	5.925E-5
Breathing Rate	3.33E-04 m^3/sec
Pu – DCF Clearance Class	F

Table 3. Verification results comparison.

	RSAC 7.2.0	SODA	Hand calculation
Total CED (rem)	5.26E-3	5.256E-3	5.256E-3

5. CONCLUSION

A Monte Carlo based code system has been developed to stochastically analyze radiological material release scenarios providing potential dose distribution results. The computer code, named Stochastic Objective Decision-Aide, provides a portable and simple to use tool to better inform decision making associated with establishing nuclear facility material-at-risk limits and safety SSC selection. Traditional radioactive material release modeling codes typically provide a bounding single point estimate of receptor dose. While this approach attempts to bound the dose estimate, it falls short in providing quantification of the expected value and the uncertainty associated with the dose estimate. This is particularly problematic when one considers the lack of governing distribution identification for input parameters. Thus, decisions regarding potential doses are frequently overstated, leading to excessively conservative material-at-risk limits and potentially over selection of safety-systems structures, or components.

Rather than producing a point estimate of the dose, SODA produces a dose distribution result to allow a deeper understanding of the dose potential. SODA allows users to select the distribution type and

parameter values for all of the input variables used to perform the dose calculation. SODA then randomly samples each distribution input variable and calculates the overall resulting dose distribution. In cases were an input variable distribution is unknown, a traditional single point value can be used. SODA was developed using the MATLAB coding framework. The software application has a graphical user input. SODA can be installed on both Windows and Mac computers and does not require MATLAB to function.

It is important to note that SODA does not replace or compete with codes such as MACCS or RSAC, rather it is viewed as an easy to use supplemental tool to help improve risk understanding and support better informed decisions.

APPENDIX A:

SODA User Manual

SODA USER MANUAL

Appendix A

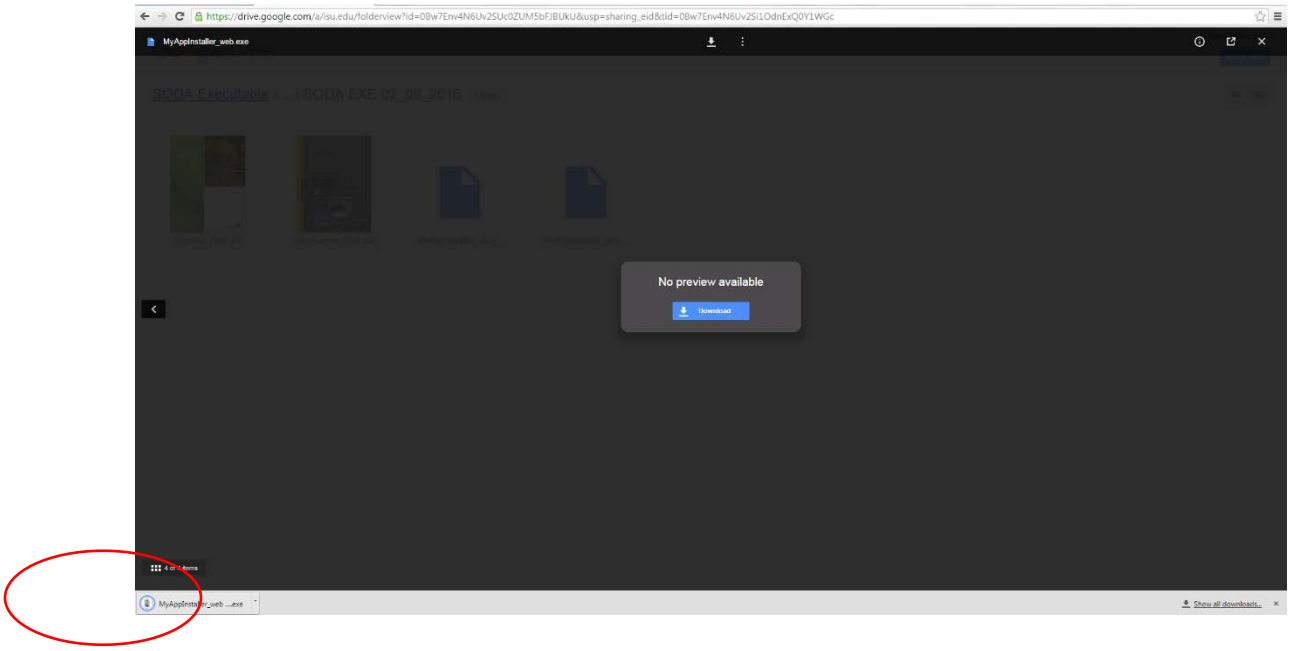
Contents

1. INSTALLATION INSTRUCTIONS

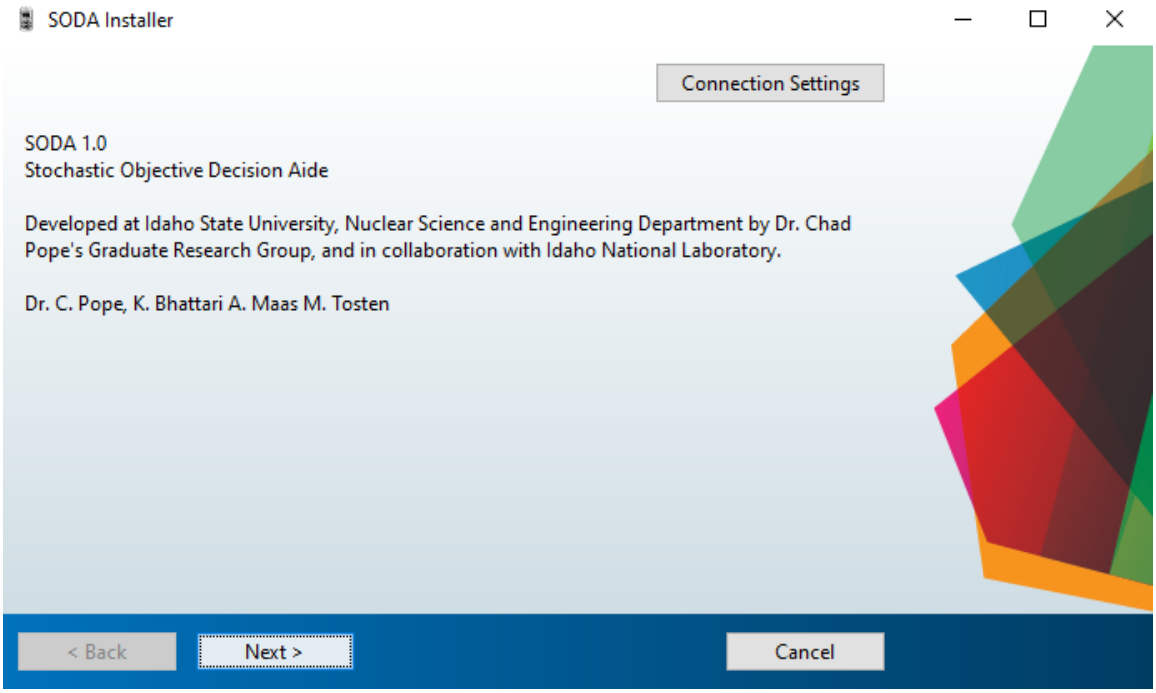
The latest version of the MATLAB runtime environment, which is required to use the SODA program, requires a 64-bit operating system on all platforms.

1.1 WINDOWS INSTALLATION INSTRUCTIONS

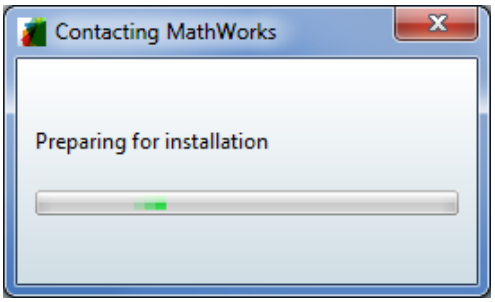
To begin, download the program. The download file will be at the bottom of your browser, or located in your default download directory. Click to open the installation program.



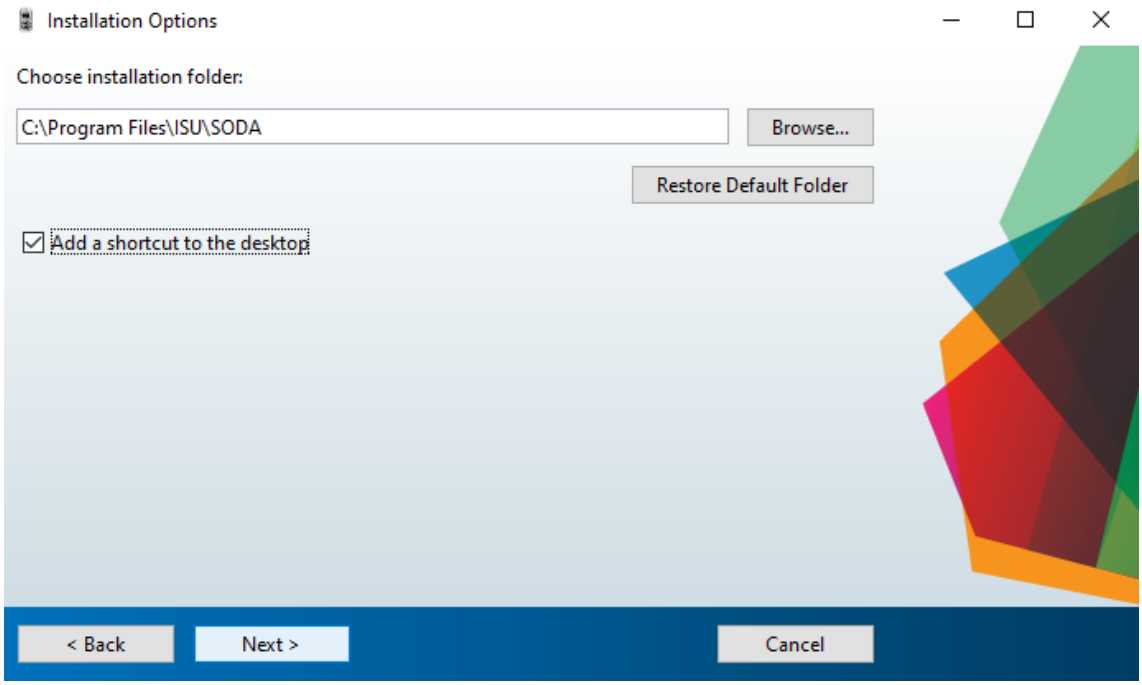
The install application may take a few minutes to load. When it is done, the SODA installer window will open.



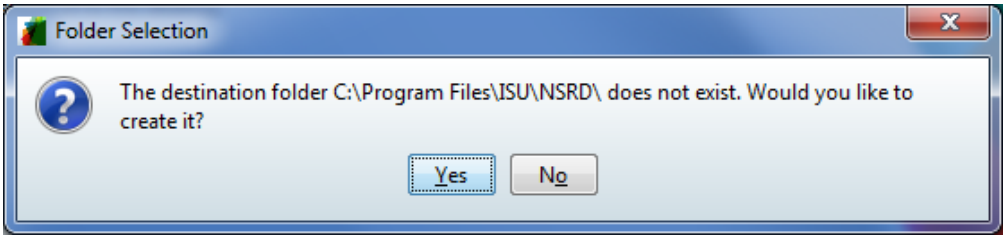
Click the 'Next' button to continue installation.



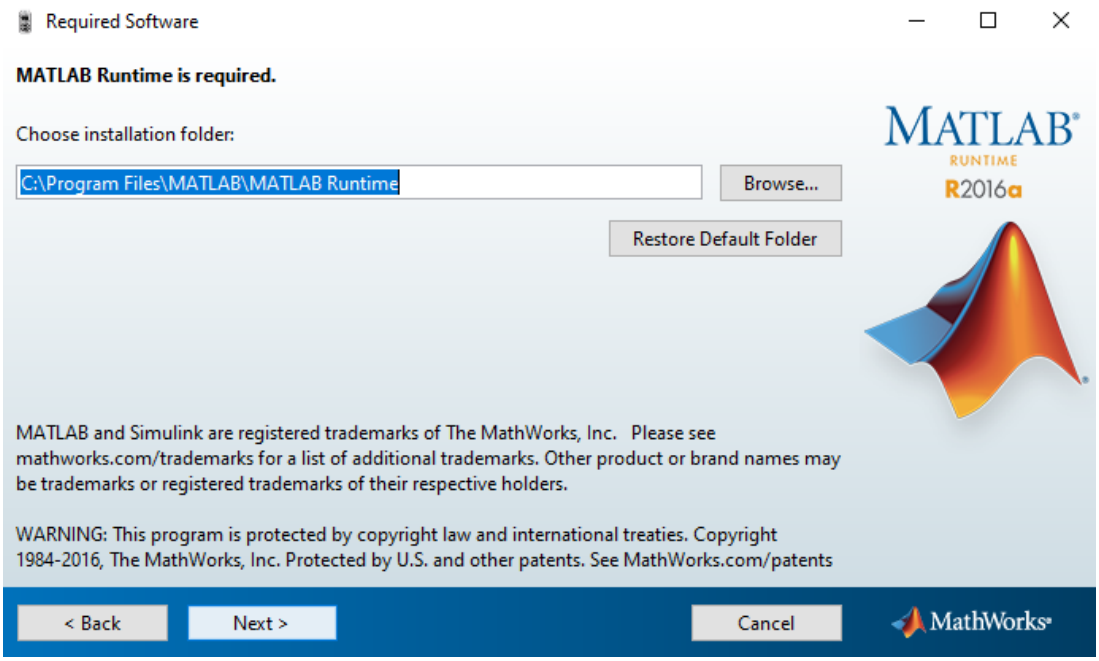
Please wait for MathWorks to prepare the installation then you will see the installation options window.



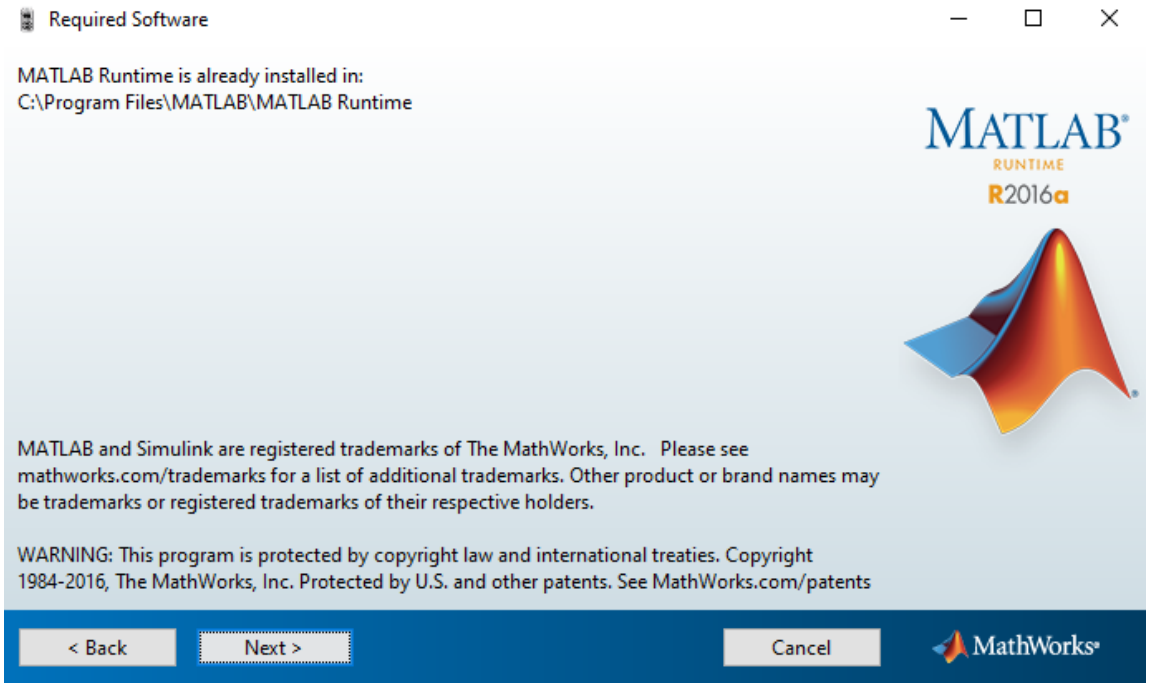
Click on the browse button to set a location for the program to be installed. You may check the box if you would like a shortcut to SODA added to your desktop. Click the 'Next' button to continue installation. If you selected to install the application to a folder that does not yet exist, you will see the following pop up window.



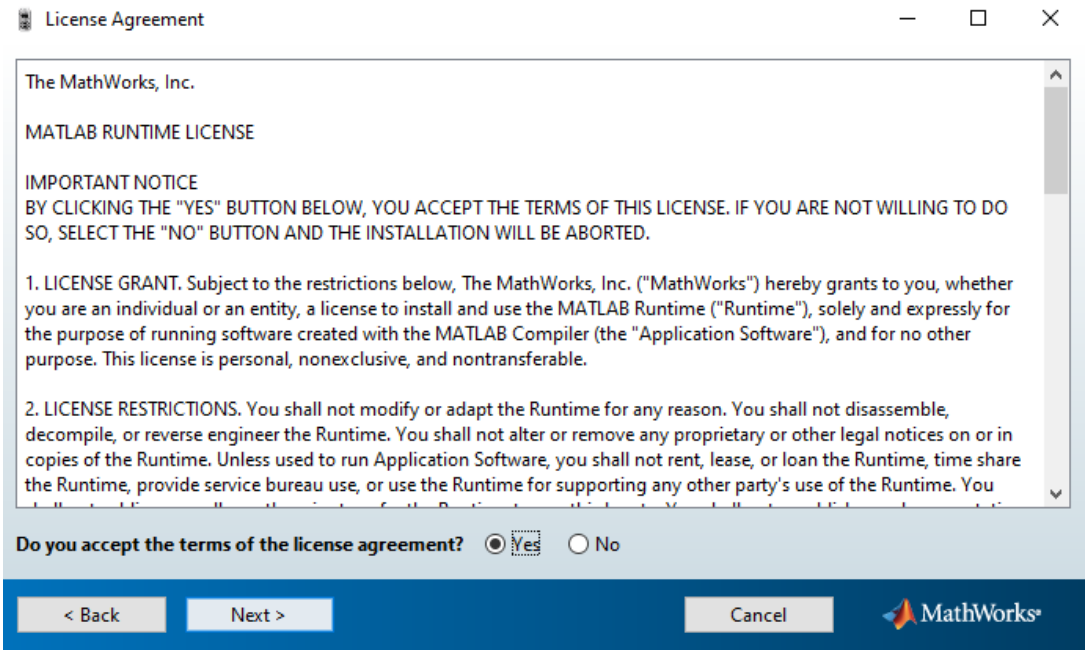
Click 'Yes' if you would like to create the folder or 'No' to change the location for installation. Once 'Yes' is selected, if MATLAB Compiler Runtime is not yet installed on the machine, the installer will then ask for the folder to install it.



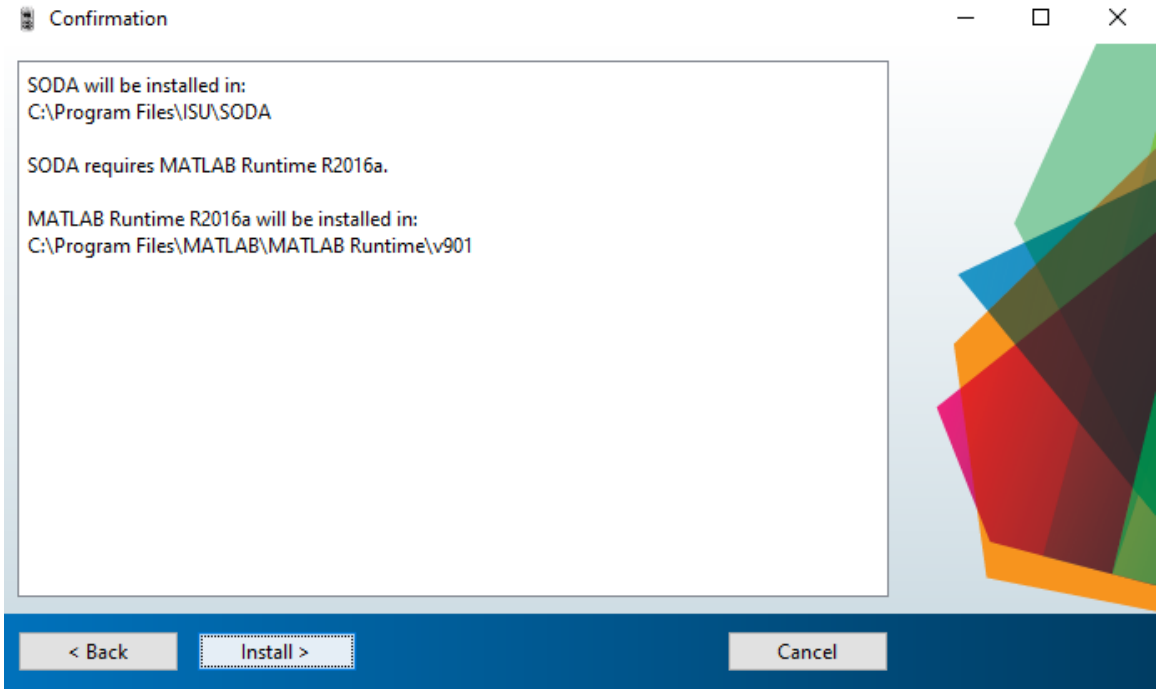
Select the folder for this to be installed and then click ‘Next’. If MATLAB Compiler Runtime was already installed on the machine, you will see the following window. Click ‘Next’ to continue.



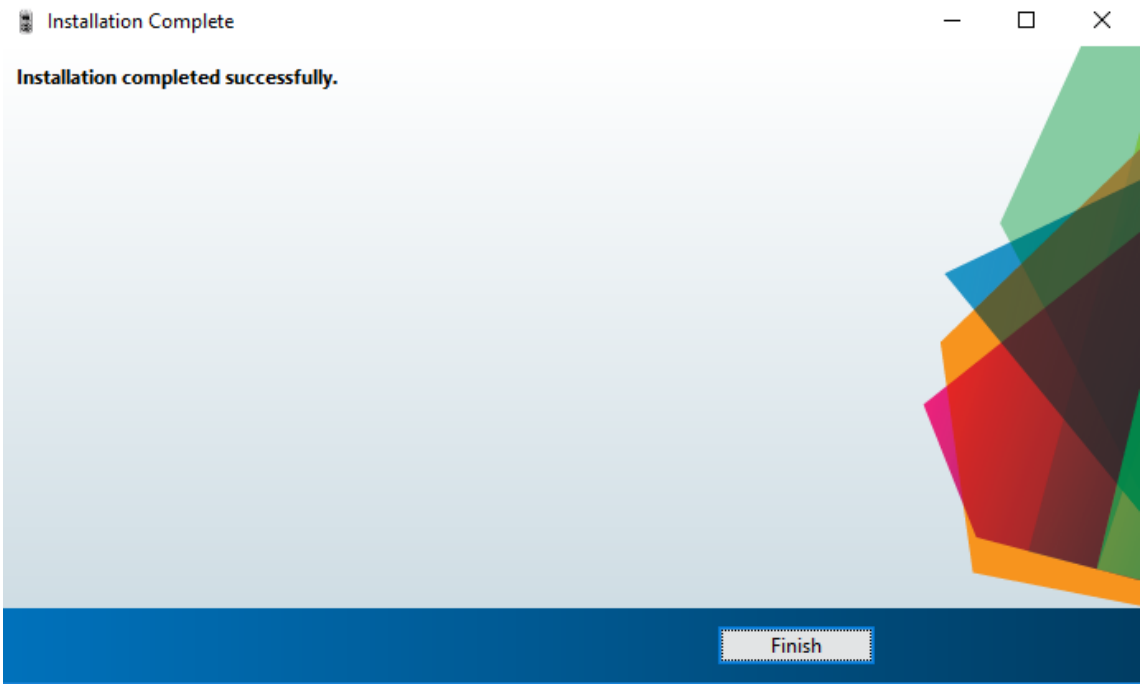
Check ‘Yes’ or ‘No’ to accept or decline the License Agreement, then click ‘Next’.
(Note: If you check ‘No’ the program will cancel.)



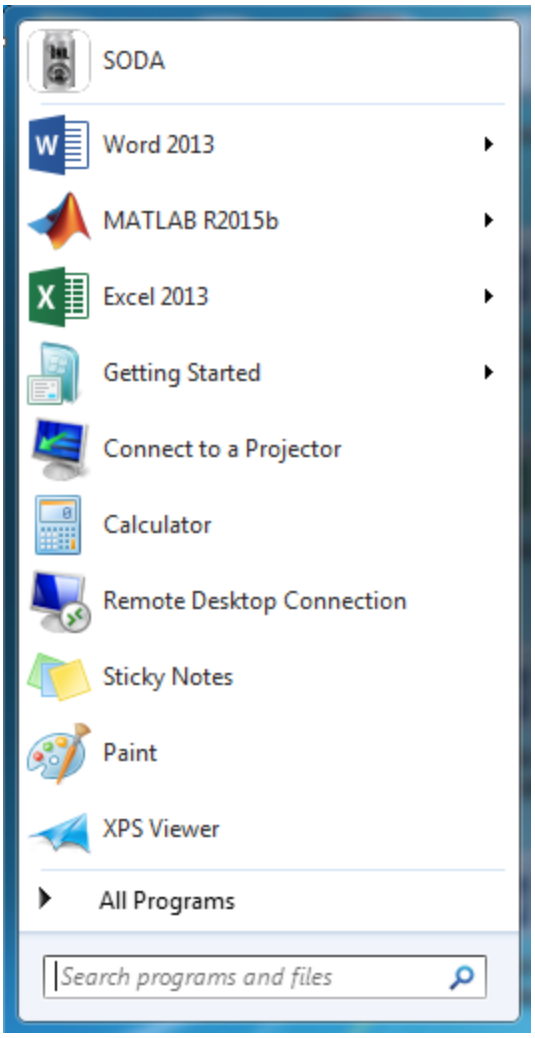
The Confirmation window will show up. Click ‘Install’ to start downloading the program.



The download will then show a window keeping track of the percentage downloaded 0%-100%. If you need to pause the program for any reason, you may click ‘Pause’. When the program reaches 100%, the ‘Installation Complete’ window will show.



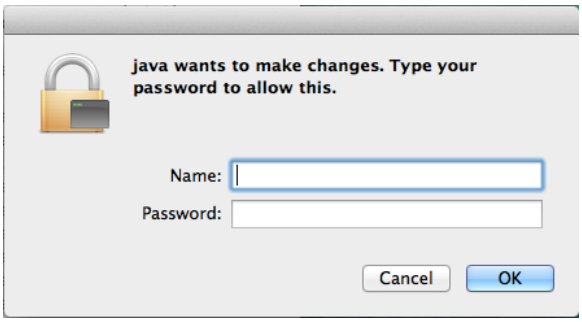
Click ‘Finish’ on this window to complete the installation process. At this time, the program can be found in several places. It will be located in the start menu as pictured below.



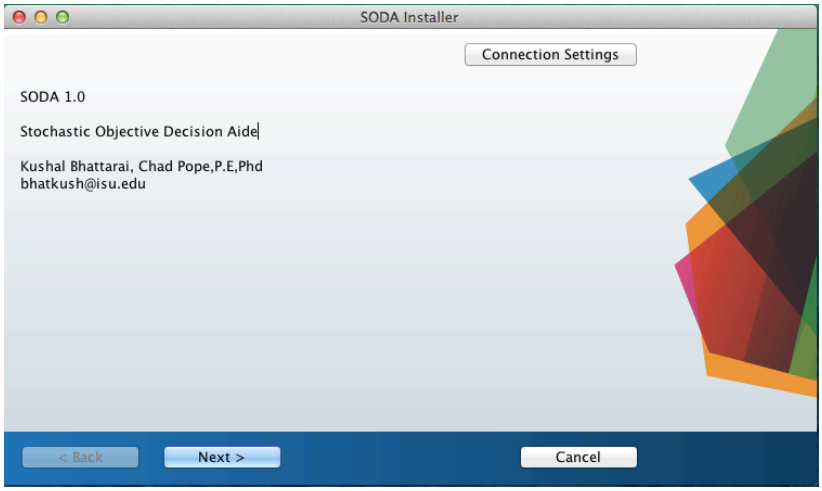
It will also be located on the desktop if you selected to allow a desktop icon to be created, and finally it will be located in the folder that you specified during the installation.

1.2 MAC INSTALLATION INSTRUCTIONS

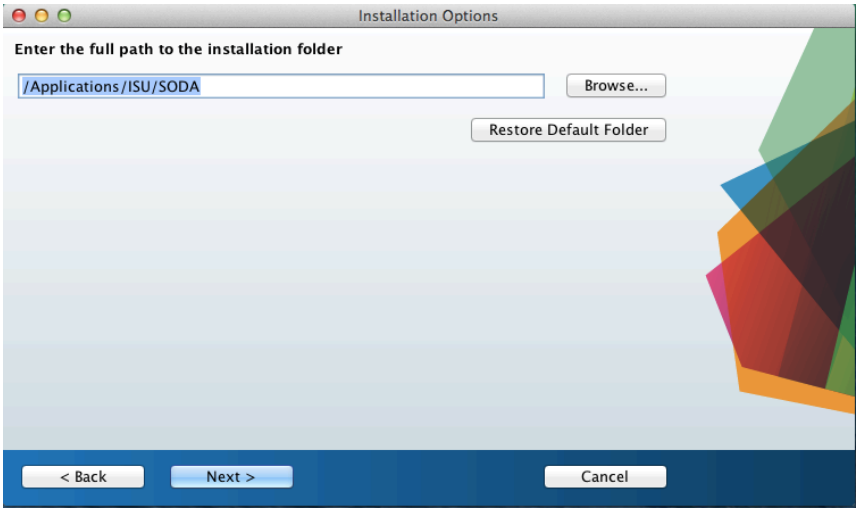
Download the program and the file will be located in the download folder. Click to open the installation program. The install application will take a few minutes to load. Please wait then put in your name and password.



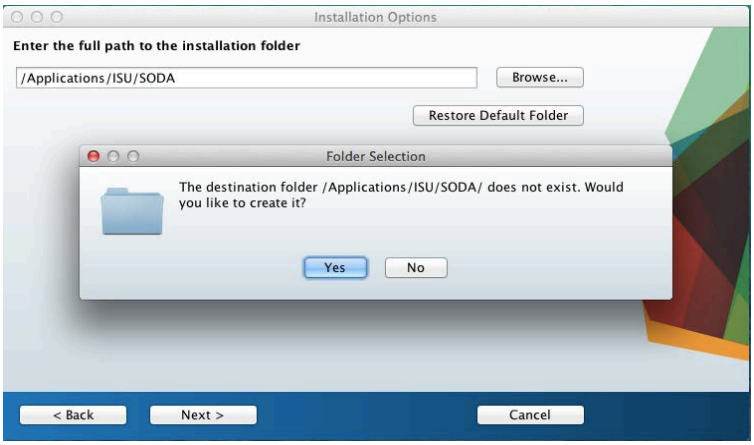
When you see the SODA installer window, click ‘Next’ to continue the installation.



Please wait for MathWorks to prepare the installation, when it is ready, you will see the ‘Installation Options’ window.

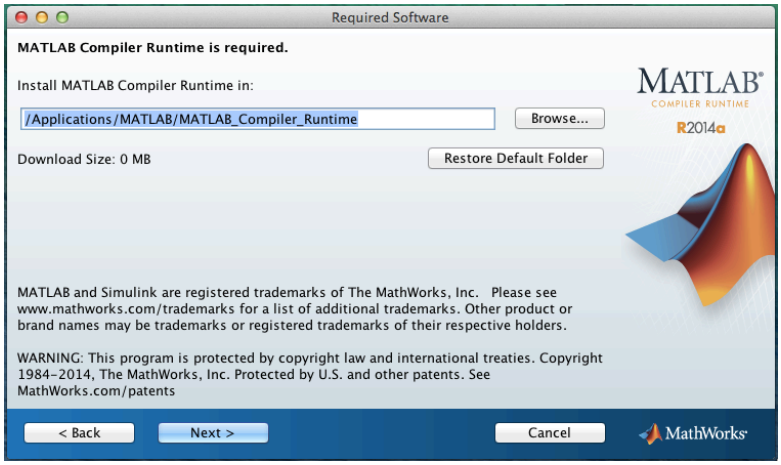


Click ‘Browse’ to set a location for the program to be installed.



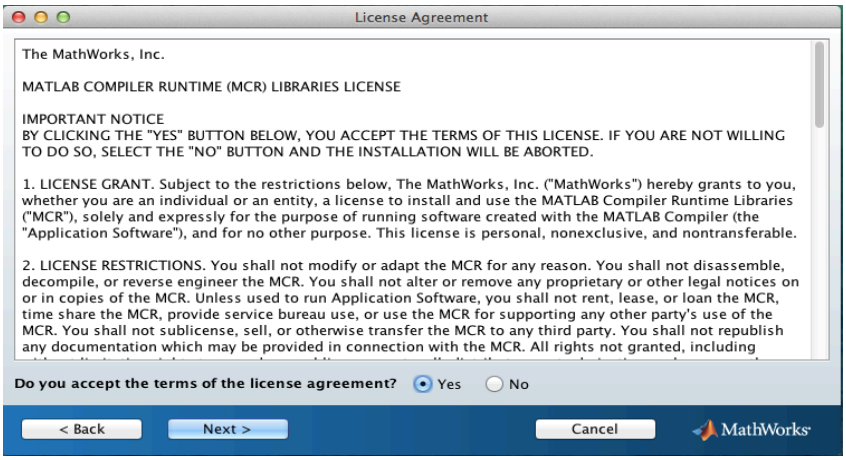
Click ‘Next’ to continue installation.

The ‘Required Software’ window will show up. Click ‘Browse’ again to set a location for the program to be installed.

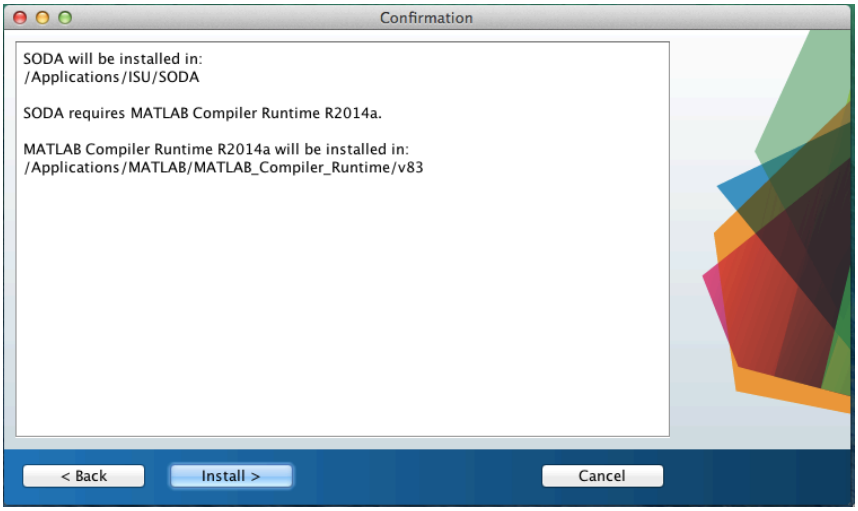


Click ‘Next’ to continue installation and the ‘License Agreement’ window will come up.

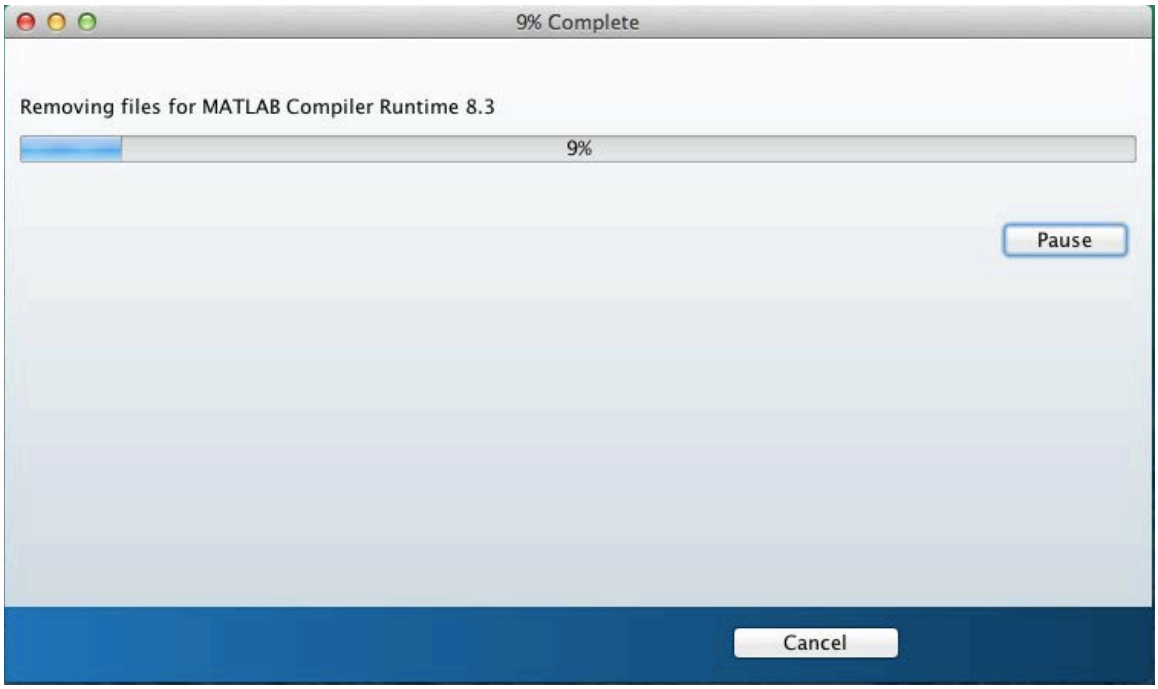
Check ‘Yes’ or ‘No’ to accept or decline the License Agreement, then click ‘Next’.
(Note: If you check ‘No’ the program will cancel.)



The Confirmation window will show up. Click ‘Install’ to start downloading the program.



The download will then show a window keeping track of the percentage downloaded 0%-100%. If you need to pause the download for any reason, you may click ‘Pause’.

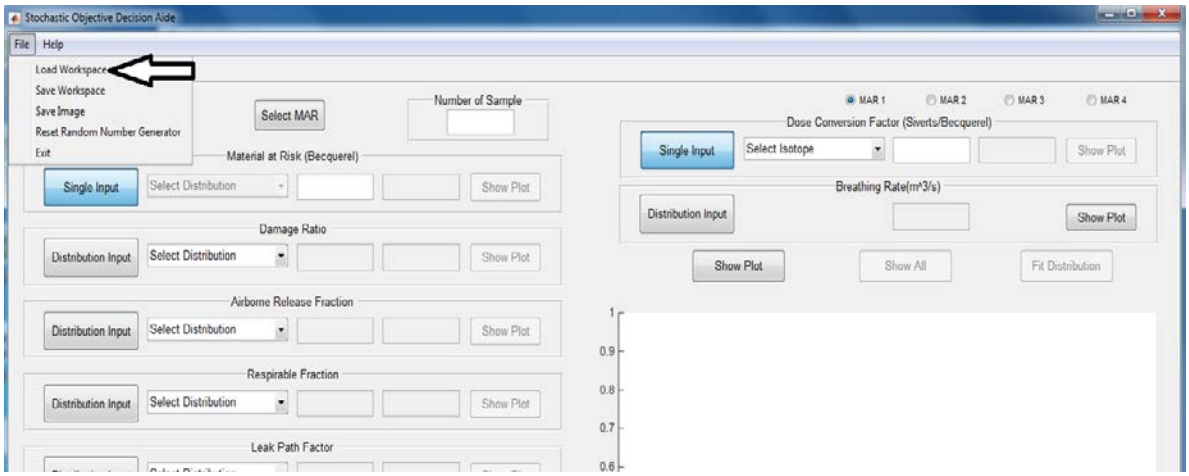


When the program reaches 100%, the ‘Installation Complete’ window will show. Click ‘Finish’ on this window to complete the installation process. At this time, the program can be found in the folder specified during installation.

2. LOADING AND SAVING

2.1 FILE LOADING

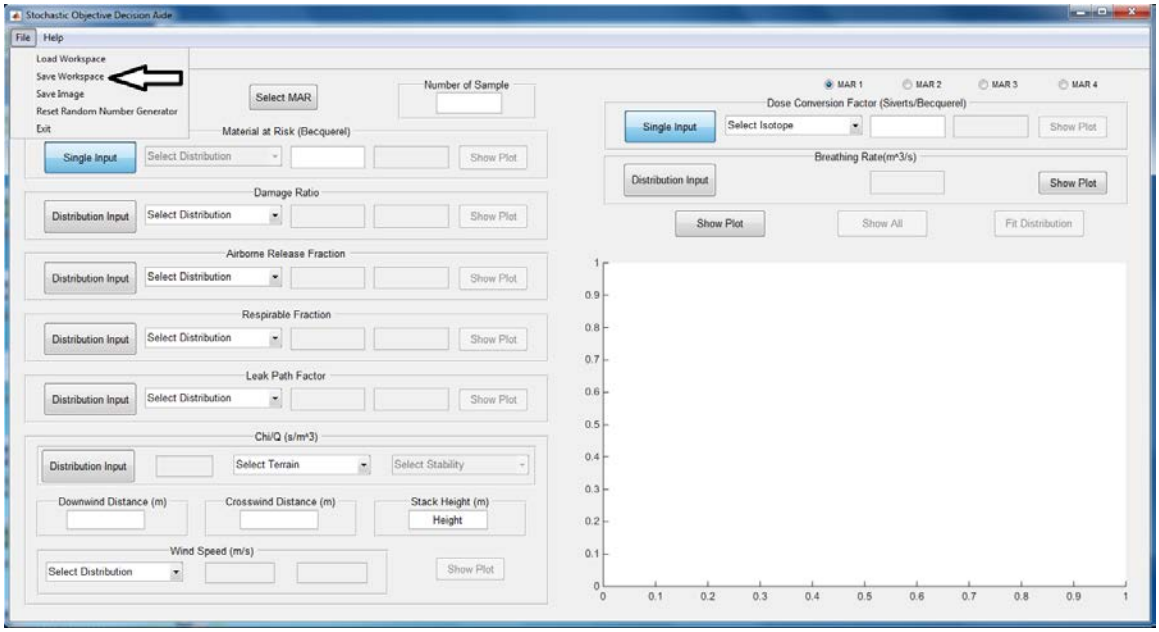
To load a workspace, click ‘File’ at the upper left hand corner of the SODA application. Then select ‘Load Workspace’ as shown below. This will allow you to select from saved files on the machine you are using.



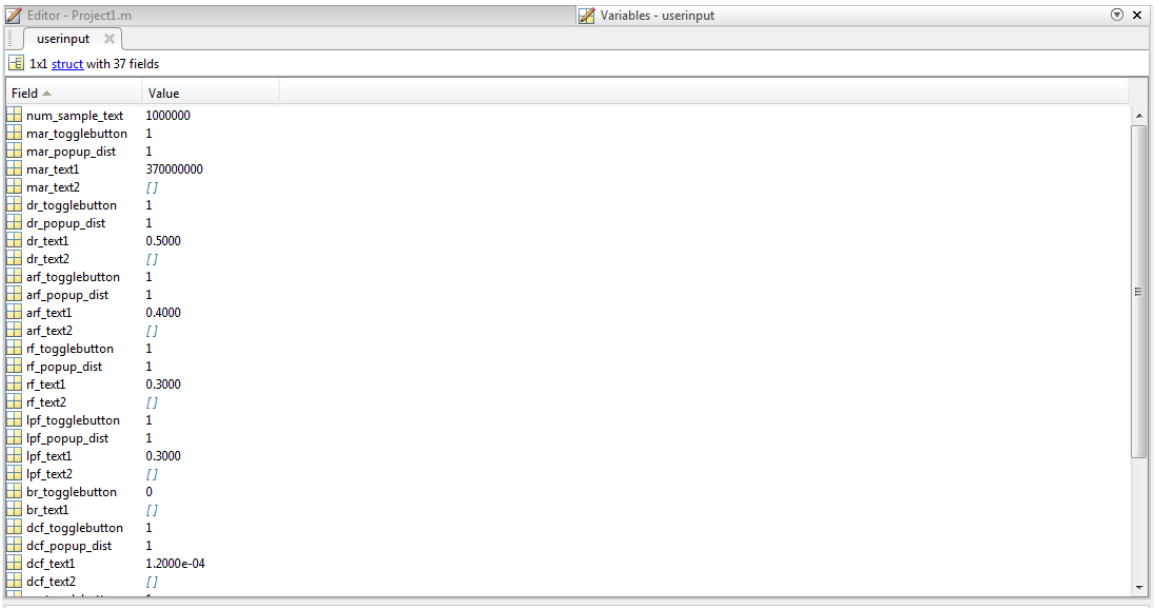
2.2 SAVING

2.2.1 FILE SAVING

To save the workspace that you are currently work on within the SODA application, click the ‘File’ button in the upper left hand corner of the program. Then select ‘Save Workspace’. This will allow you to save the workspace to any location on the machine that you are using.

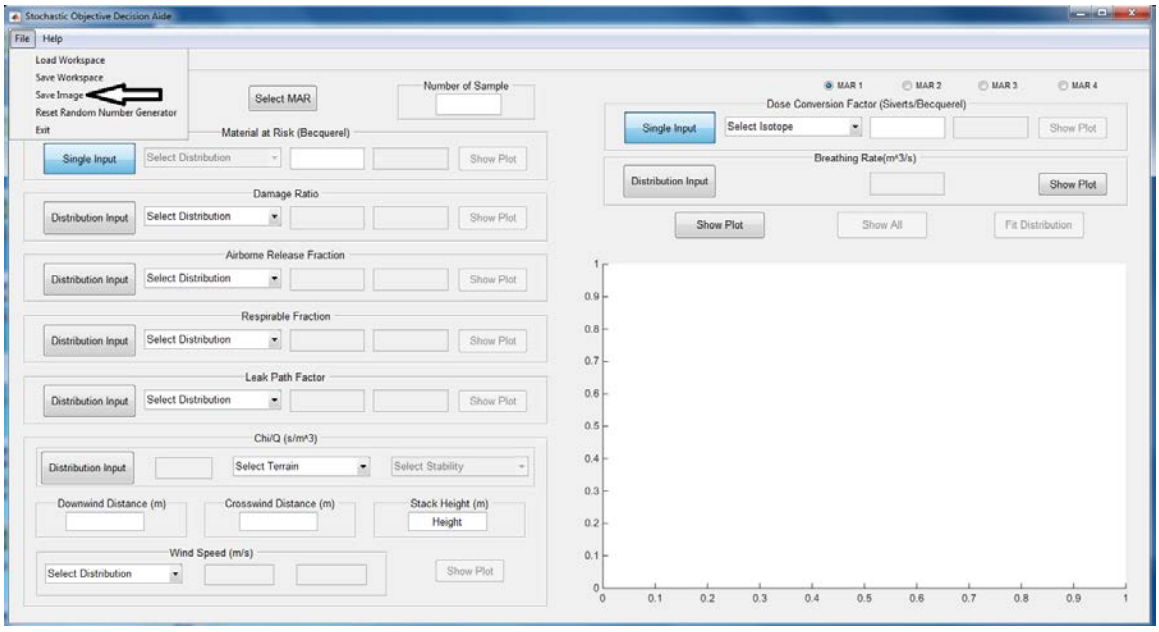


When saving the workspace, a MATLAB file will be saved. This file contains the parameters that were specified in the SODA application when the workspace was saved. An example of a saved workspace can be seen below.

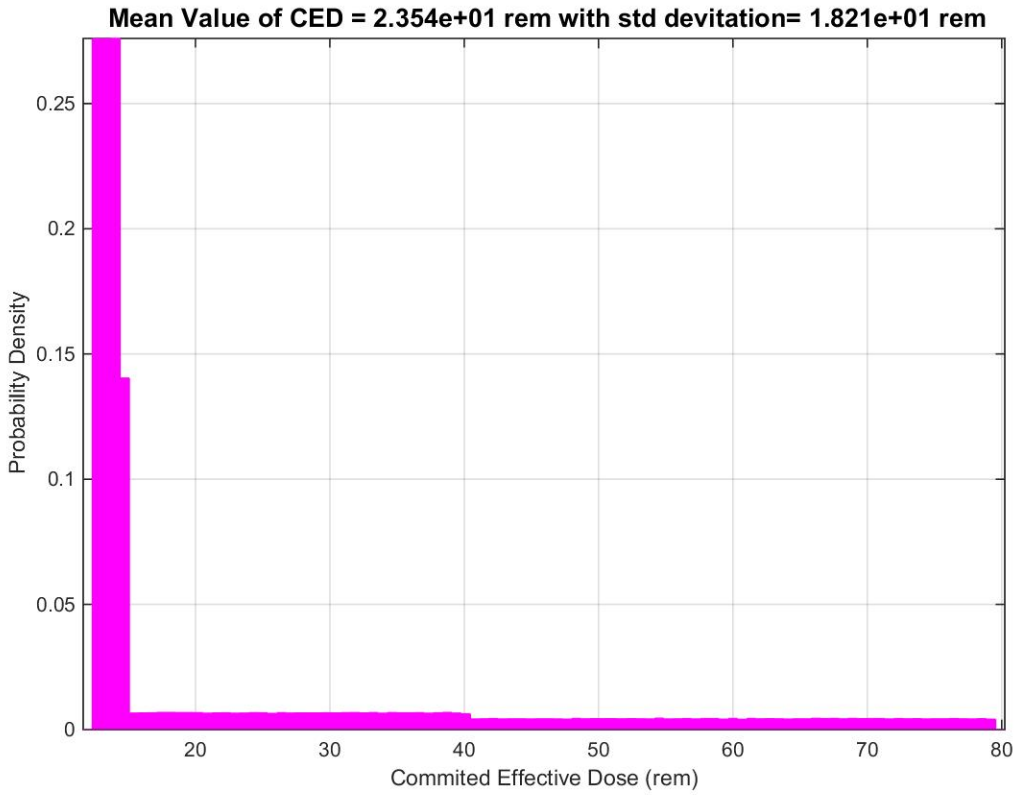


2.2.2 IMAGE SAVING

To save the image that has been generated by the SODA application in the plot space, click ‘File’ in the upper left hand corner of the program. Then select ‘Save Image’. This will allow you to save the image to any location on the machine that you are using.



This will save the plot portion of the screen as a JPEG image. An example of a saved image can be seen below.



3. ENTERING VALUES FOR PARAMETERS

3.1 DISTRIBUTION TYPES

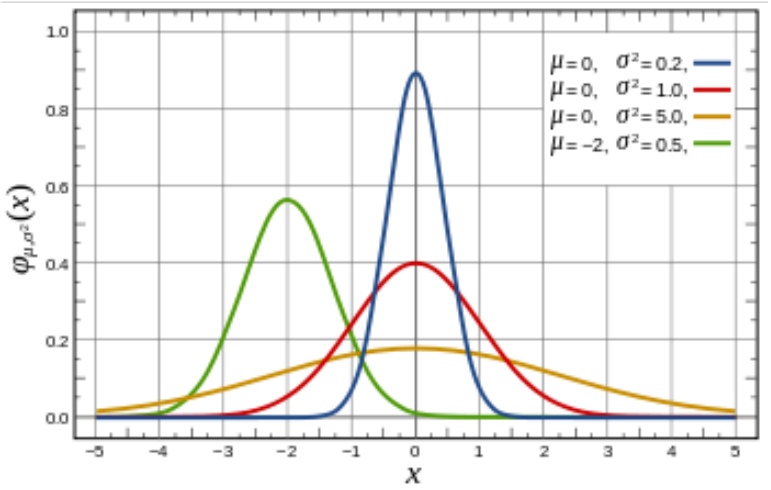
The type of distribution selected for each of the parameters within SODA will depend on the data available to the user for input into the application. The distribution options within the program include normal, beta, exponential, uniform, log normal, and user-defined distributions. The other option for a user without distribution information is a single input value.

3.1.1 NORMAL DISTRIBUTION

The normal distribution, also known as Gaussian distribution, is a common distribution in which the PDF has a bell shaped curve. The PDF equation of a normal distribution is

$$f(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where μ is the mean or expected value and σ is standard deviation. Below are examples of normal distributions using different mean and standard deviation values. In SODA, a normal distribution is specified by its mean and standard deviation.



3.1.2 BETA DISTRIBUTION

The beta distribution, plotted in Figure 2.11, is defined over the interval of 0 to 1 with two positive shape parameters, alpha and beta. The PDF equation of a beta distribution is

$$f(x \mid \alpha, \beta) = \begin{cases} \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)} & 0 < x < 1 \\ 0 & \text{Otherwise} \end{cases}$$

where B is the beta function. Since the beta distribution is defined between 0 and 1 it can be used for the damage ratio, leak path factor, respirable fraction, and airborne release fraction. In SODA, the beta distribution is specified by its alpha and beta values.

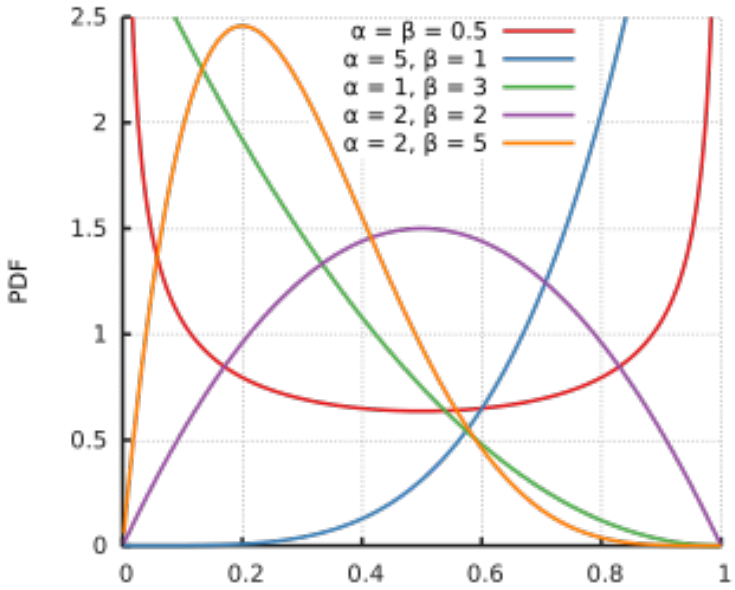


Figure 2.11: Beta Distribution.

3.1.3 UNIFORM DISTRIBUTION

The uniform distribution is a uniform, continuous distribution in which all possible values between the minimum and maximum value are equally probable. The PDF equation of a uniform distribution is

$$f(x\mid, a, b) = \begin{cases} \frac{1}{b-a} & \text{for x} \in [a,b] \\ 0 & \text{Otherwise} \end{cases}$$

where *b* is the maximum value and *a* is the minimum value. These upper and lower limits are specified by a user in SODA to define a uniform distribution. A uniform distribution plot is shown in Figure 2.12.

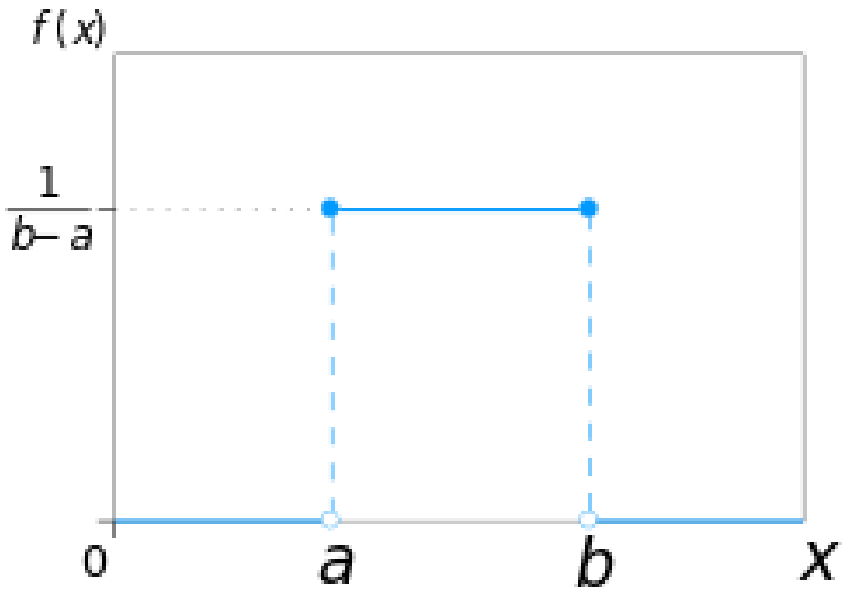


Figure 2.12: Uniform Distribution.

3.1.4 EXPONENTIAL DISTRIBUTION

The exponential distribution describes a process in which events occur continuously and independently at a constant average rate λ (see Figure 2.13). The exponential distribution PDF equation is

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x > 0 \end{cases}$$

A plot of Exponential distributions with varying lambdas is shown below. In SODA, a user specifies an exponential distribution by its mean value.

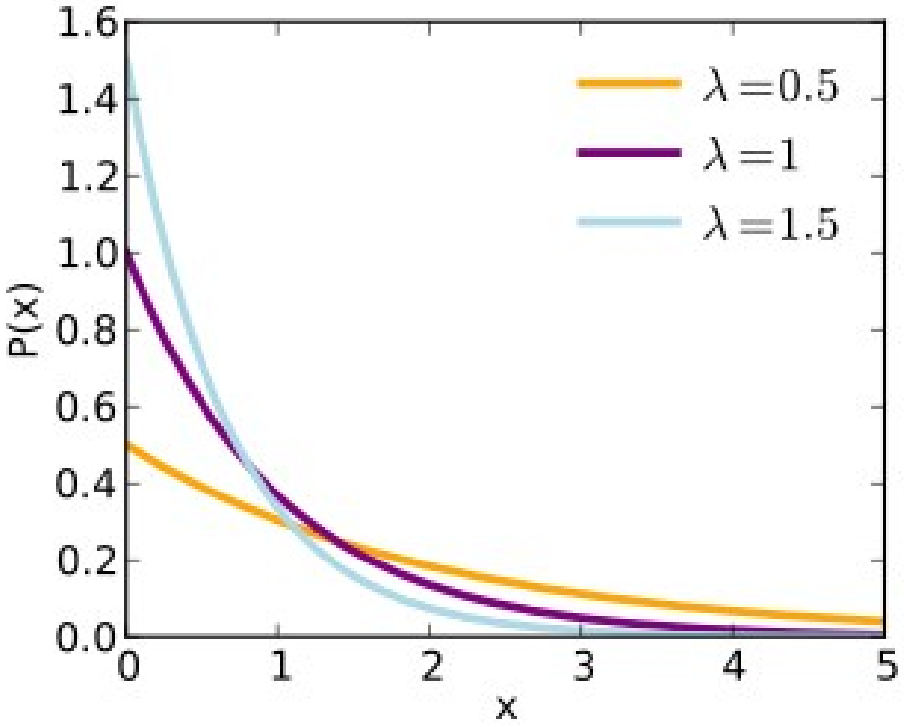


Figure 2.13: Exponential Distribution.

3.1.5 LOG-NORMAL DISTRIBUTION

The probability distribution in which the logarithm of a variable is normally distributed is log normal distribution. The PDF equation of a log normal distribution
is

$$f(x\,|,\mu,\sigma) = \begin{cases} \frac{1}{\sigma x\sqrt{2\pi}} e^{-\frac{(log(x)-\mu)^2}{2\sigma^2}} & x > 0 \\ 0 & \text{Otherwise} \end{cases}$$

A Log-Normal distribution plot is shown in Figure 2.14.

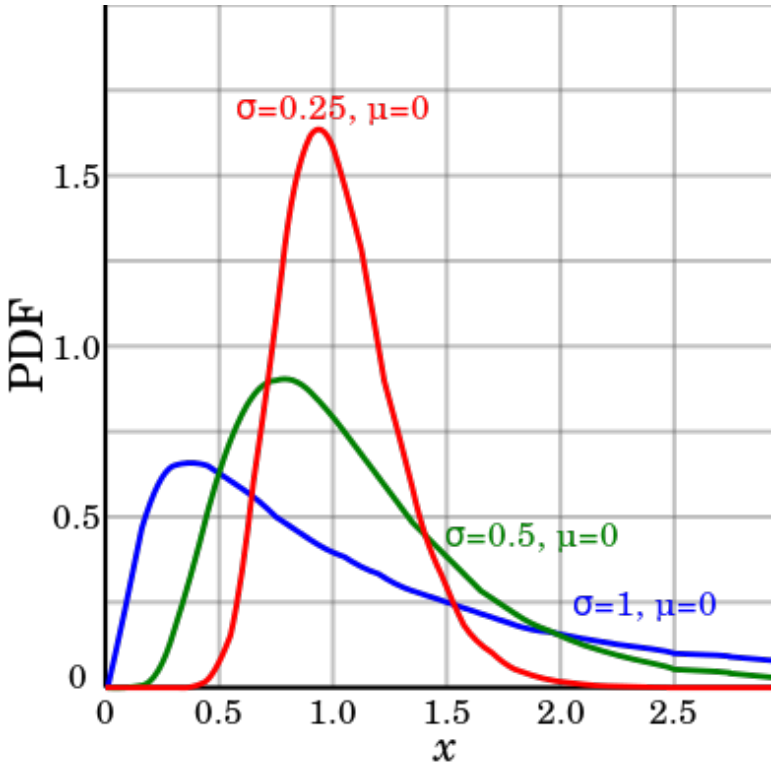


Figure 2.14: Log-Normal Distribution.

In the SODA application, the Log-Normal Distribution is defined by the mode of the distribution, also known as the most probable value, rather than the location parameter. If a different value is to be selected to locate the distribution, such as the location parameter (shown above as μ) or the median, then mathematical relationships may be used to convert to the mode. Given the usual μ=location parameter, σ =scale parameter:

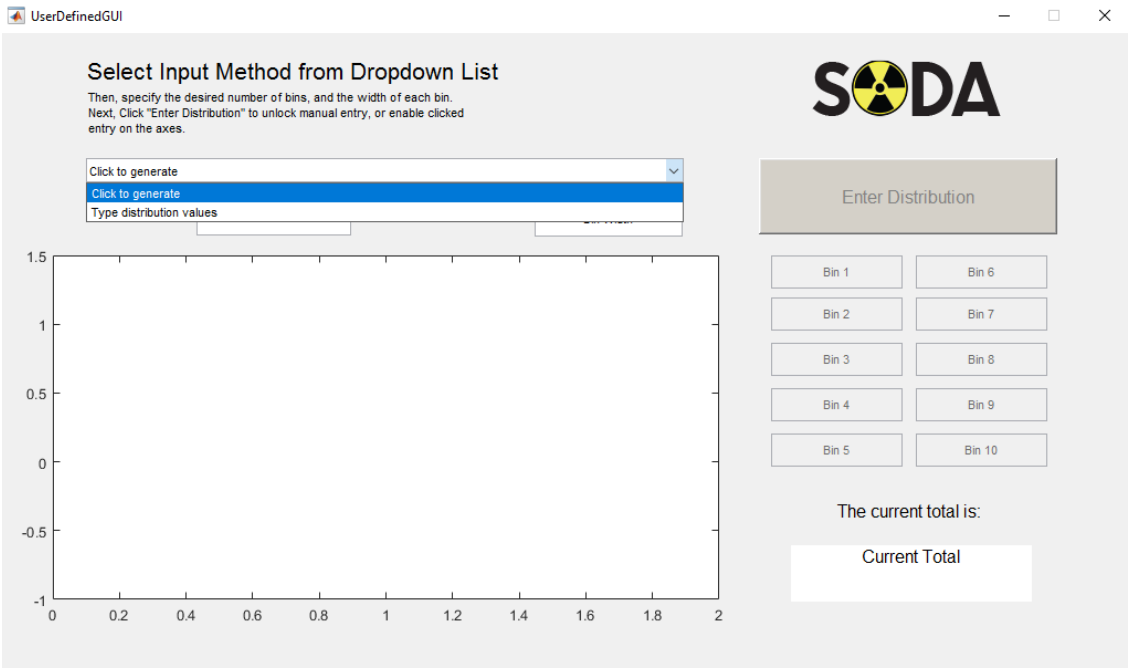
$$Mode(X) = e^{\mu-\sigma^2}$$

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

The mode was selected for specifying peak location in SODA, so that when a user specifies a known (or accepted) value to locate the distribution, that the known value is the most probable value. (For example, when a tabulated DCF is specified, it should be the most probable value when sampling for the DCF)

3.1.6 USER DEFINED DISTRIBUTION

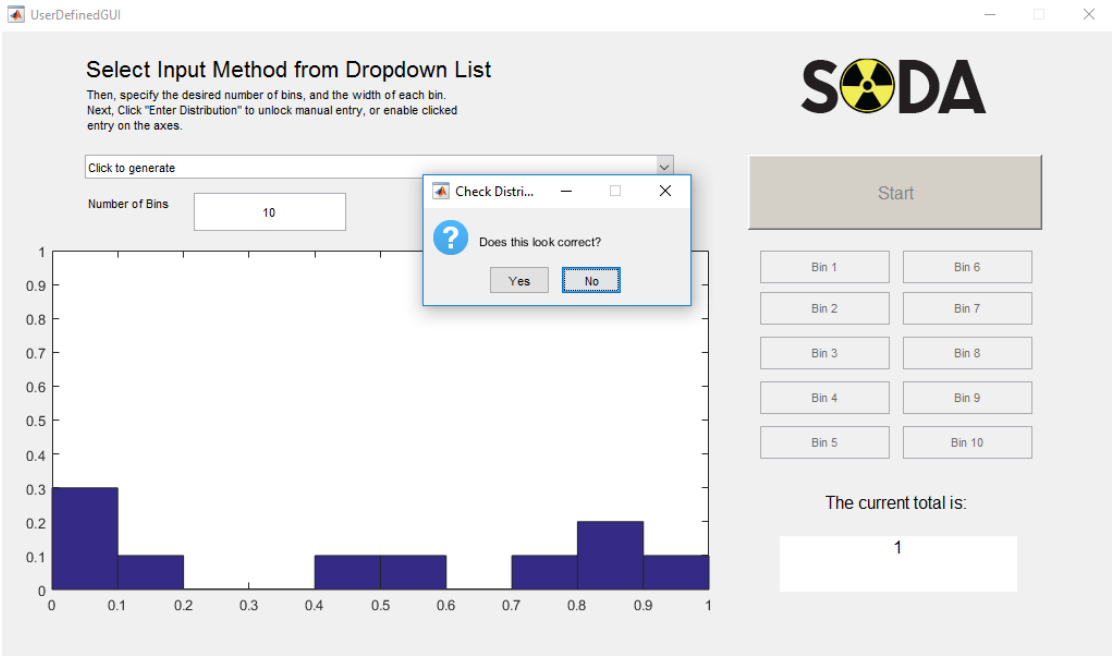
For the user defined distribution option, the mean, standard deviation, or type of distribution for the values do not need to be known. The user is able to select values and their probabilities based on known data rather than a distribution type. The two options for this type of entry are clicked entry and typed value entry. The entry method can be selected from the drop down box on the User Defined Distribution page as shown below.



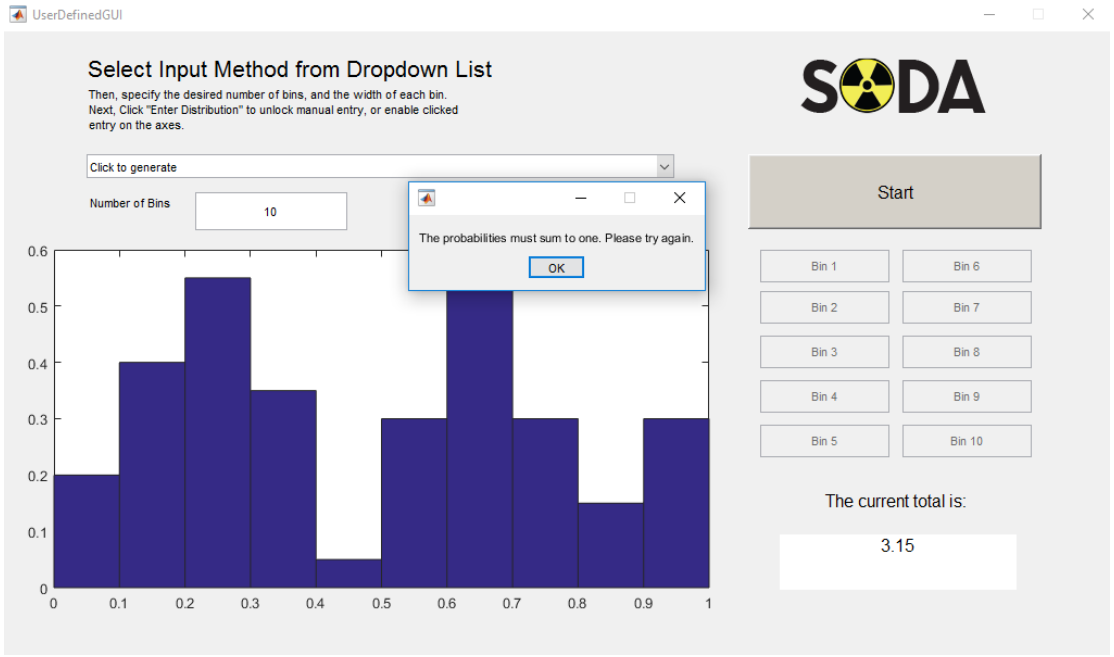
The number of bins and bin width must be specified by the user. The maximum number of bins for this entry method is 10. If a number greater than 10 is entered in the ‘Bins’ textbox, the program will automatically change it to 10.

3.1.6.1 Clicked Entry

If the entry method selected is ‘Click to generate’, then after entering the number of bins and the bin width, please click the ‘Start’ button. At this time, the program is ready for the clicked values to be entered. To do this, begin by clicking in the first bin located to the left hand side of the axes. Continue clicking until you have entered the number of values for the number of bins specified. The ‘Current Total’ box on the bottom right hand side of the window will update as the total increases. This value must add to 1. If the distribution values entered add to 1 and all of the bin values have been entered, the application will display the distribution and confirm that it is correct.



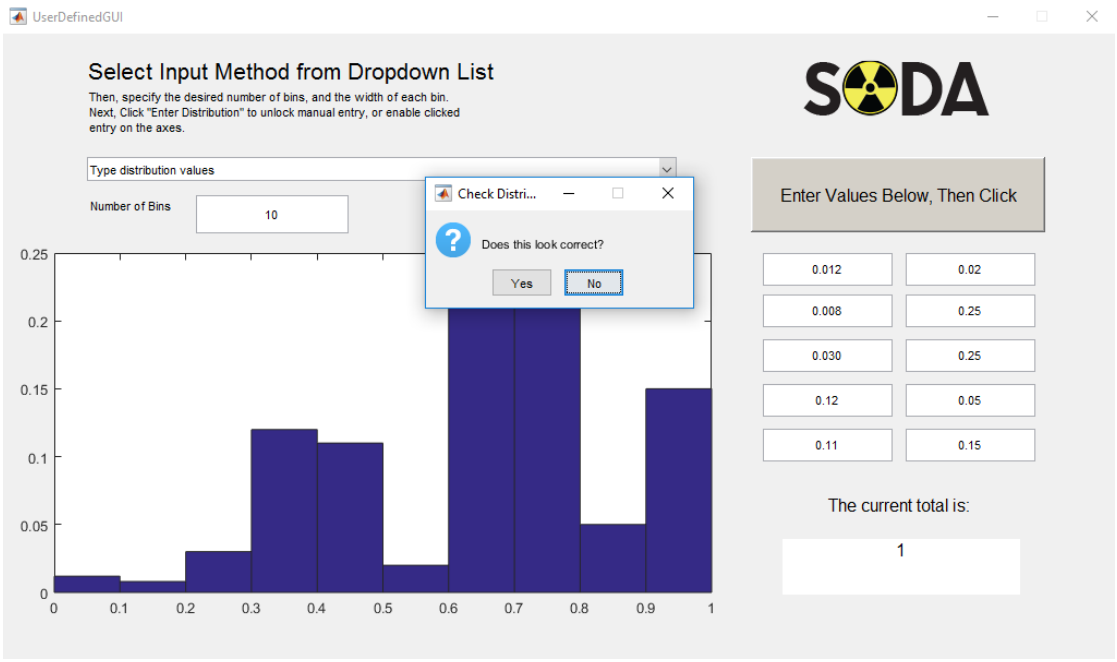
If the distribution looks correct, select 'Yes' here and the values will be passed back to SODA. If it does not look correct, select 'No' and you will be given the option to try again. If the values to not sum to 1, the application will display the following message.



You will then be given the opportunity to reenter the values.

3.1.6.2 Typed Entry

When the typed entry option is selected from the drop down menu the bin and bin width boxes still must be filled. The limit of 10 bins applies to this entry option as well and again the application will automatically correct any bin value over 10 to the maximum value of 10. Once the number of bins and the bin width has been entered, you must type the values for each probability into the text boxes on the right side of the window. When all of the values are entered, click 'Enter Values below before Clicking'. The program will create a plot of the typed distribution values and ask you if it looks correct if everything summed to 1. If it did not sum to 1, it will notify you and you will be given the opportunity to try again.



3.1.7 Values Needed for Distribution Options

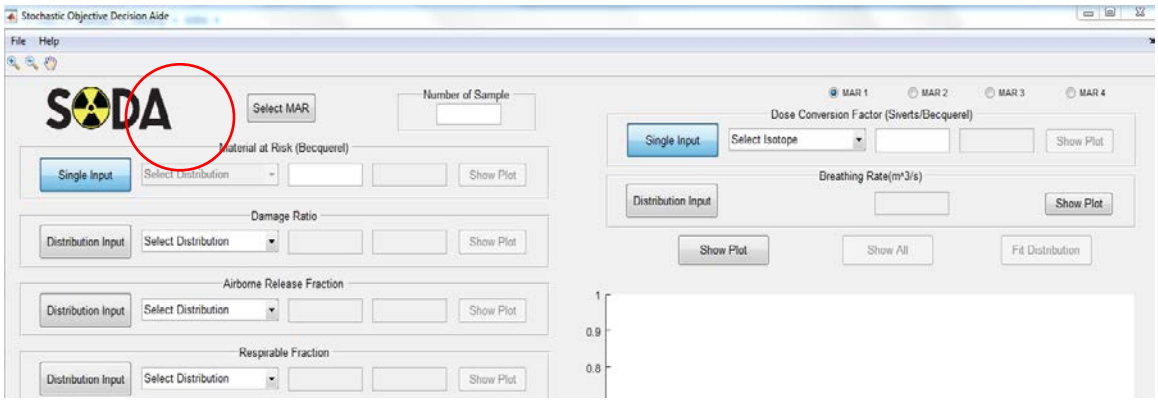
Values Needed for Distribution Option

Distribution Type	Value 1	Value 2
Normal	Mean	Standard Deviation
Beta	Alpha	Beta
Uniform	Upper Limit	Lower Limit
Exponential	Mean	N/A
Log Normal	Mode	Scale Parameter

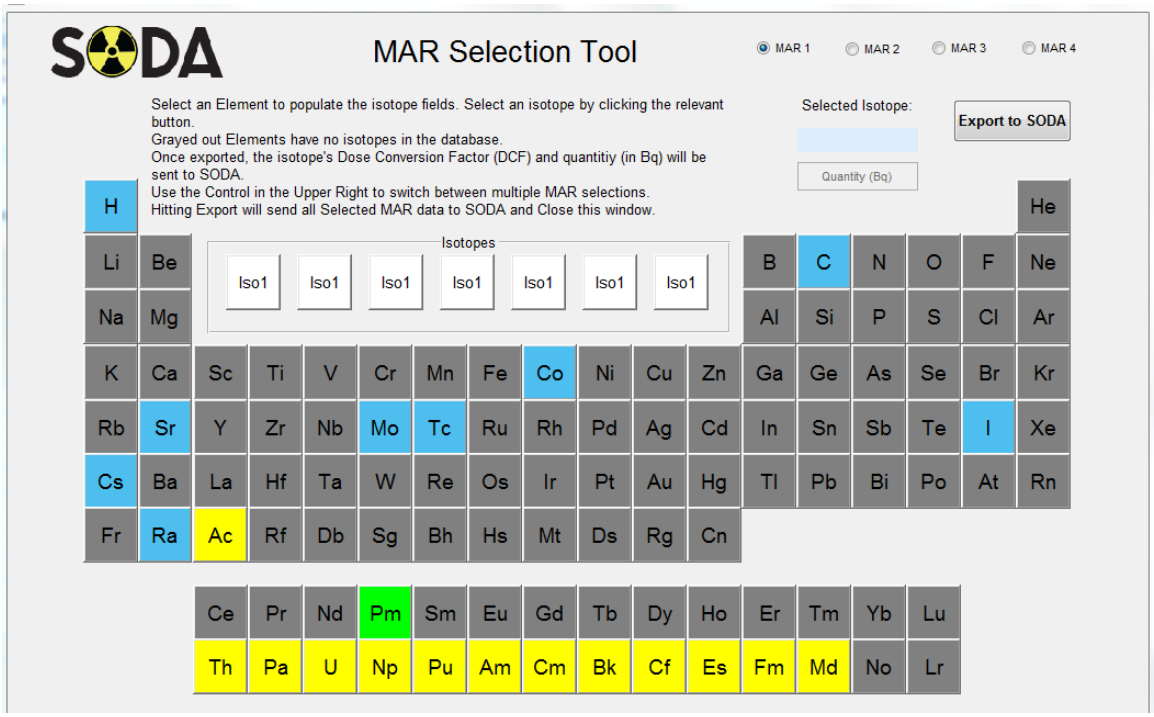
3.2 VALUES FOR PARAMETERS

3.2.1 MATERIAL AT RISK (MAR)

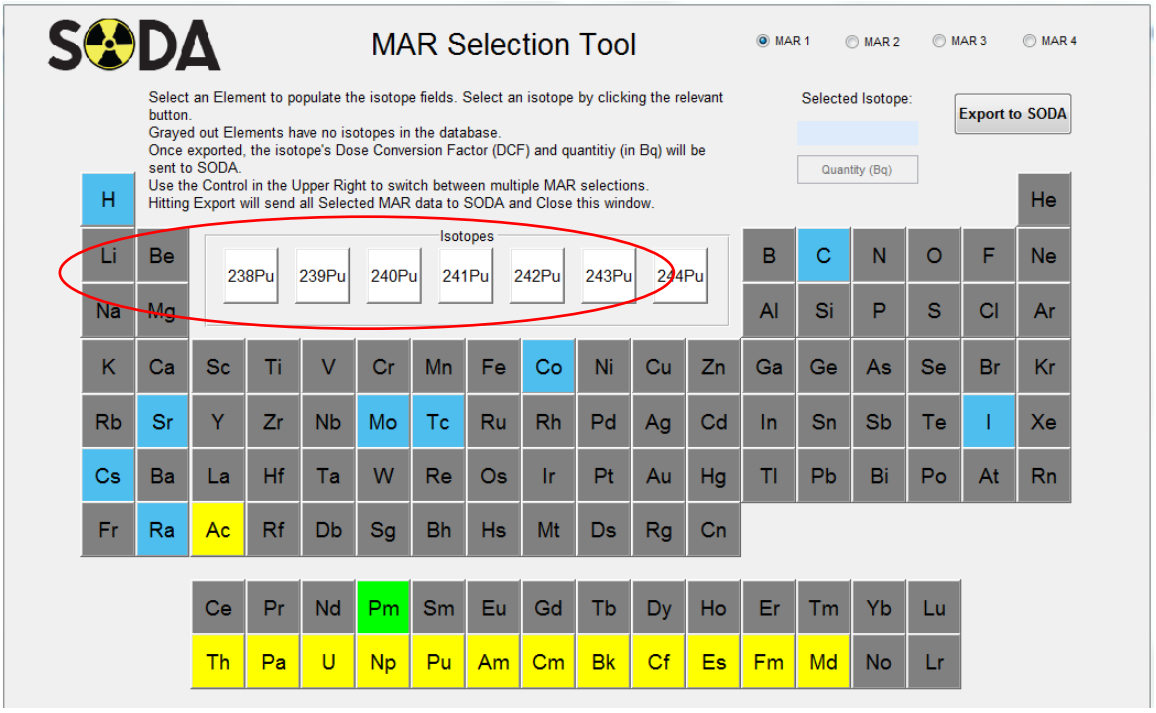
To enter the MAR, click ‘Select MAR’ at the top of the screen.



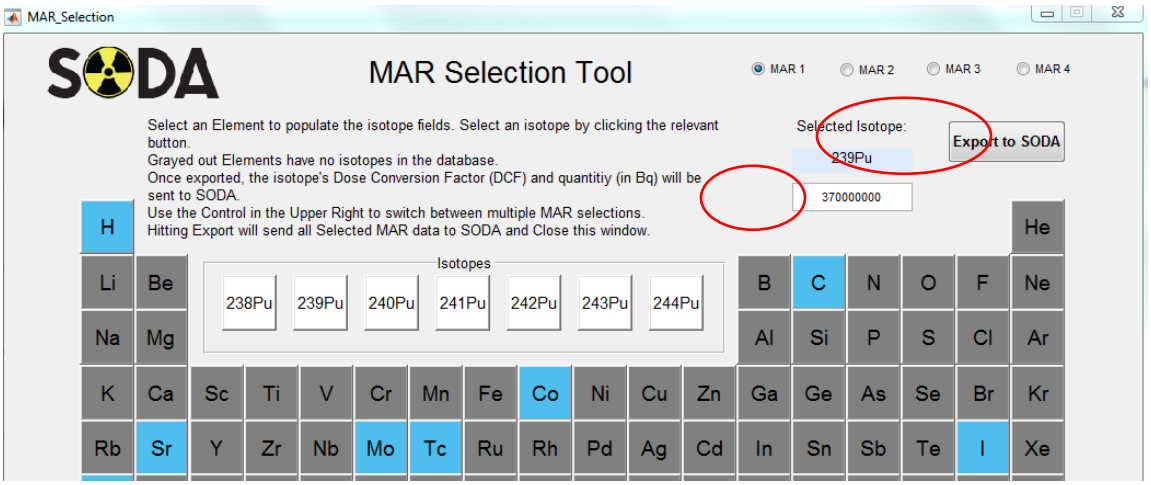
This will bring up the periodic table of elements window where you can select materials for the CED calculation.



Any elements in this table that are grayed out do not have available isotope data. The green, yellow, and blue elements are available for selection. If your desired isotope is not in the database, you can input the MAR quantity and corresponding DCF value in the SODA main screen. Many of the elements have multiple isotopes that will show up in the Isotopes section in the middle of the window when the element is selected. An example of the isotopes that show when plutonium is selected can be seen below.

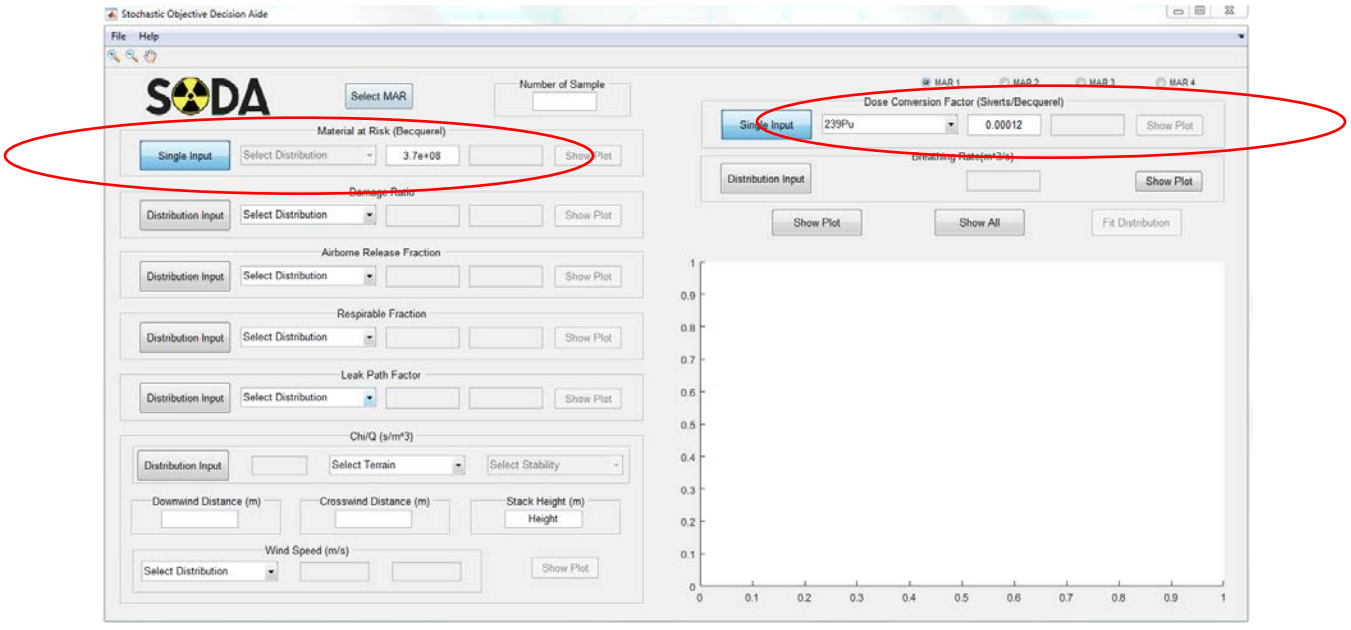


When one of the isotopes is selected it will appear in the ‘Selected Isotope’ box in the upper right hand corner of the window. At this time the quantity of the material must be specified. Type the quantity in Becquerels. Once this has been done, you may select another MAR or you can click ‘Export to SODA’ to continue with the MAR that has been specified. In the example below, the quantity has been entered. At this point if only a single MAR is needed, then the user would click the ‘Export to SODA’ button. If multiple MARs need to be entered, the user would first perform these same steps after selecting another MAR.



In the multiple MAR case, repeat this process for MAR 2 and then click the MAR 3 radio button. Finally, after repeating the process for MAR 3, the MAR 4 radio button can be selected and the selection process can be completed for the fourth MAR. Any number of MAR's may be selected; it is not necessary to enter information for all of the available four. When all of the MAR information necessary has been entered, click the 'Export to SODA' button to send the data back to the SODA application.

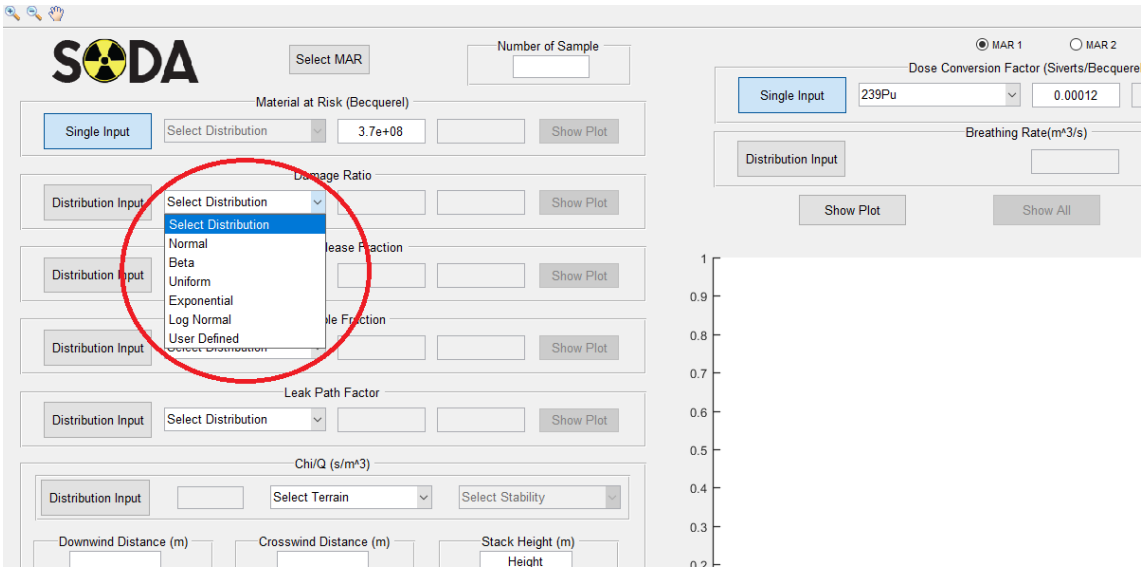
In the main SODA window, you will see the Material at Risk (Becquerel) section and the Dose Conversion Factor (Sieverts/Becquerel) section have been populated. These are based on the isotope and the quantity provided in the MAR Selection Tool.



3.2.2 Damage Ratio, Airborne Release Fraction, Respirable Fraction, Leak Path Factor

To enter the values for DR, ARF, RF, and LPF, first select ‘Single Input’ or ‘Distribution Input’ by clicking the button next to the parameter you are working on. If ‘Single Input’ is selected, you may enter the single point value in the textbox that has been enabled. This will give you less information about the CED, but may be appropriate if the uncertainty for the parameter being entered is not understood. DR and LPF selections are universal across MAR selections, while ARF and RF are specific to the selected MAR. An example of this type of entry can be seen for the DR below.

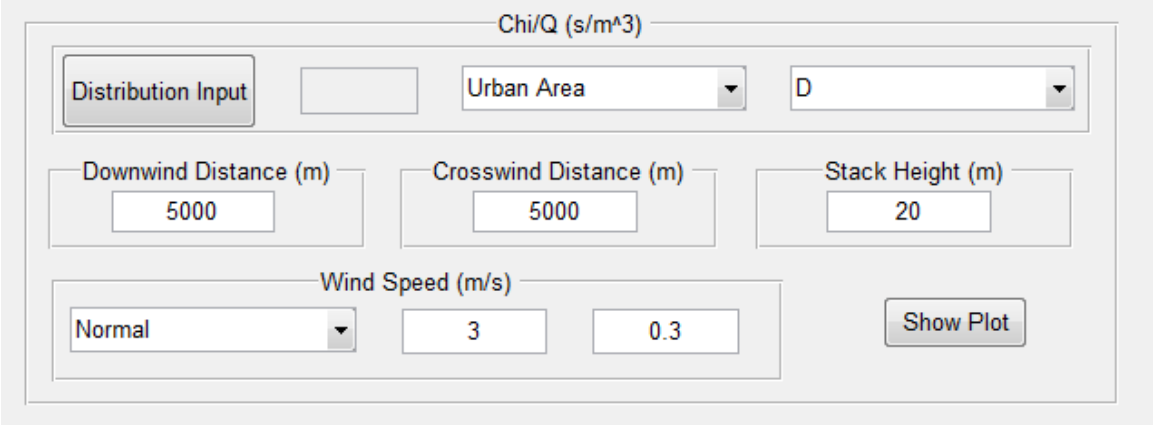
If a distribution of values is known for the parameter, the ‘Distribution Input’ option should be selected. When this option is selected, the drop down menu for the types of available distribution inputs is displayed. This menu can be seen in the figure below.



From this drop down list, the distribution type is selected. With the distribution type selected, the textboxes for the parameter autofill with the type of information that you will need to provide. For example, if Normal distribution is selected from the drop down menu, the textboxes will indicate that you must enter the mean and standard deviation for the distribution. If Beta distribution is selected, the textboxes will show text indicating that you must enter alpha and beta values. If Uniform distribution is selected, you will need to enter the lower and upper limits. If Exponential distribution is selected, you will see text in one box indicating that you need to enter just the mean.

3.2.3. ENTERING VALUES FOR X/Q

When entering values for the plume dispersion parameter, a single value can be selected or a distribution option is available. If you choose to use the distribution option, you must know the type of terrain, the stability class, the downwind and crosswind distances, the stack height, and the wind speed. The wind speed must be entered as a distribution. The terrain types available are rural and urban terrain. Rural terrain is terrain that has less than 50% developed area, and urban terrain is terrain that has greater than 50% developed area. The stability classes available range from A to F with A being the least stable of the classes, with a rating of 'Very Unstable' and F being the most stable with a rating of 'Stable'. In the urban terrain option, stability classes A and B are grouped together and classes E and F are grouped together. The stability classes are used to determine the dispersion from the centerline of the plume, and the calculations are the same for A and B stability classes and E and F stability classes when using urban terrain. In rural terrain, all six stability classes are separate. An example of the χ/Q section of SODA filled out can be seen below.



When entering the values for plume dispersion, if you are unsure of which distribution to use for wind speed, it is more conservative to use normal distribution than uniform distribution when using rural terrain. Both distributions give values that are virtually the same when using urban terrain.
If you are unsure of the terrain type in the scenario you are interested in, the rural terrain option will give a higher CED estimation, therefore giving more conservative information than urban terrain. A less stable atmospheric stability class will provide a more conservative distribution for the CED also. The χ/Q parameter suggestions are as follows:

- Terrain
  - For downwind distances less than 1000m the urban terrain option will give the most conservative dose
  - For downwind distances greater than 1000m the rural option will the most conservative dose when used with stability classes A, B, or C
  - For downwind distances greater than 1000m the urban option will give the most conservative dose when used with stability classes D, E, or F
- Stability Class
  - For downwind distances less than 1000m, the stability class should be known with certainty
  - The use of stability class A at low downwind distances will give the most conservative estimate if the class is unknown
- Crosswind Distance
  - The crosswind distance should be known with certainty when the terrain is rural and the atmospheric stability class used is D, E, or F
  - The crosswind distance can be estimated when the stability class is A, B, or C
  - The crosswind distance can be estimated when the terrain is urban
- Stack Height
  - If downwind distance is less than 1000m, the exact effective stack height should be known
  - If downwind distance is greater than 1000m, the stack height can be estimated

- Wind speed and distribution
  - The wind speed and the distribution used to represent the variation in wind speeds does not have a significant effect on the overall CED
  - If the exact wind speed is unknown, a lower speed will give a more conservative estimate

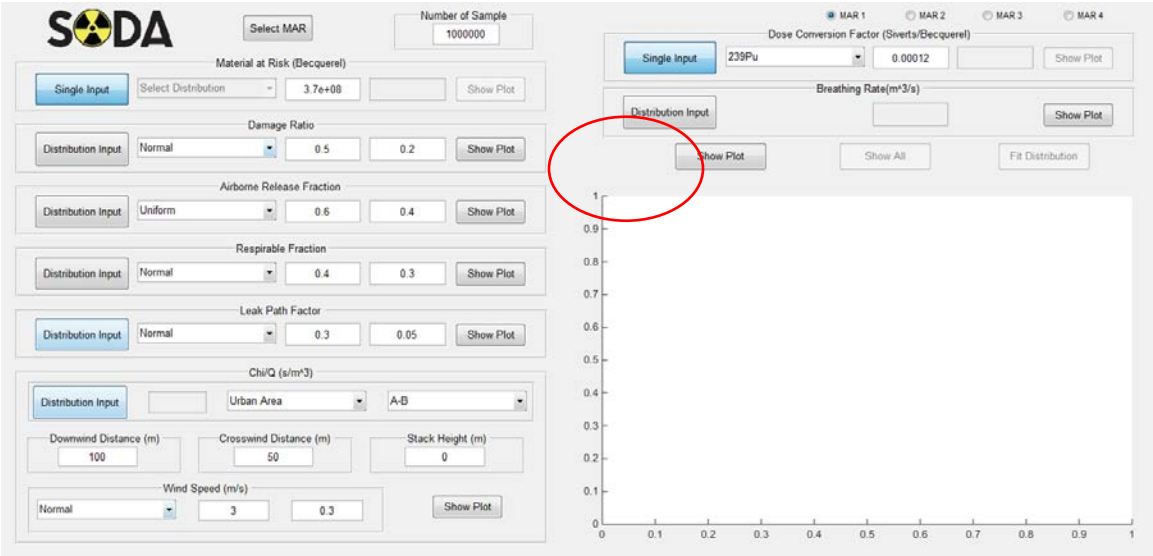
3.2.4. Entering Values for Number of Sample

The value entered in the number of samples textbox will be used to determine how many samples of each parameter to take and how many CED calculations to complete. The greater the number in the box, the more accurate the distribution generated will be. However, if a number is entered that is too large it can cause the program to run very slowly or even freeze and it then defeats the purpose of having a quick and easy tool. A warning will show if you select a sample number larger than 100 million (1E8). A good number to use for this parameter is between 1 and 10 million (1E6 to 1E7). For more details on sample count and ram usage, look in the Limitations section of this document.

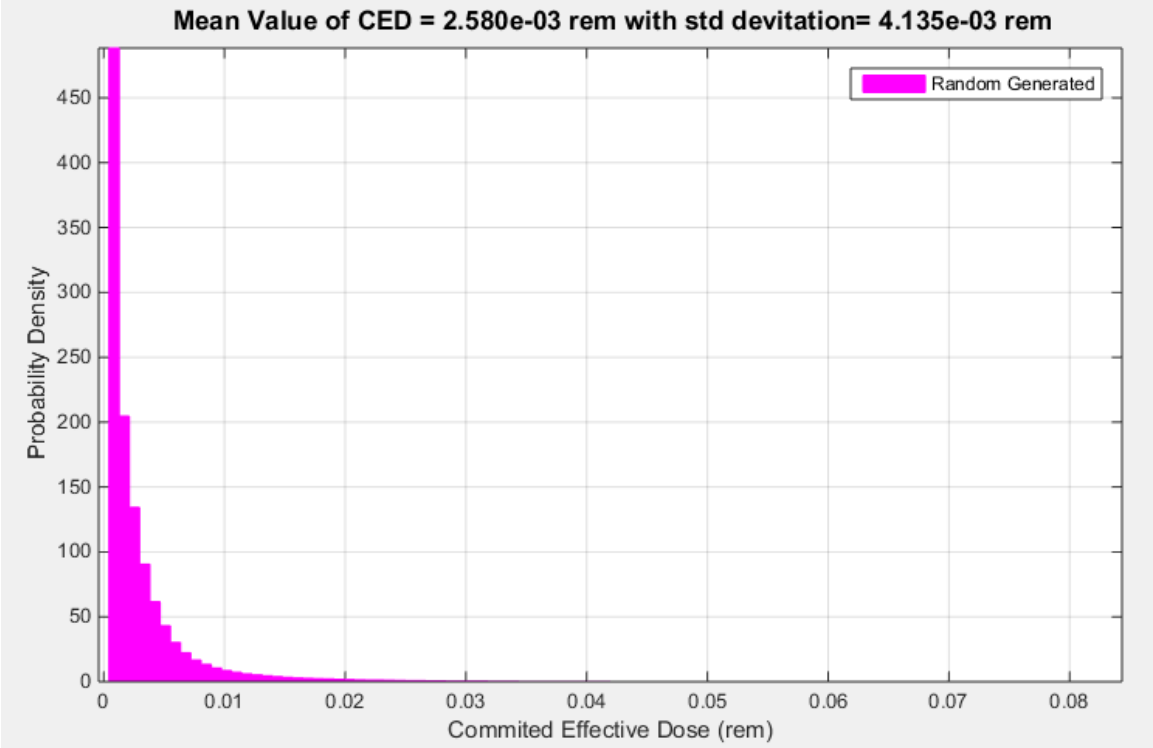
4. PLOTS

4.1. HOW TO PLOT

Once values have been entered for all variables, the CED distribution can be calculated and plotted. In order to plot the CED distribution, click the ‘Show Plot’ button above the axis.



Once the ‘Show Plot’ button has been clicked, please wait without clicking anything else for the distribution plot to be generated. This can take a moment depending on your machine. Once the plotting is complete you will see a distribution like the one below.



The SODA application will provide the mean value and standard deviation in rem in addition to the plotted distribution. The mean and standard deviation give a quick idea of where the dose value lies, while the plotted distribution gives more detail. As can be seen from the figure above, the mean value for this example is $2.580*10^{-3}$ with a standard deviation of $4.135*10^{-3}$.

In the event that multiple MARs have been selected, and a calculation incorporating all of these selections is desired, Show All should be selected. Note that the information for each MAR entry must be complete. An incomplete entry will not be used in the calculation of the CED, and a message will be directed at the user to inform them of this. The show all button becomes usable once a second MAR is selected with the radio buttons on the SODA main GUI. After clicking Show All, the resulting CED is for all of the selected MARs.

5. LIMITATIONS

SODA does not account for wet or dry deposition. When using Show All to perform a multiple MAR CED calculation, the model assumes that the various MARs are well mixed and are inhaled in the proportions of their original quantity. This may not always be the case, depending on the material properties of each MAR, but is generally a good assumption. This tool is meant to be used as a guideline for decision making, but will not supply a single yes or no answer for decision makers. This tool may be used to gain an understanding of possible dose values from a hypothetical accident scenario and to make a decision regarding the necessity of safety structures in that specific scenario.

SODA uses Monte Carlo methods to perform its calculations. As is typical when using MC methods for calculations, a higher sample count will take longer to compute, but achieve a better answer. Due to the nature of this calculation, very high sample counts will result in a memory usage penalty. 1E8 samples, if all available distribution inputs are selected, can result in a usage in excess of 10GB of memory. Typically, 1E6 samples are sufficient for a general answer, while 5E6 to 1E7 can be used for a final answer. Higher counts can of course be employed on powerful machines, containing larger memory sizes. The reason for the sizable memory usage is how SODA computes a CED from distribution inputs. Each distribution will be sampled as many times as specified in the same count. Each of these results take up a sizable amount of memory when a large sample count is specified. The distribution for each input must be saved until they are multiplied to get a ST or CED result. It is the point just before calculation of the CED result where the memory usage reaches its peak. In the case of a multi-MAR calculation, the CED result for each individual material is saved until the final multi-MAR CED result is computed. As a result, the peak usage in a multi-MAR problem will be higher than it would be in a single MAR calculation.

The MAR Database, which is a file included in the SODA distribution, contains the isotope data that is accessed in the MAR Selection GUI. If the information contained in this file becomes outdated, or an application user wishes to include more isotopes in the file, there are some limitations which must be known. The format of the file must be preserved, which means that only 7 isotopes per element are accommodated, with 7 columns for isotope A number, and 7 columns for the corresponding DCF values. The actinide portion of the original file will make for a good example. The CSV file may be viewed in

excel, but changes must be saved in csv format, not xls format. The program is setup to read a csv file, not an xls file.

APPENDIX B:

MATLAB 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','EdgeColo
r','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','EdgeColo
r','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','EdgeColo
r','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','EdgeColo
r','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','EdgeColor','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},'Select
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_dist);
    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.windspeed_popup_di
st);
    set(handles.cq_text1,'Enable','off','String','');
    set(handles.height_text,'String',userinput.height_text);
    set(handles.distance_text1,'Enable','on','String',userinput.distance_text1);
    set(handles.distance_text2,'Enable','on','String',userinput.distance_text2);
    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},'Select
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},'Select
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 C:

Damage Ratio Experiment Data

3 m drop (pint)	Container Break	
	Yes	No
1		1
2		1
3		1
4		1
5		1
6		1
7		1
8	1	
9		1
10		1
11		1
12		1
13		1
14		1
15		1
16		1
17		1
18		1
19		1
20	1	
21		1
22		1
23		1
24		1
25		1
26		1
27	1	
28		1
29		1
30		1
31		1
32		1
33	1	
34		1
35		1
36		1
37		1
38		1
39		1
40		1
41		1

3 m drop (pint)	Container Break	
	Yes	No
42		1
43		1
44		1
45	1	
46		1
47	1	
48		1
49		1
50		1
51		1
52		1
53		1
54		1
55		1
56		1
57		1
58		1
59		1
60		1
61		1
62		1
63	1	
64		1
65		1
66		1
67		1
68		1
69		1
70		1
71		1
72		1
73		1
74		1
75		1
76		1
77		1
78		1
79		1
80		1
81		1
82		1

3 m drop (pint)	Container Break	
	Yes	No
83		1
84		1
85		1
86		1
87	1	
88		1
89	1	
90		1
91		1
92		1
93		1
94		1
95		1
96		1
97		1
98		1
99		1
100		1
Total	9	91

3 m drop (quart)	Container Break	
	Yes	No
1		1
2		1
3		1
4		1
5		1
6	1	
7		1
8		1
9		1
10		1
11		1
12		1
13		1
14	1	
15		1
16		1
17		1
18		1
19		1
20		1
21		1
22	1	
23		1
24		1
25		1
26	1	
27		1
28	1	
29		1
30		1
31		1
32	1	
33	1	
34		1
35	1	
36	1	
37		1
38	1	
39		1
40		1
41		1

42	1	
43		1
44		1
45		1
46		1
47		1
48		1
49		1
50		1
51		1
52		1
53		1
54		1
55		1
56		1
57		1
58		1
59	1	
60		1
61	1	
62		1
63		1
64		1
65	1	
66		1
67		1
68		1
69		1
70		1
71		1
72		1
73		1
74		1
75	1	
76	1	
77		1
78	1	
79		1
80		1
81	1	
82	1	
83		1
84	1	
85		1

86		1
87		1
88	1	
89	1	
90	1	
91		1
92	1	
93		1
94		1
95	1	
96		1
97	1	
98	1	
99		1
100		1
Total	27	73

1 m drop (pint)	Container Break	
	Yes	No
1		1
2		1
3		1
4		1
5		1
6		1
7		1
8		1
9		1
10		1
11		1
12		1
13		1
14		1
15		1
16		1
17		1
18		1
19		1
20		1
21		1
22		1
23		1
24		1
25		1
26		1
27		1
28		1
29		1
30		1
31		1
32		1
33		1
34		1
35		1
36		1
37		1
38		1
39		1
40		1
41	1	

1 m drop (pint)	Container Break	
	Yes	No
42	1	
43	1	
44		1
45	1	
46		1
47	1	
48	1	
49	1	
50		1
51		1
52		1
53	1	
54		1
55		1
56		1
57		1
58		1
59		1
60	1	
61		1
62		1
63		1
64	1	
65		1
66		1
67		1
68	1	
69		1
70	1	
71		1
72		1
73		1
74	1	
75	1	
76	1	
77		1
78	1	
79		1
80	1	
81	1	
82		1

1 m drop (pint)	Container Break	
	Yes	No
83	1	
84		1
85		1
86	1	
87		1
88	1	
89		1
90		1
91	1	
92	1	
93		1
94	1	
95		1
96		1
97		1
98		1
99	1	
100	1	
Total	26	74