INL/EXT-15-36669
Revision 0

NSR&D Program Fiscal Year 2014
Funded Research

Stochastic Modeling of Radioactive Material Releases

Idaho National Laboratory - Nuclear
Energy & Idaho State University

September 2015

Idaho National Laboratory

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

NSR&D Program Fiscal Year 2014 Funded Research

Stochastic Modeling of Radioactive Material Releases

Revision 0
September 2015

Idaho National Laboratory - Nuclear Energy & 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 operated 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 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 (SSCs) 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 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. 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.

SODA provides improved risk understanding leading to better informed decision making associated with establishing nuclear facility 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. The work was funded through a grant from the DOE Nuclear Safety Research and Development Program.

Contents

1. INTRODUCTION 1
2. BACKGROUND 1
3. SODA 4
3.1. Coding Framework 8
3.2. GUI 9
3.3. Monte Carlo Method 12
3.4. Distribution Mathematics 13
3.5. Bayesian Information Criterion 17
4. Comparison with RSAC 18
5. Conclusion 19
Appendix A: SODA Help File Information 21
APPENDIX B: Installation Instructions 39
APPENDIX C: MATLAB Code 48
APPENDIX D: Damage Ratio Experiment Data 122

Figures

Figure 1. Plume dispersion. 4
Figure 2. SODA input options. 5
Figure 3. χ/Q values from random sampling. 6
Figure 4. Adult male BR. 7
Figure 5. SODA GUI screen. 8
Figure 6. MATLAB GUI quick start. 9
Figure 7. Default GUI figure window in MATLAB. 10
Figure 8. SODA About window. 11
Figure 9. SODA main window. 12
Figure 10. Normal distributions. 13
Figure 11. Beta distribution. 14
Figure 12. Uniform distribution. 14
Figure 13. Exponential distribution. 15
Figure 14. Breathing rate distribution. 16
Figure 15. Example grapnhical representation of BIC fit to a randomly generated dose distribution..... 18

Tables

Table 1. Breathing rate data. 15
Table 2. Verification Case Input Data. 19
Table 3. Verification Results Comparison. 19

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
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 operated 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 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 MELCOR Accident Analysis Code System (MACCS) or Radiological Safety Analysis Computer (RSAC); rather it is viewed as an easy-to-use supplemental tool to help improve risk understanding and support better informed decisions. The work was funded through a grant from the DOE Nuclear Safety Research and Development (NSRD) Program.

2. BACKGROUND

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, 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 members of the public are frequently overstated, leading to excessively conservative MAR limits and potentially 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 waste containers play a significant role in decreasing the uncertainties. With 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, does not provide a particularly meaningful result.

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 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. Typically, a higher χ / Q will result in a higher potential dose to a population.

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 International Commission on Radiological Protection (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. For example, 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.

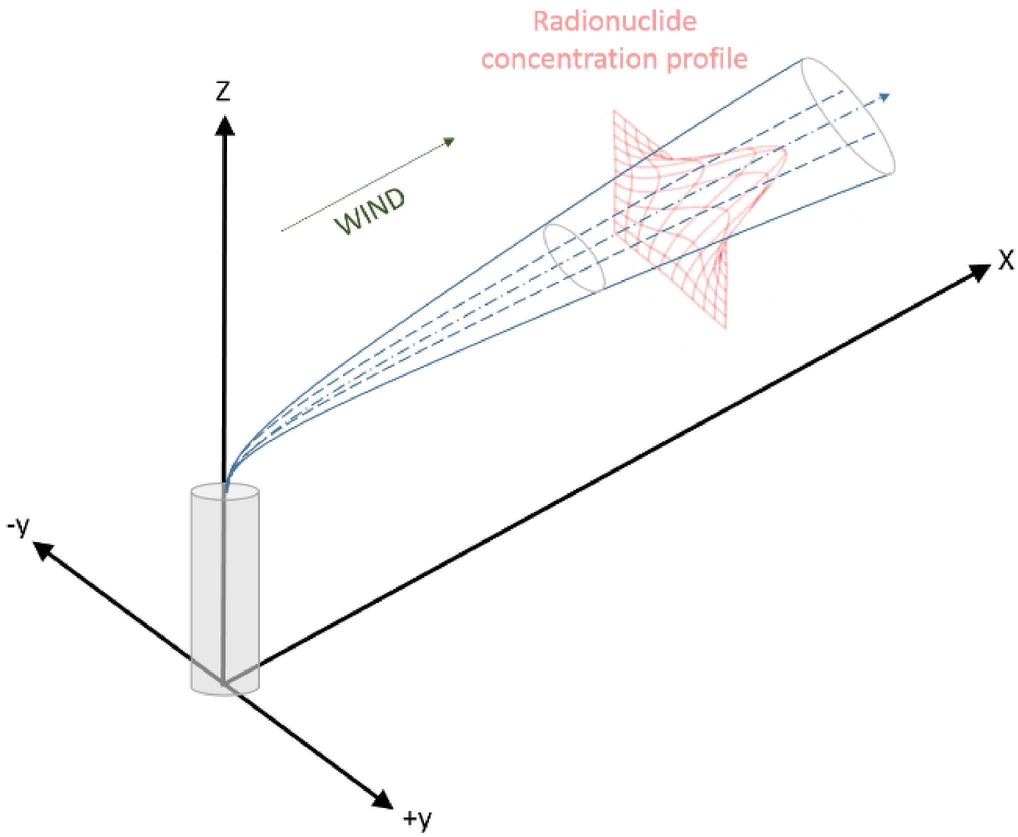


Figure 1. Plume dispersion.

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 software application is not intended to replace or even compete with traditional radioactive material release modeling codes such as the MACCS or 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 as the 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 one million random samples.

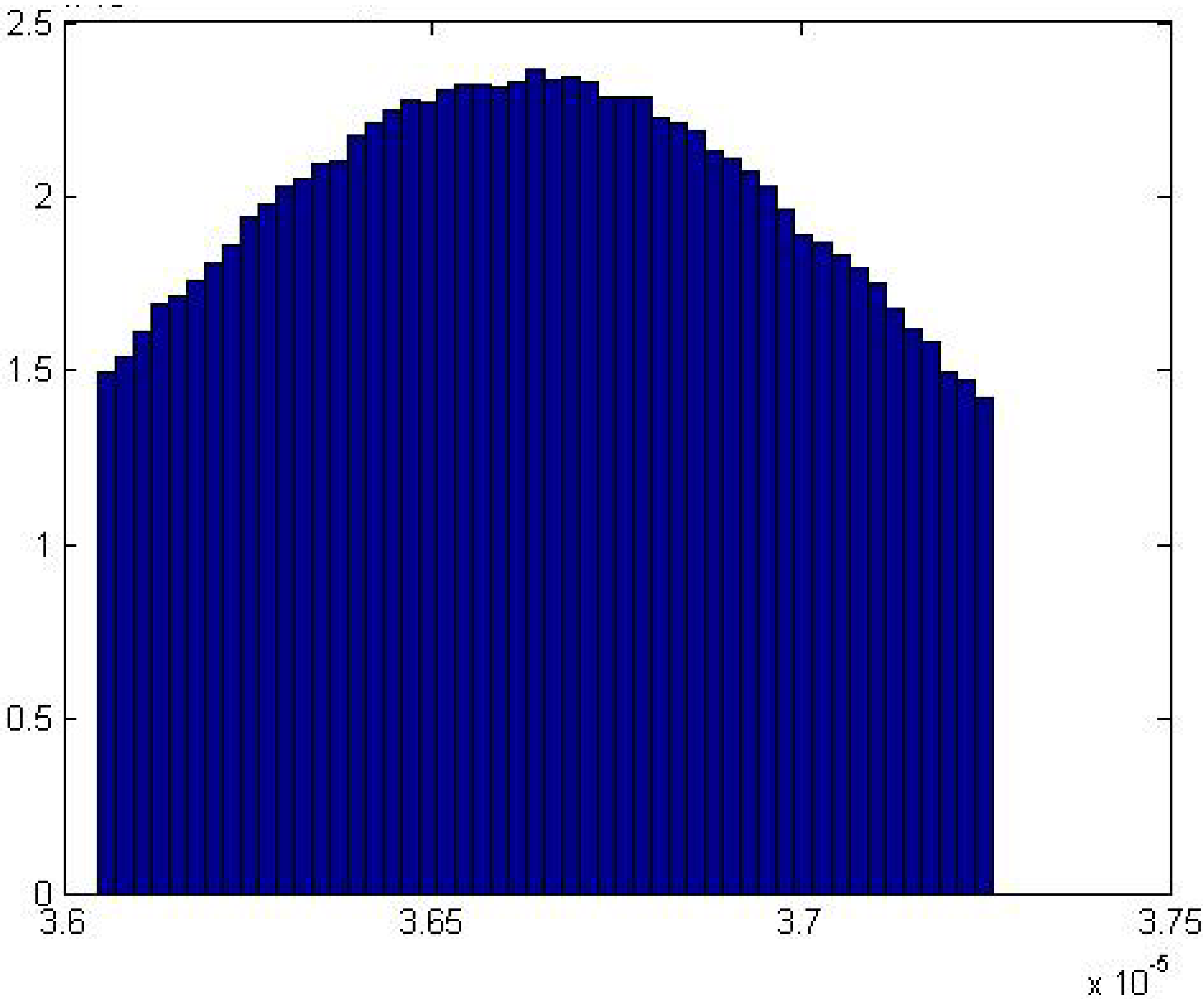


Figure 3. χ/Q values from random sampling.

The resulting dose consequence distribution is automatically analyzed to determine its similarity to well know distributions. If the resulting distribution is sufficiently similar to a well know distribution, the application provides information about the distributions such as the mean value and the probability that the dose will exceed a particular dose value.

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 BR distribution for an adult male over a hypothetical 24 hour period with BRs for sleep, sedentary activity, light physical activity, and heavy physical activity (ICRP-89, “Basic Anatomical and Physiological Data for Use in Radiological Protection Reference Values”). The traditional approach suggested by DOE-STD-3009-2014, “Preparation of Nonreactor Nuclear Facility Documented Safety Analysis,” is to select a value of 3.3×10^{-2} m^3/hr, corresponding to light activity for an adult male. However, it is clear that there are periods where the BR 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.

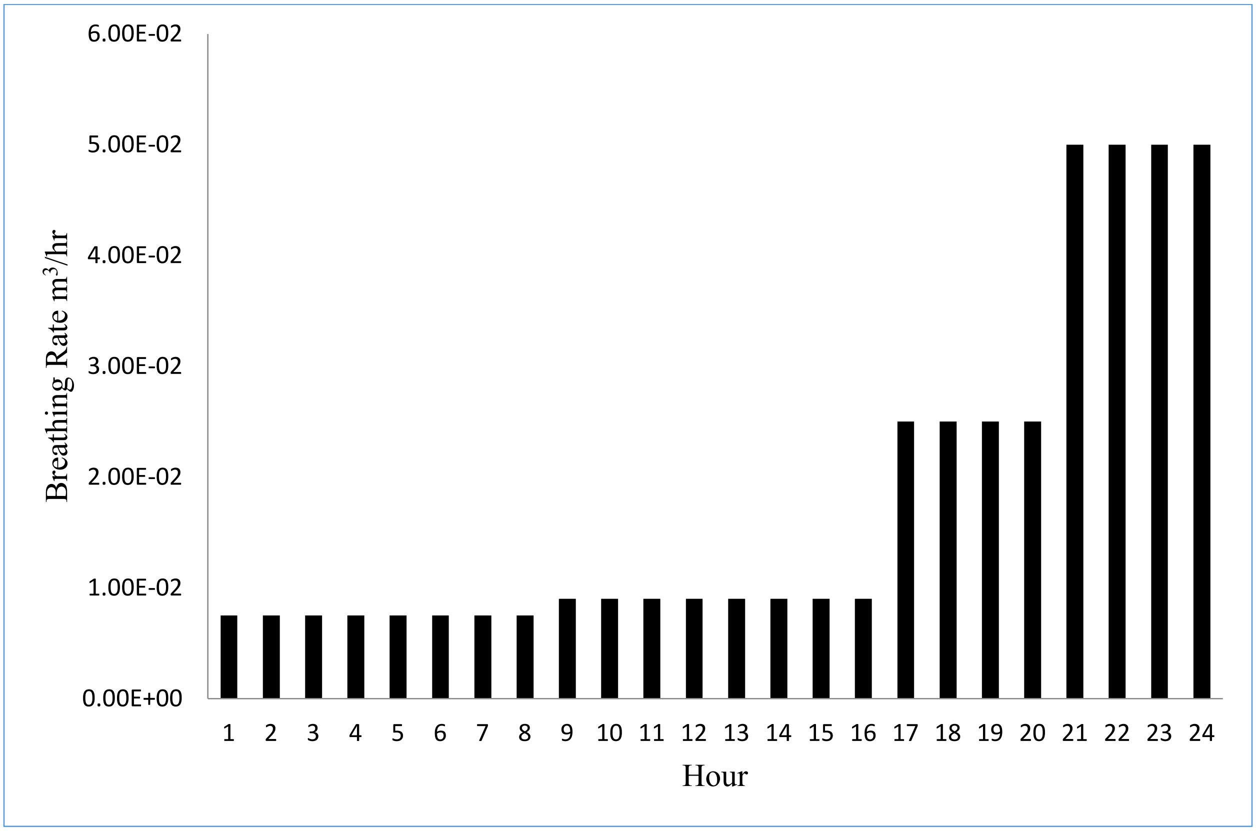


Figure 4. Adult male BR.

The SODA GUI interface screen 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. The code allows the user to save the input selections as well as export the resulting dose distribution plot.

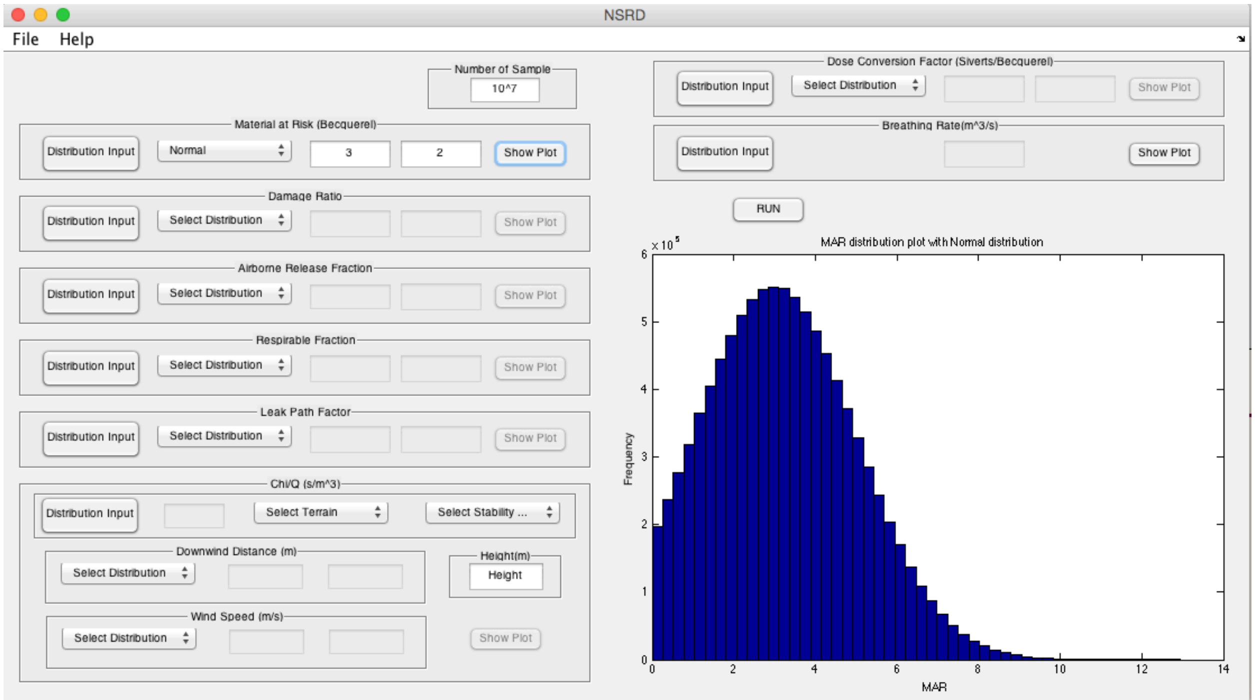


Figure 5. SODA GUI screen.

3.1. Coding Framework

The 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 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 easy to use integrated development environment (IDE). MATLAB is a high-level programming language; thus it is easy to code. 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, high-level languages are a good choice to develop 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 greatly reducing the time 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 code to standalone application for Mac and Windows computers.

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. 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 activates the quick start menu shown in Figure 6.

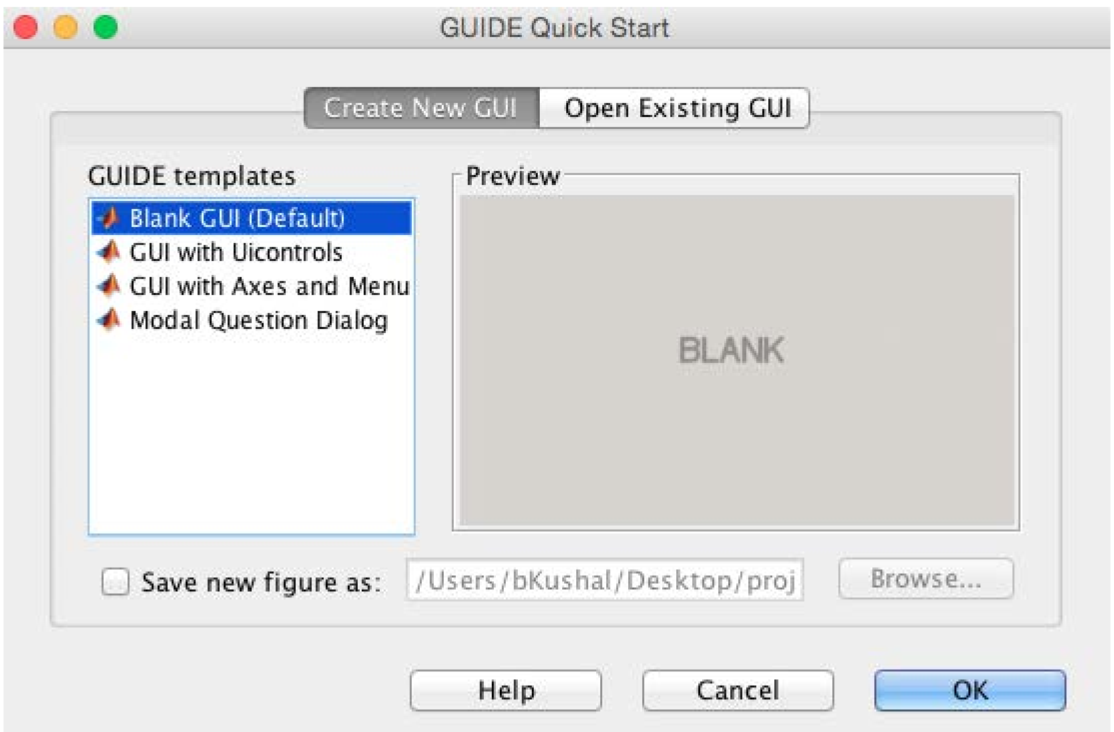


Figure 6. MATLAB GUI quick start.

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.

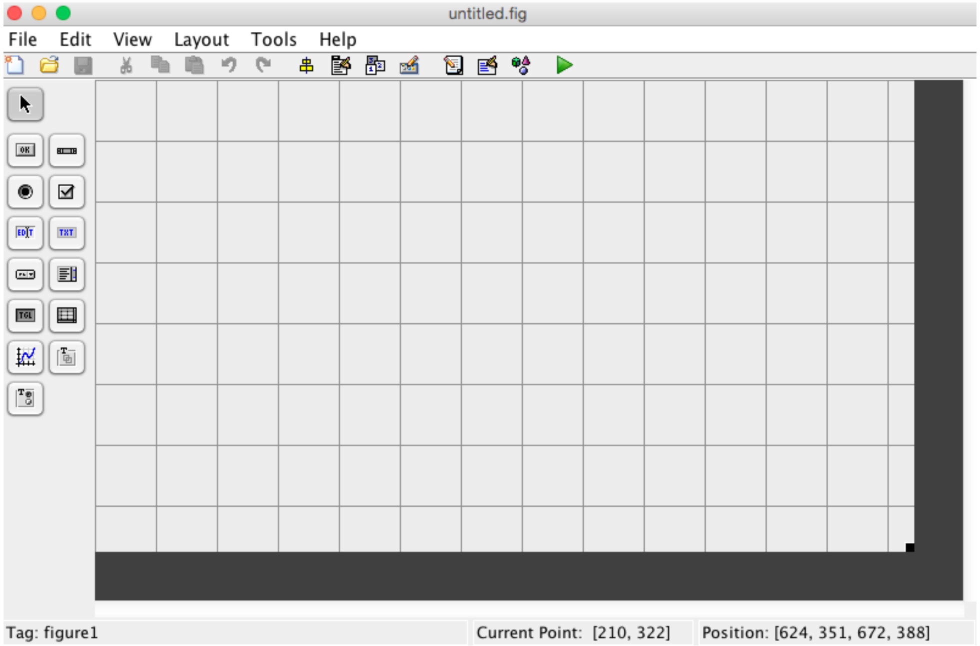


Figure 7. Default GUI figure window in MATLAB.

On the side panel shown in Figure 7, there are lists 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,* and *Panel* tools were used. The SODA application uses two figures created using the GUI figure window: the Main Window and the About Window. First, the About Window shown in Figure 8 provides the logo of DOE and name of ISU, as well as static text identifying the names of the people involved in the project.

Figure 8. SODA *About* window.

The main 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).

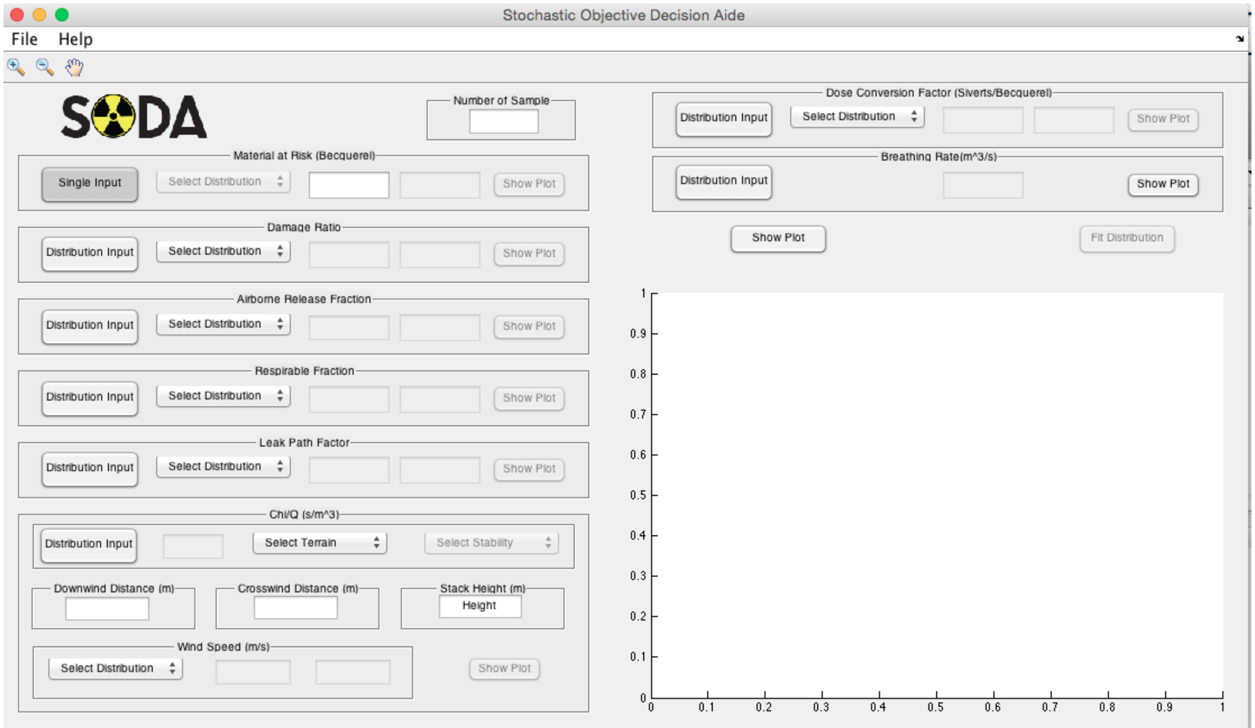


Figure 9. SODA main window.

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 the user exit the application with a confirmation to make sure that the user is about to exit the application.

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. For each input parameter, the user can select from four different probability distributions: Normal, Uniform, Beta, and Exponential. 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. Finally, for the exponential distribution, the user is required to input the mean. Once the 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:

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

where mu is the mean or expected value, and sigma is standard deviation. Examples of normal distributions using different mean and standard deviation values are provided in Figure 10.

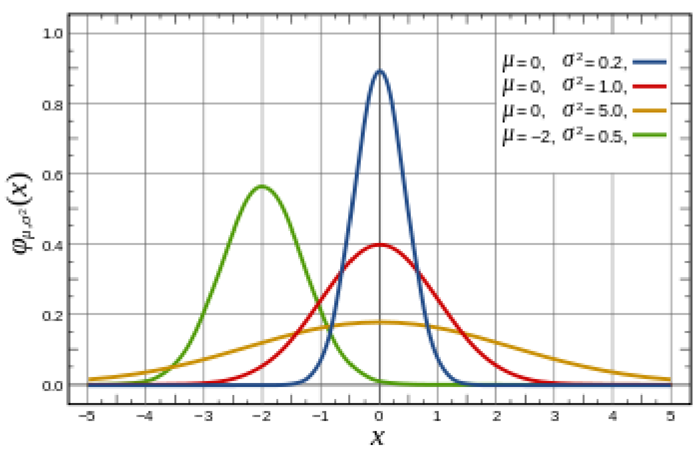


Figure 10. Normal distributions.

The beta distribution, plotted in Figure 11, 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 in between zero and one, it can be used for damage ratio and LPF.

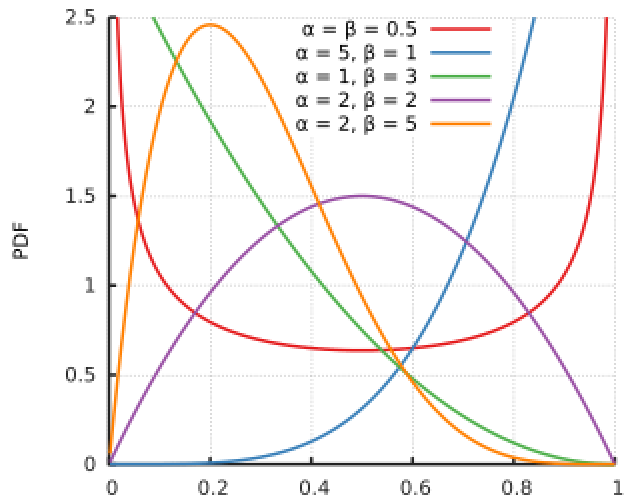


Figure 11. Beta distribution.

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 12.

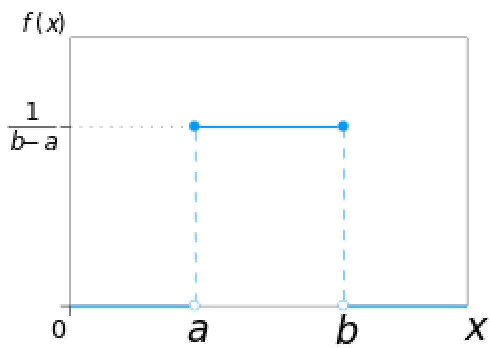


Figure 12. Uniform distribution.

The exponential distribution describes a process in which events occur continuously and independently at a constant average rate lambda (see Figure 13). The exponential distribution equation is:

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

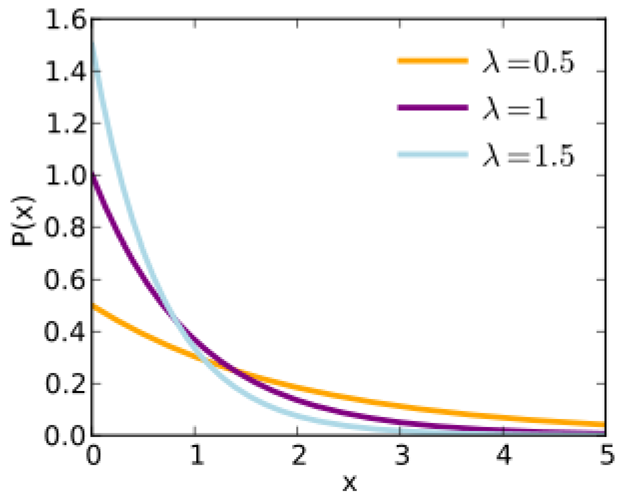


Figure 13. Exponential distribution.

In addition to the distributions described above, a specialized BR distribution is available in the application. The distribution is intended to be reasonable and yet conservative. 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 BR 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 BR 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. The resulting BR distribution is shown in Figure 14.

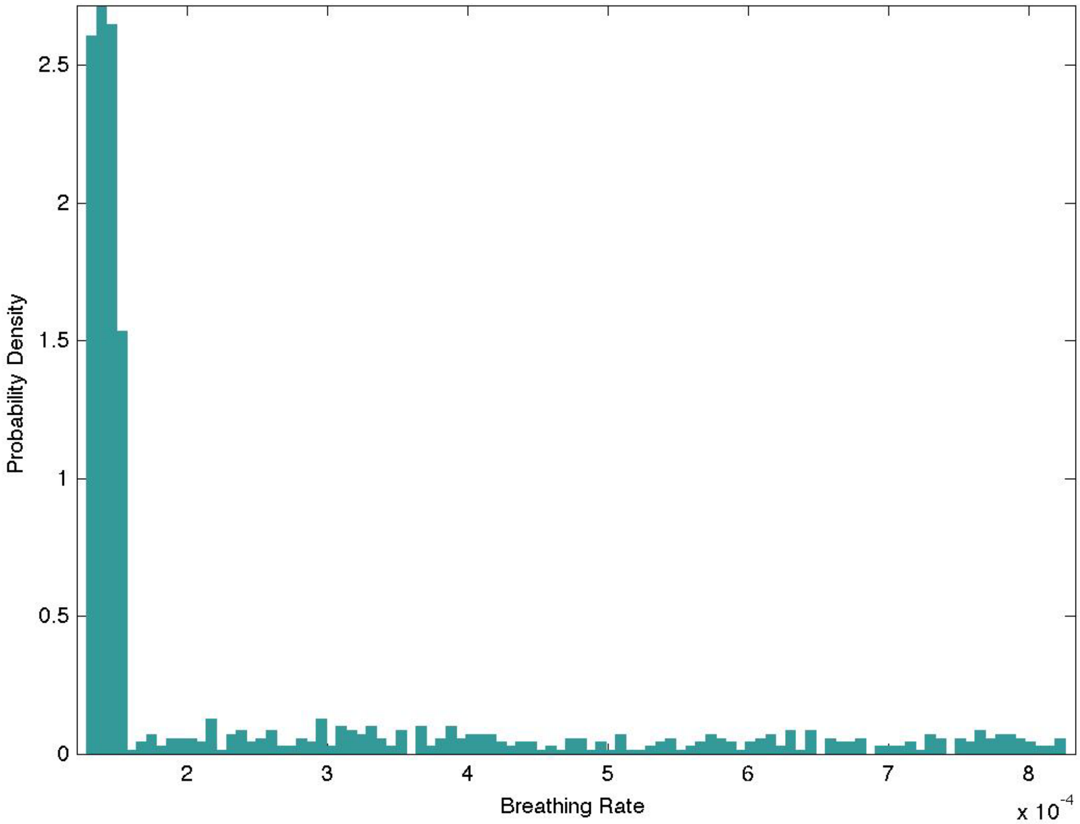


Figure 14. Breathing rate distribution.

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 test. The 3-m drop test 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 D contains the drop test results data.

3.5. Bayesian Information Criterion

Bayesian Information Criterion is the method by which SODA finds the best possible probability distribution for the resultant CED distribution. 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. Figure 15 below provides an example of a BIC fit.

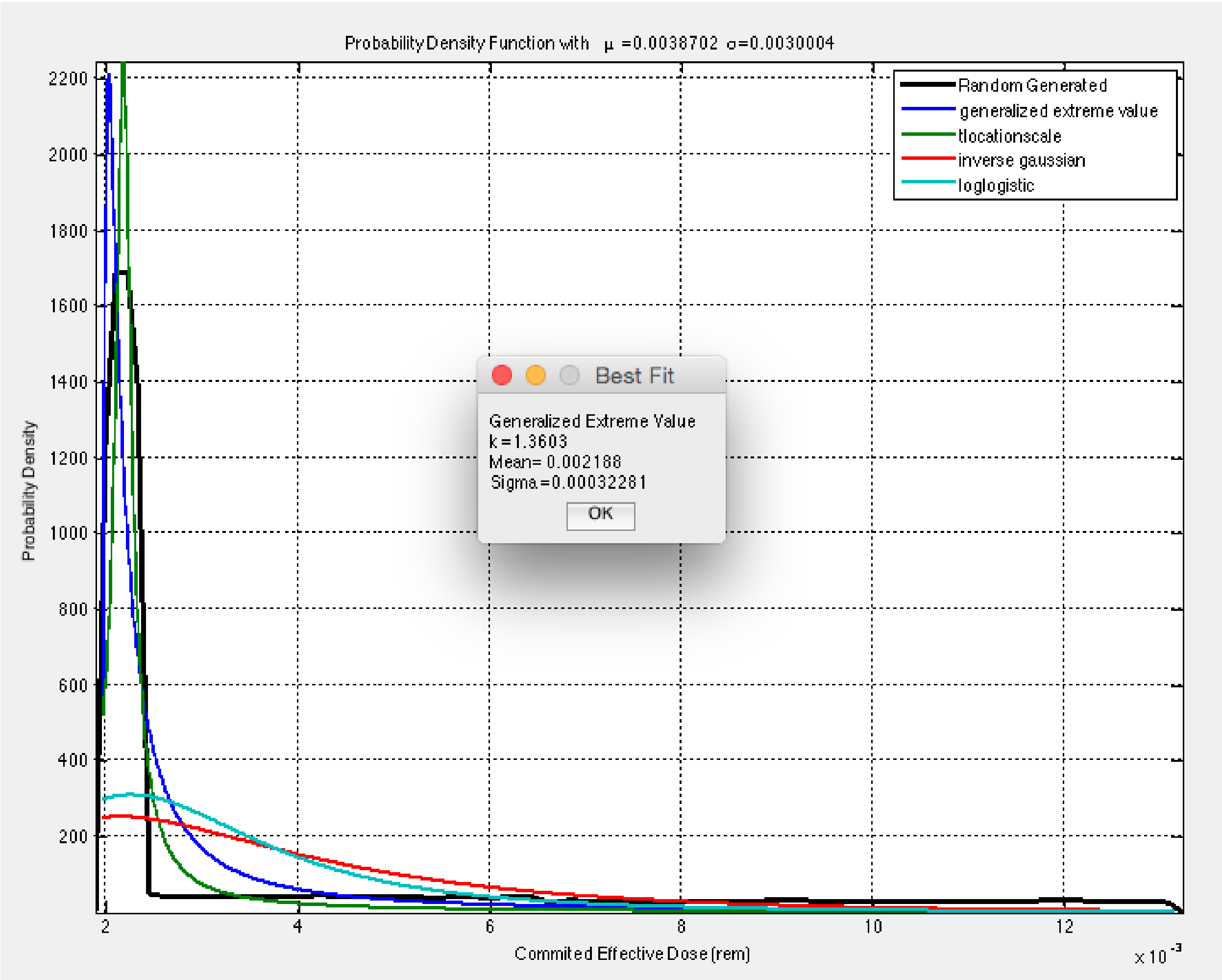


Figure 15. Example grapnhical representation of BIC fit to a randomly generated dose distribution.

4. COMPARISON WITH RSAC

Numerous improvements were identified during beta testing conducted by undergraduate students. Verification calculations were also performed to ensure that SODA was performing properly. The application 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 x below. Results obtained from RSAC, SODA, and a hand calculation are provided in Table y 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 MAR 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 MAR limits and potentially over selection of safety SSCs.

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.

To further magnify SODA, it is proposed that the code be expanded to allow more detailed MAR selection, incorporate user defined input variable distributions, perform a comprehensive parametric study, and complete code verification and validation. Systematic investigation of each input parameter contributing to the dose result can be pursued to quantify the parameter’s contribution to the overall dose estimate uncertainty. Systematic study of each contributing parameter will lead to identification of the parameters that have the largest impact on the resulting dose estimate uncertainty. Once identified, investigation into reducing uncertainty in the key parameters can be accomplished.

Appendix A:
SODA Help File Information

Provided below is a listing of the help file topics that are linked within SODA to assist users with various issues. Following the summary table, the actual images that are included in the SODA help file are provided.

SODA Help File Summary

1. File loading …… 22

2. File saving …… 22

3. Image saving …… 23

4. Random number resetting …… 23

5. Application exiting …… 24

6. Entering Values for Parameters …… 25-38

6.1. Entering Number of Samples …… 25

6.2. Entering Values for MAR Parameters …… 25

6.3. Entering Values for Damage Ratio Parameters …… 27

6.4. Entering Values for ARF Parameters …… 28

6.5. Entering Values for Respirable Fraction Parameters …… 30

6.6. Entering Values for LPF Parameters…… 31

6.7. Select Terrain for Chi/Q…… 33

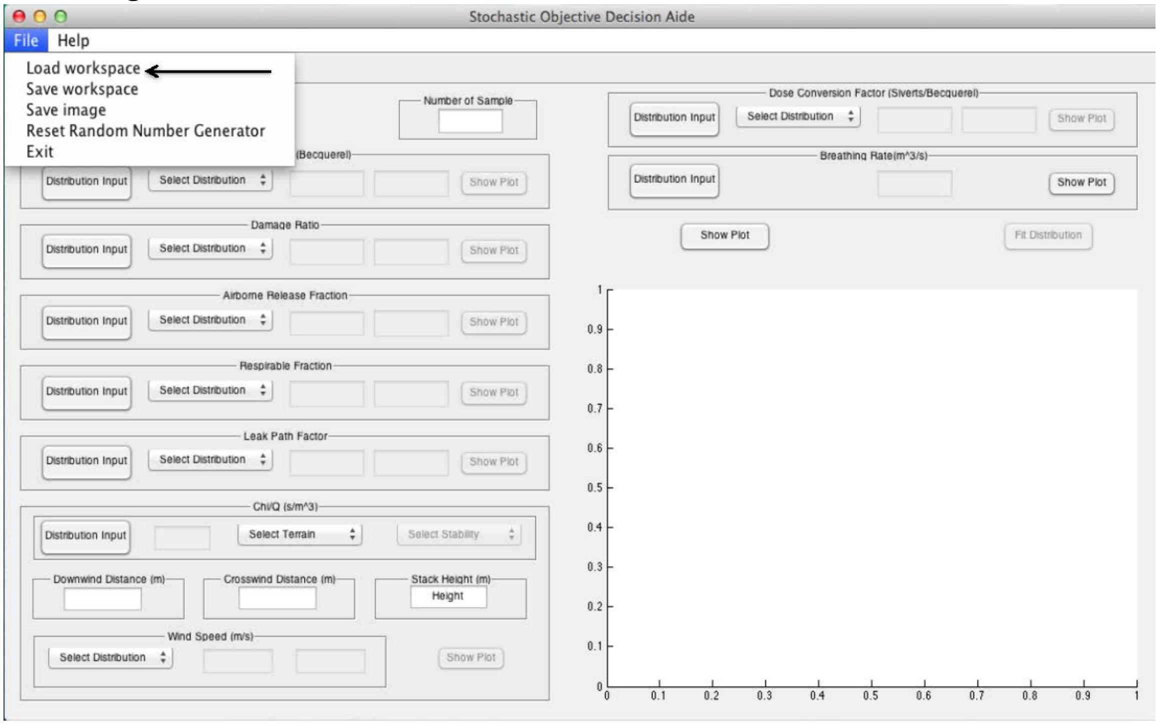
6.8. Entering Values for Downwind Distance, Crosswind Distance, and Stack Height …... 34

6.9. Selecting Distribution for Wind speed …… 35

6.10. Entering Values for Dose Conversion Factor Parameters …… 35

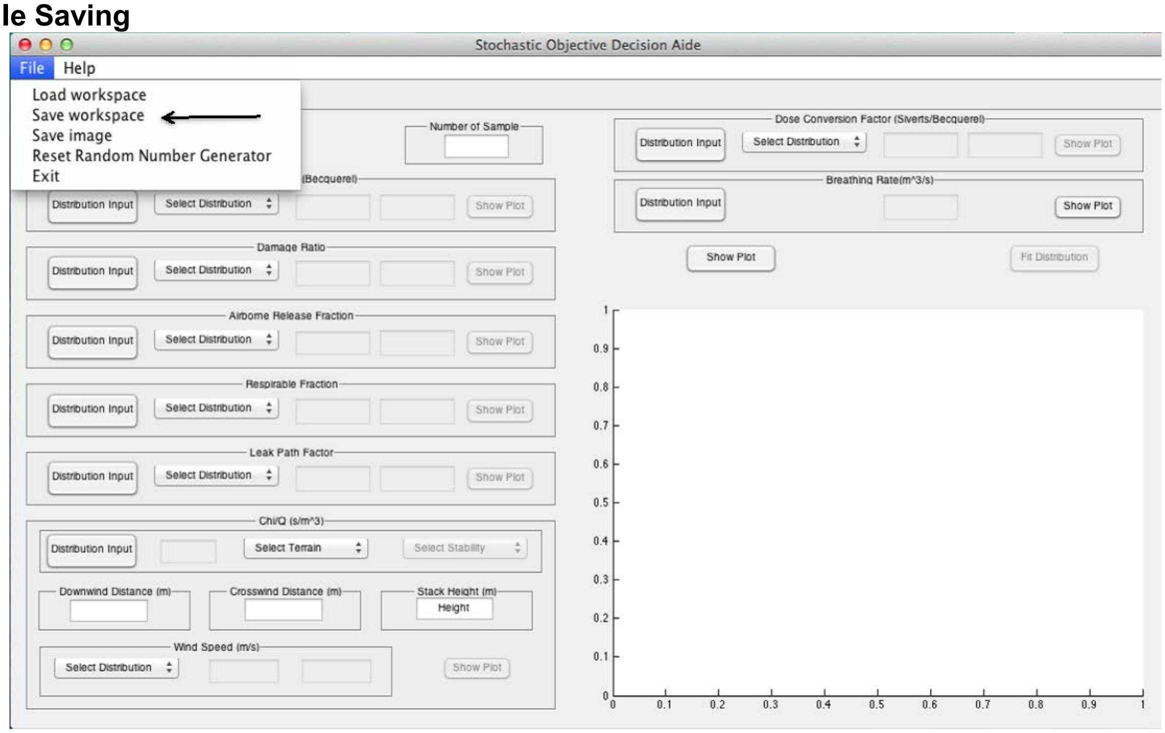
6.11. Entering Values for Breathing Rate Parameters …… 37

A.1. File Loading



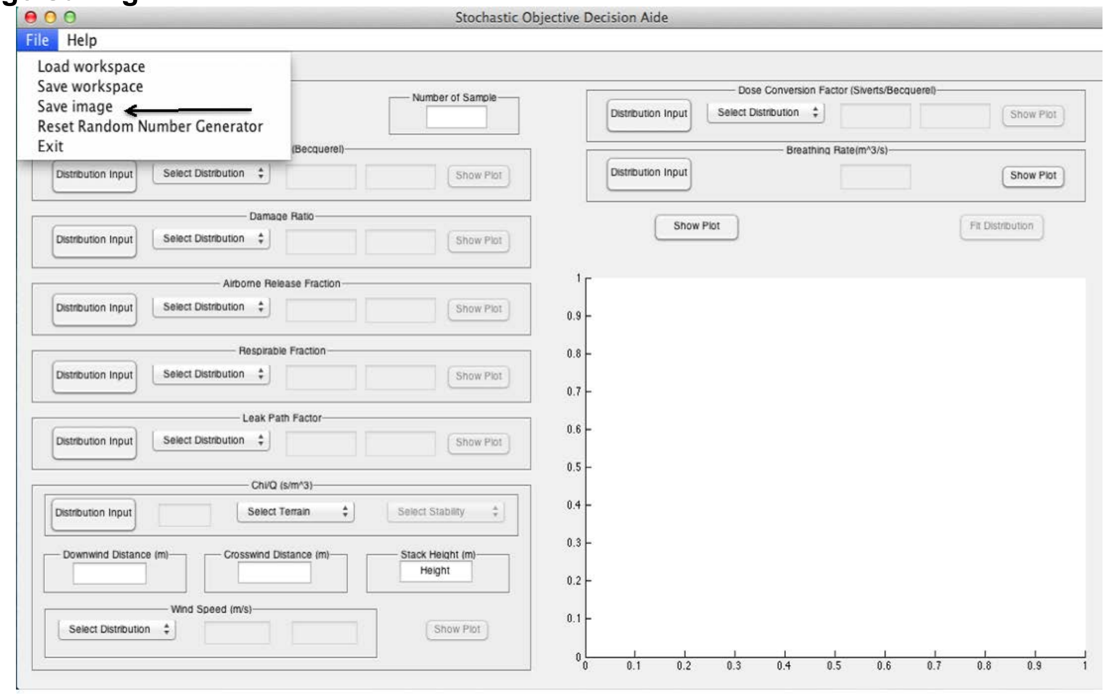
To load a file, go to "file", then "Load workspace"

A.2. File Saving



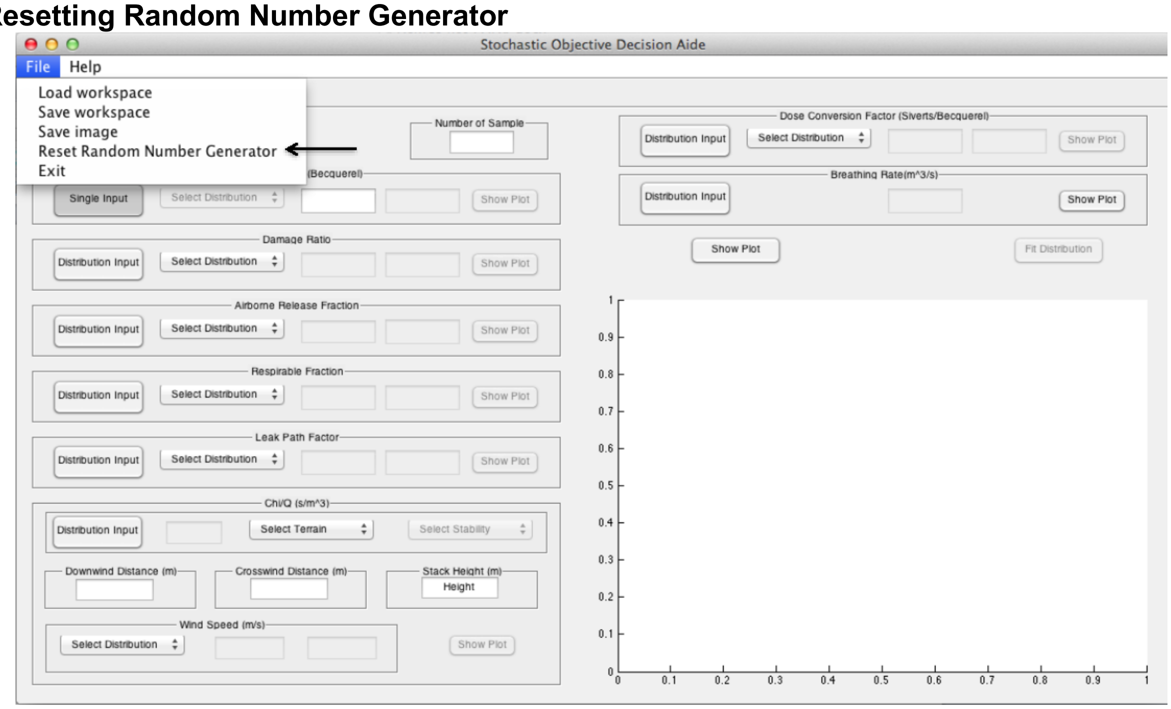
To save a file, go to "file", then "Save workplace".

A.3. Image Saving



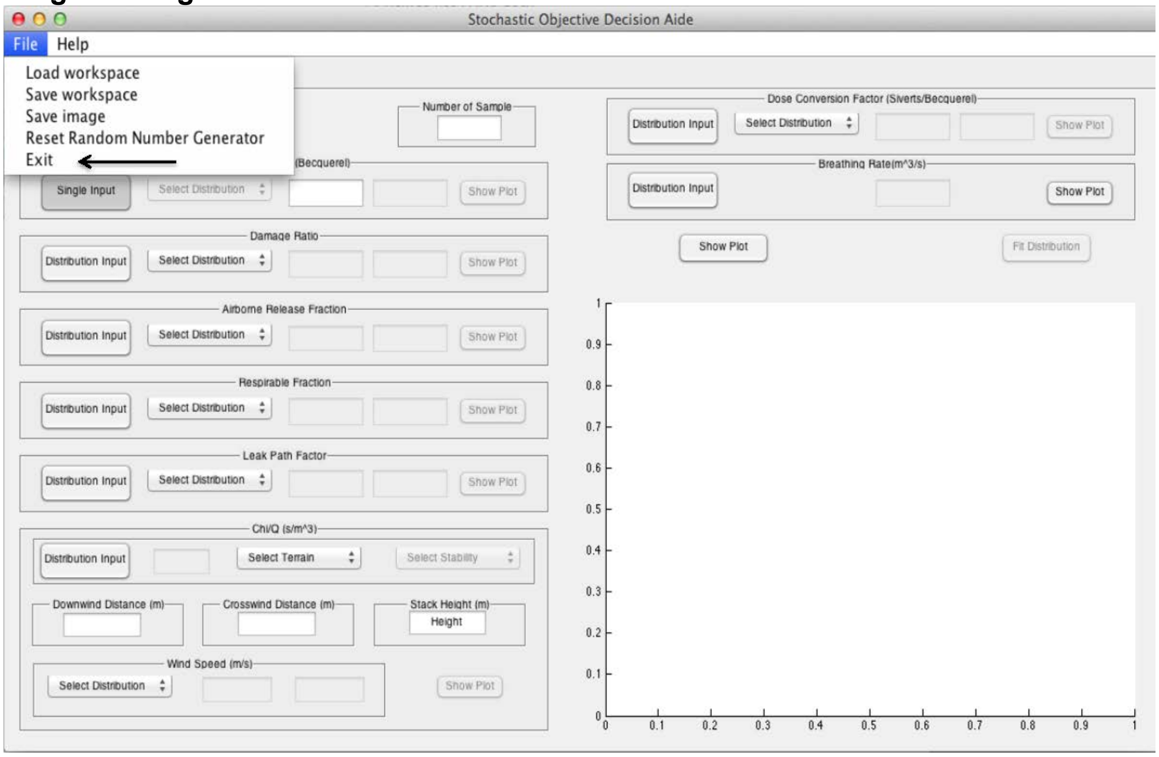
To save an image, go to "file", then "Save workplace".

A.4. Resetting Random Number Generator



To reset the Mersenne Twister random number algorithms so that the calculations can be performed in two machines or repeated to get same result, click on "Reset Random Number Generator".

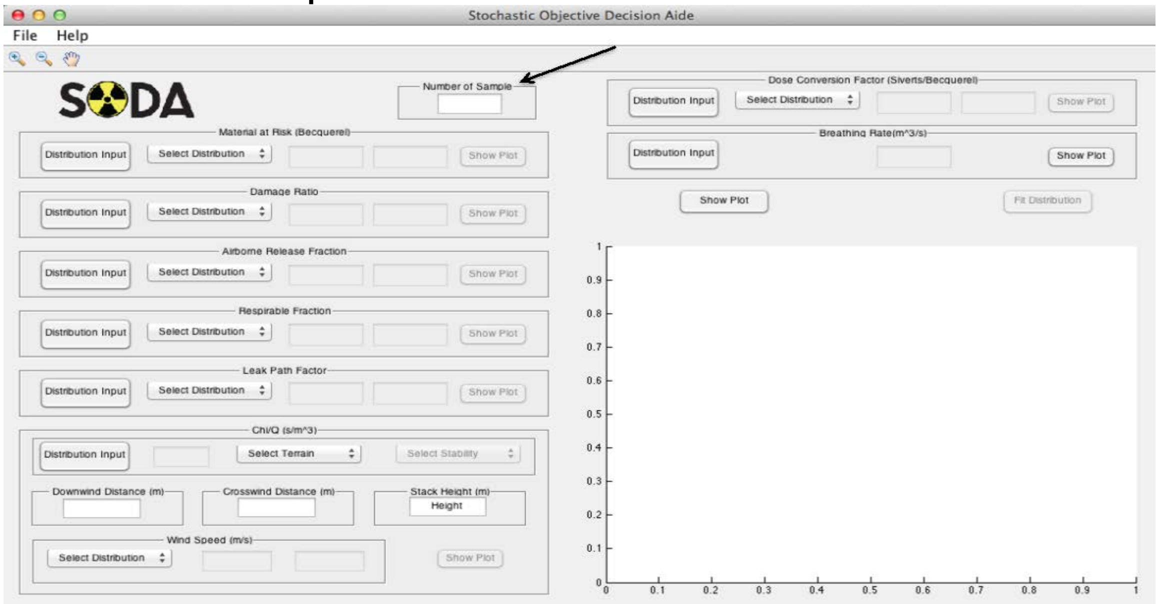
A.5. Exiting the Program



To exit the application, go to ''file'', then ''Exit''.

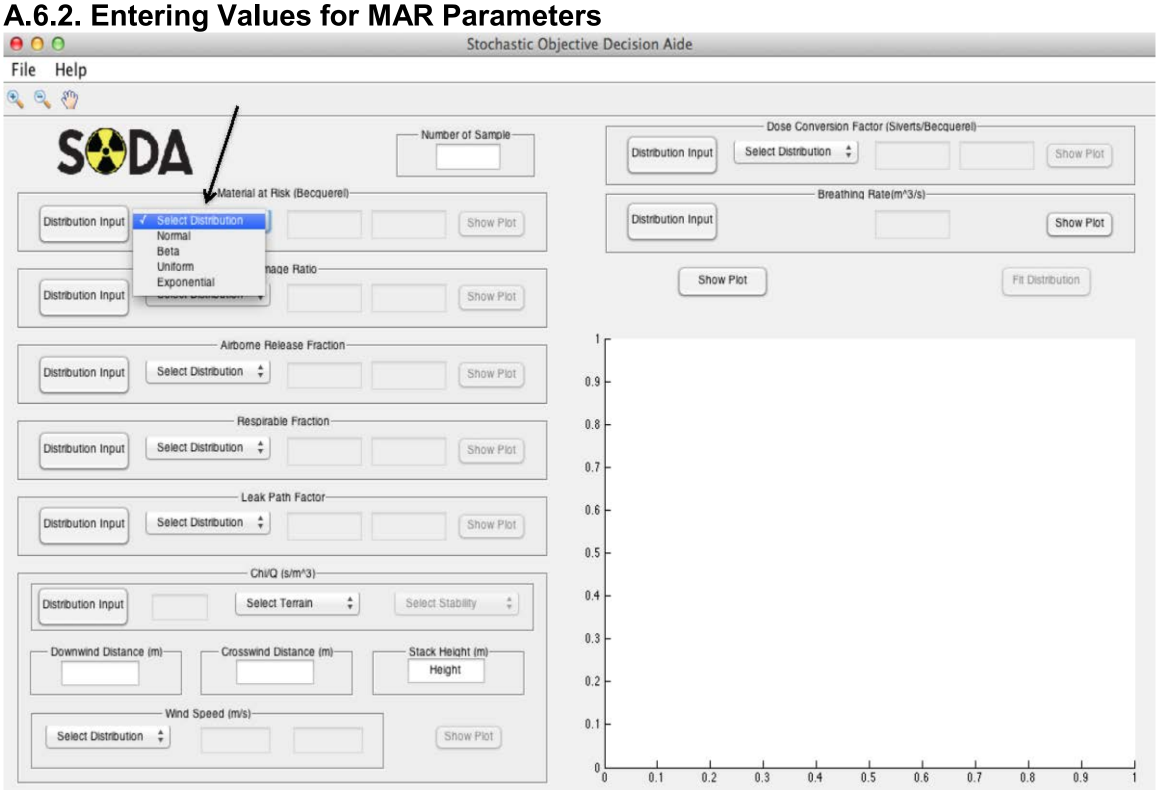
A.6. Entering Values for Parameters

A.6.1. Number of Samples

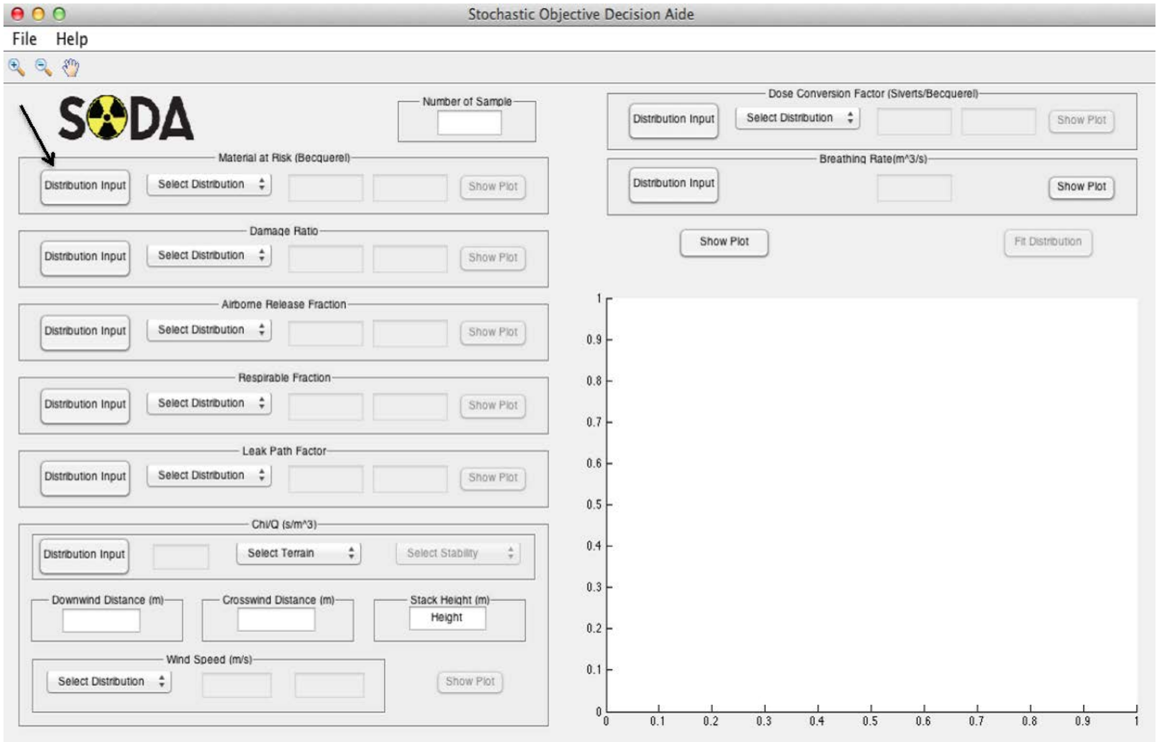


Enter the Number of samples.

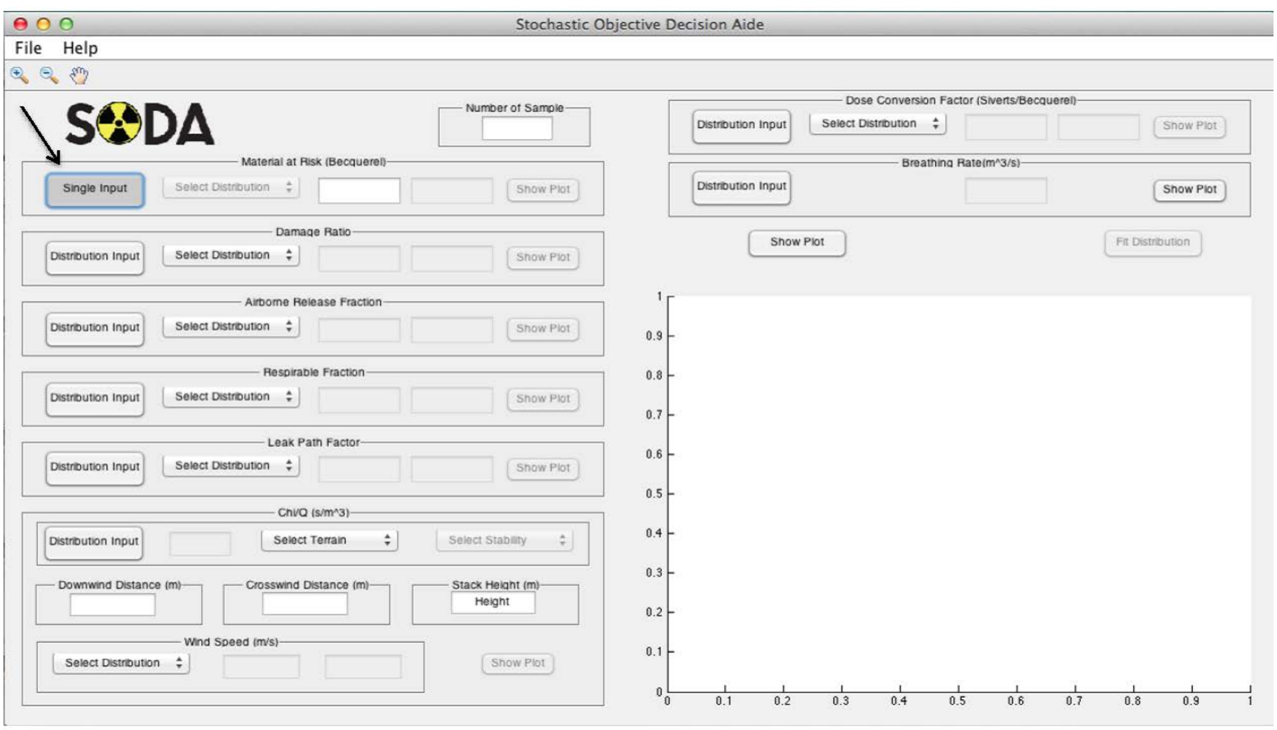
A.6.2. Entering Values for MAR Parameters



In Material at risk, for a distribution input, choose the distribution type (Normal, Beta, Uniform, Exponential)

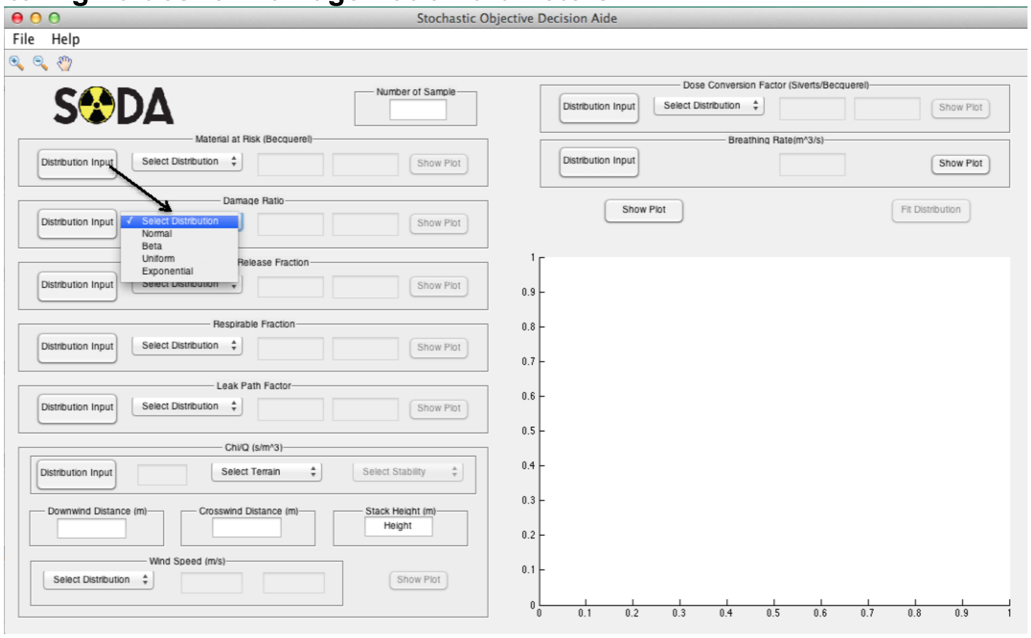


To switch from distribution to single input, click on ''Distribution Input''.

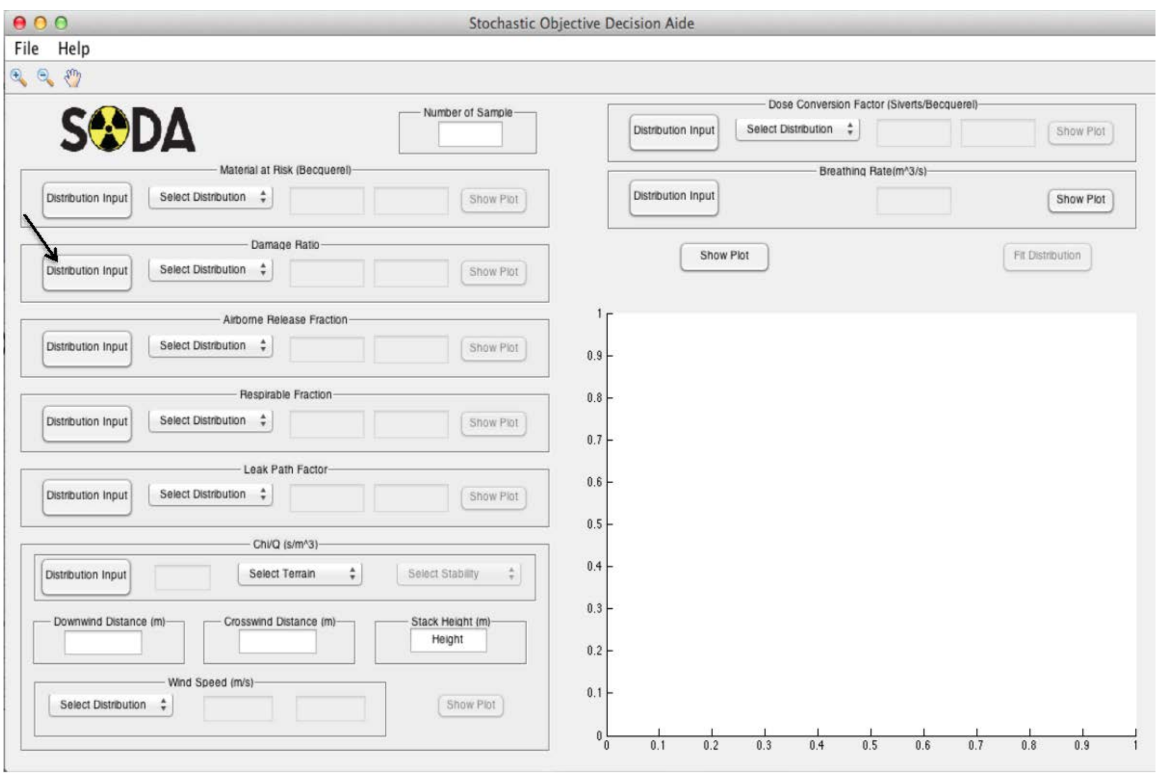


For a single input, enter the number in Becquerel's.

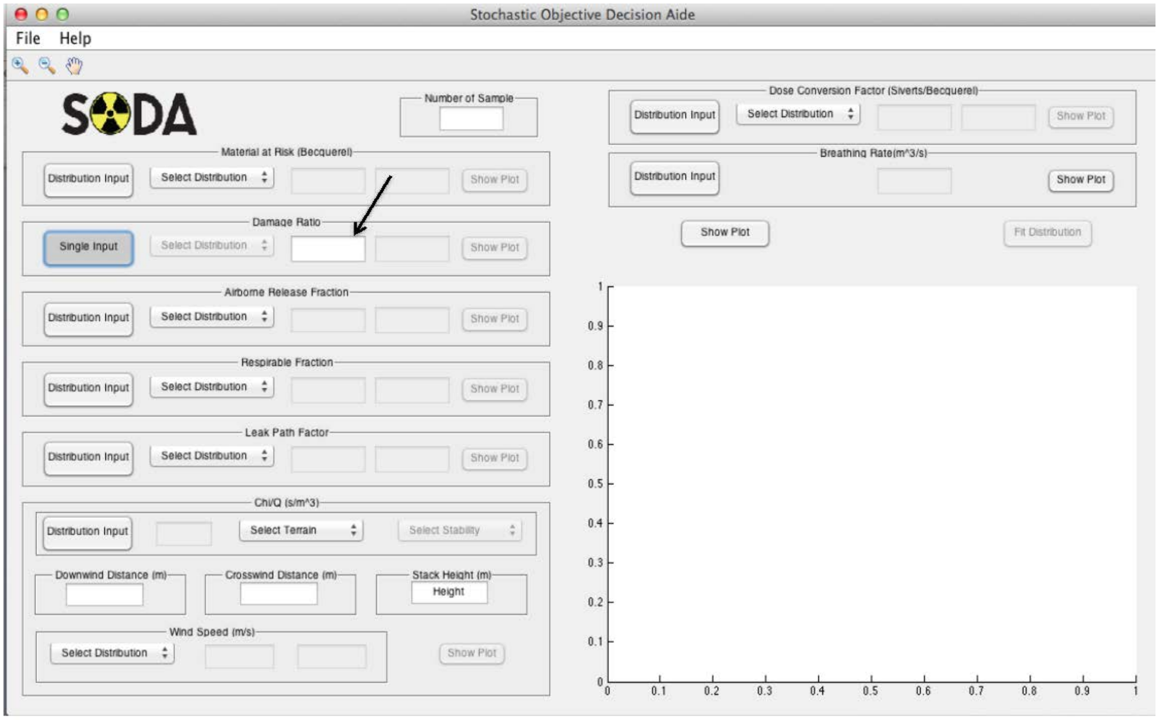
A.6.3 Entering Values for Damage Ratio Parameters



In Damage Ratio, for a distribution input, choose the distribution type (Normal, Beta, Uniform, Exponential)

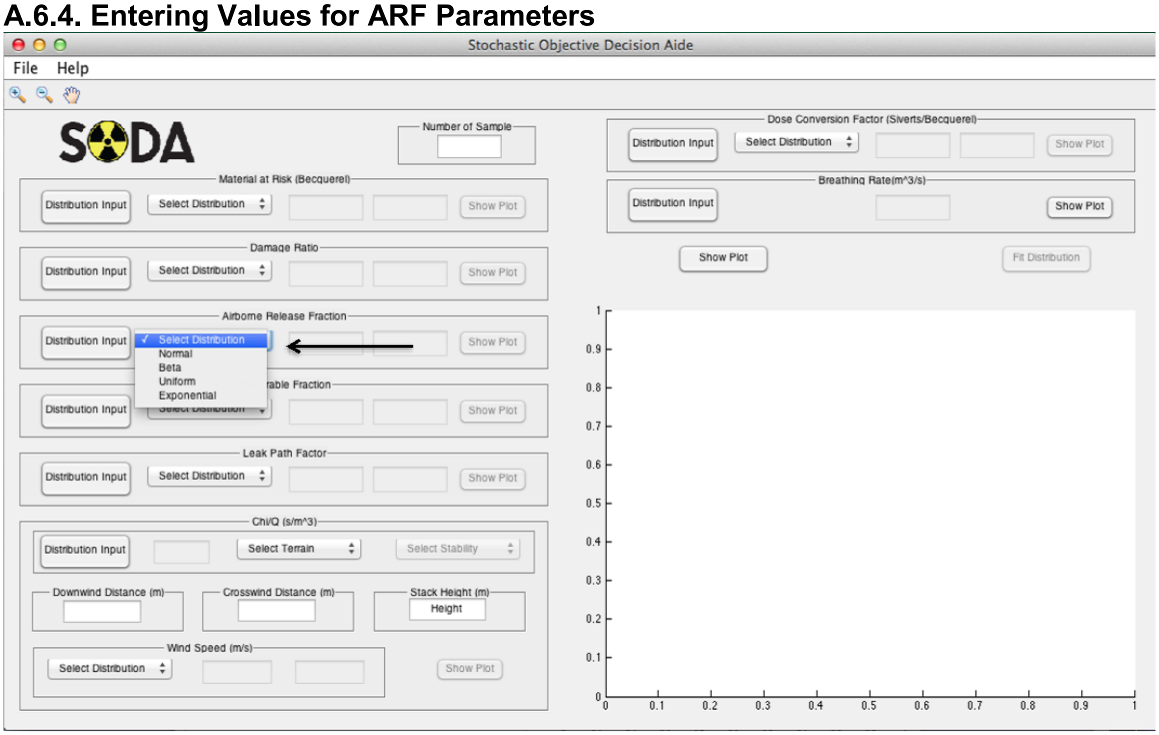


To switch from distribution to single input, click on "Distribution Input"

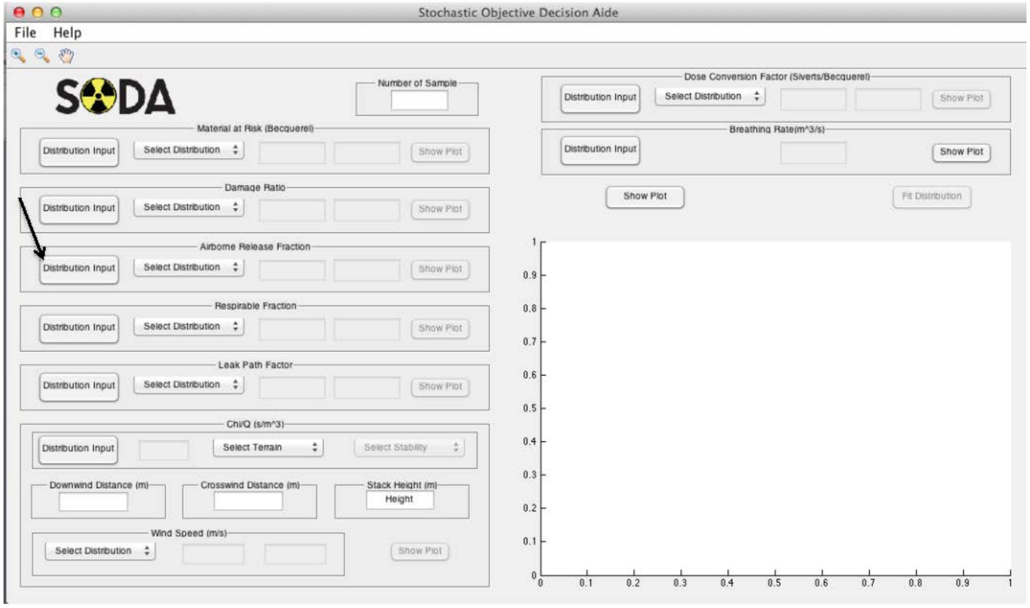


For a single input, enter the number in Becquerel's

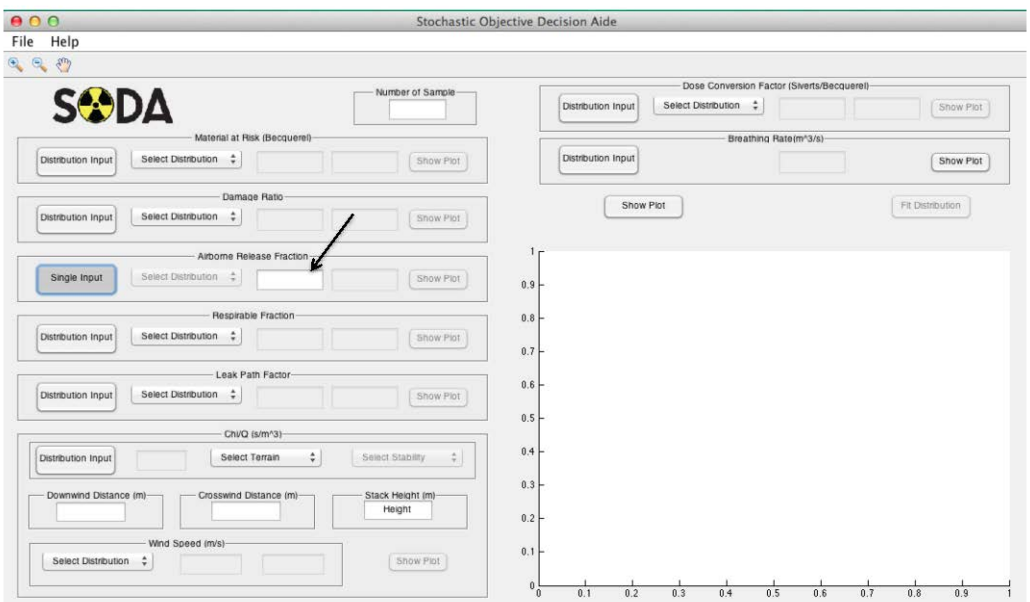
A.6.4. Entering Values for ARF Parameters



In Airborne Release Fraction, for a distribution input, choose the distribution type (Normal, Beta, Uniform, Exponential)

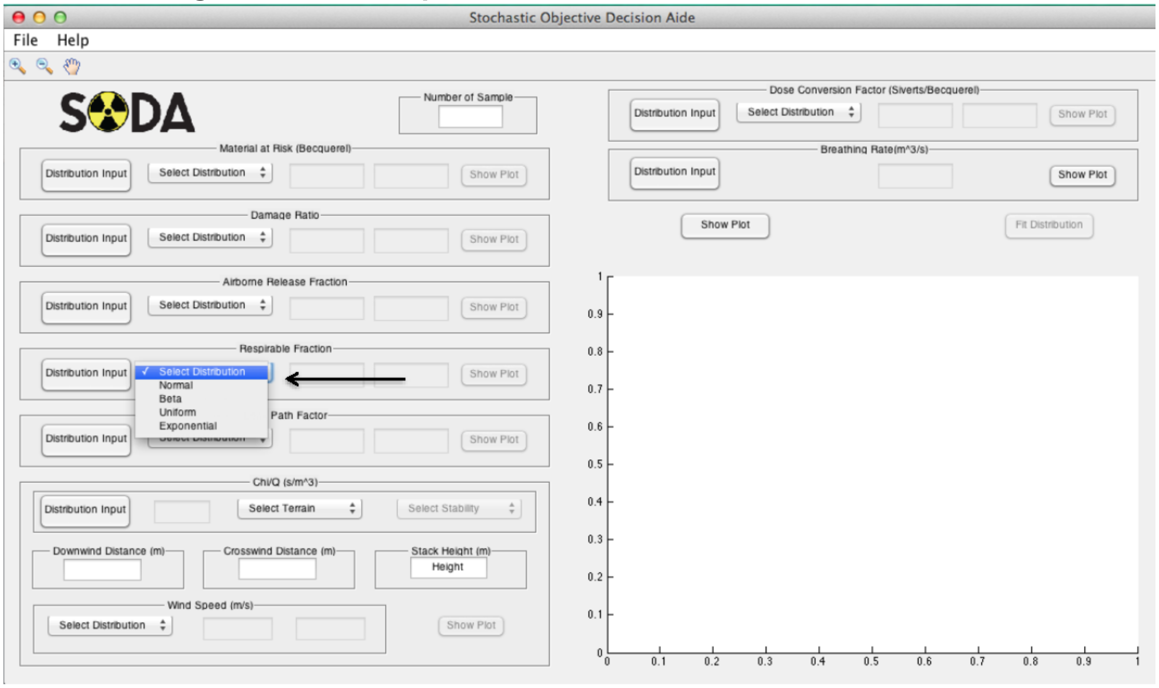


To switch from distribution to single input, click on ''Distribution Input''.

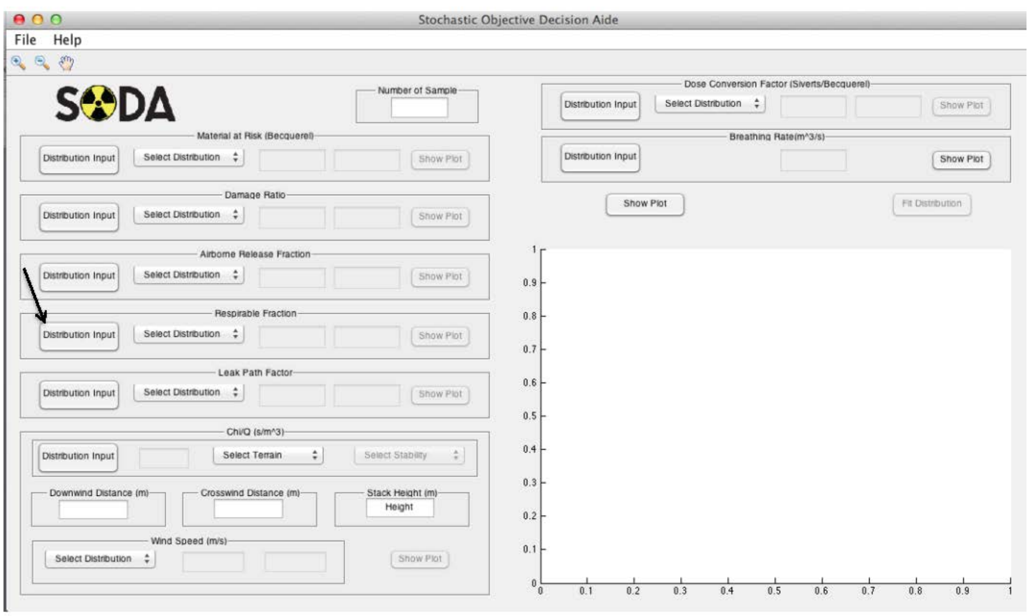


For a single input, enter the number in Becquerel's

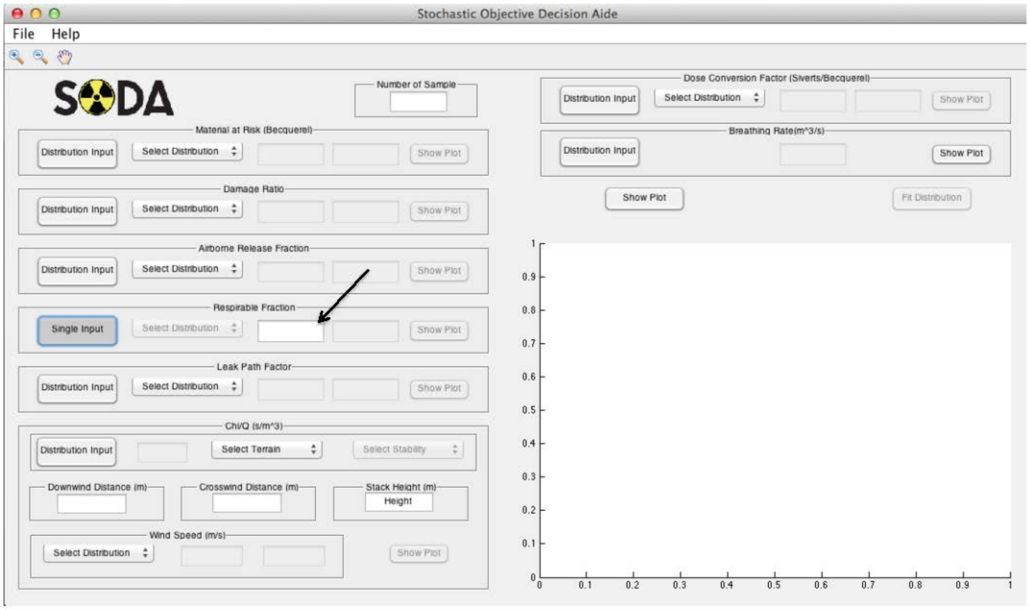
A.6.5. Entering Values for Respirable Fraction Parameters



In Respirable Fraction, for a distribution input, choose the distribution type (Normal, Beta, Uniform, Exponential)

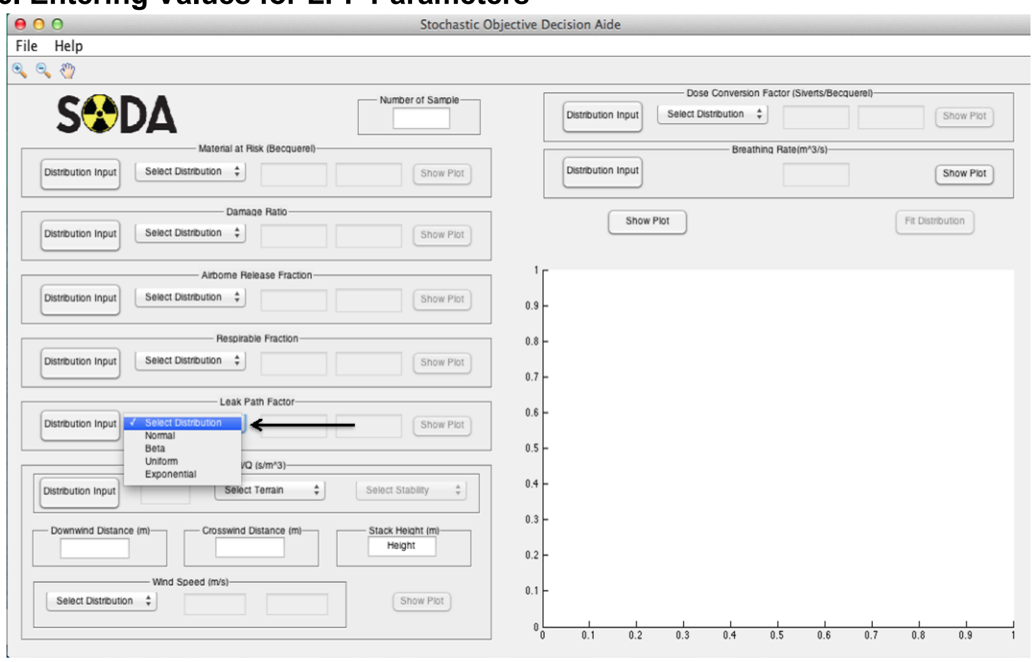


To switch from distribution to single input, click on ''Distribution Input''

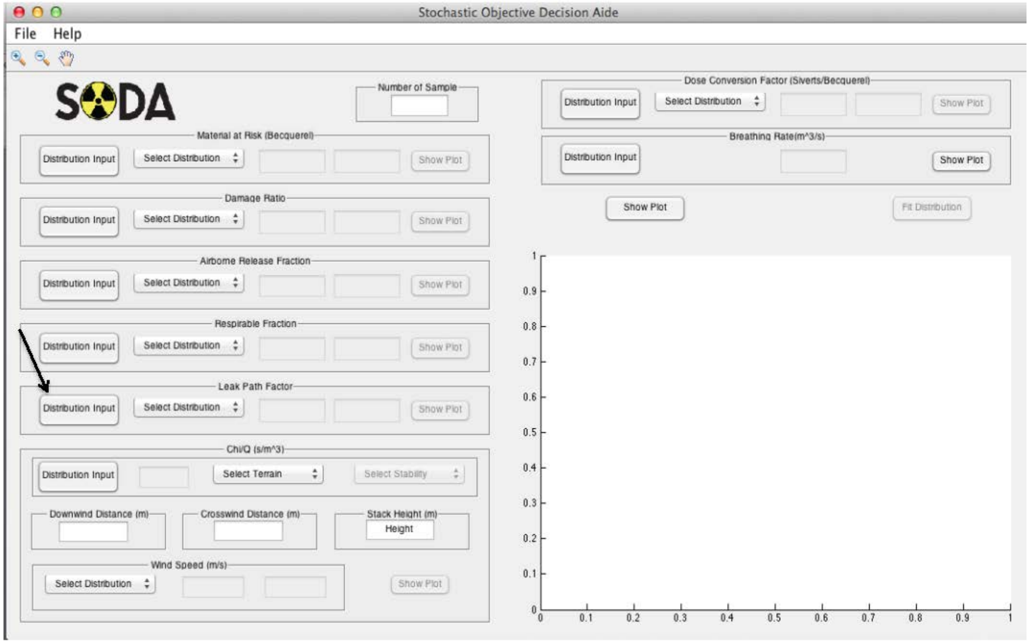


For a single input, enter the number in Becquerel's

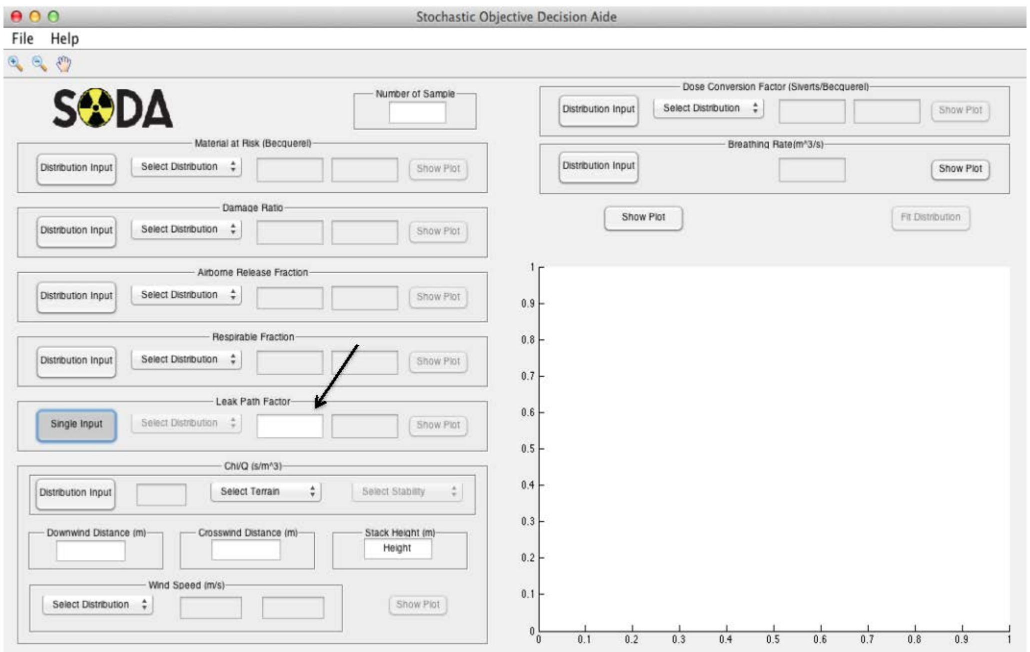
A.6.6. Entering Values for LPF Parameters



In Leak Path Factor, for a distribution input, choose the distribution type
(Normal, Beta, Uniform, Exponential)

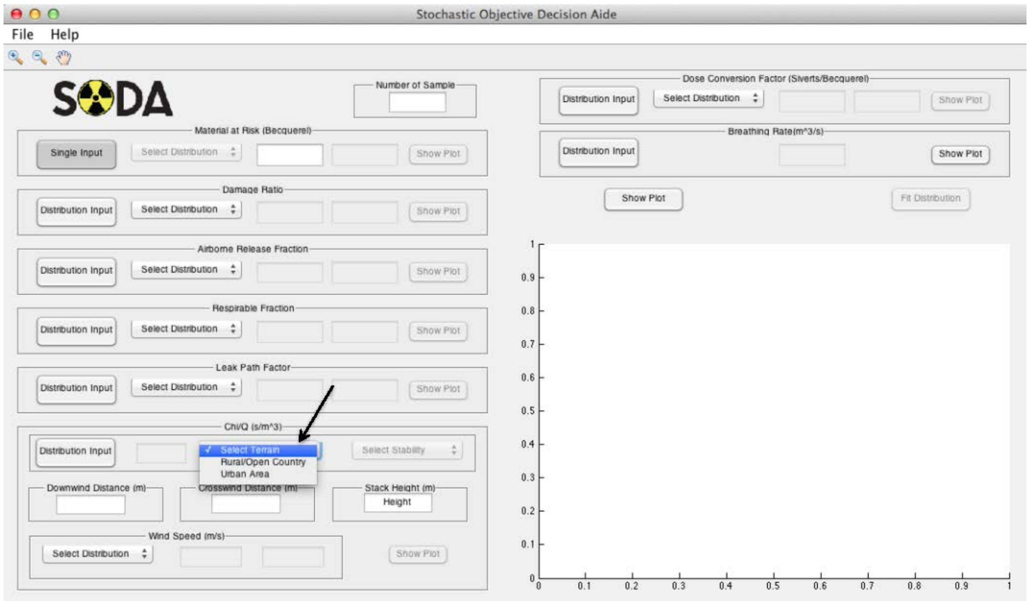


To switch from distribution to single input, click on "Distribution Input"

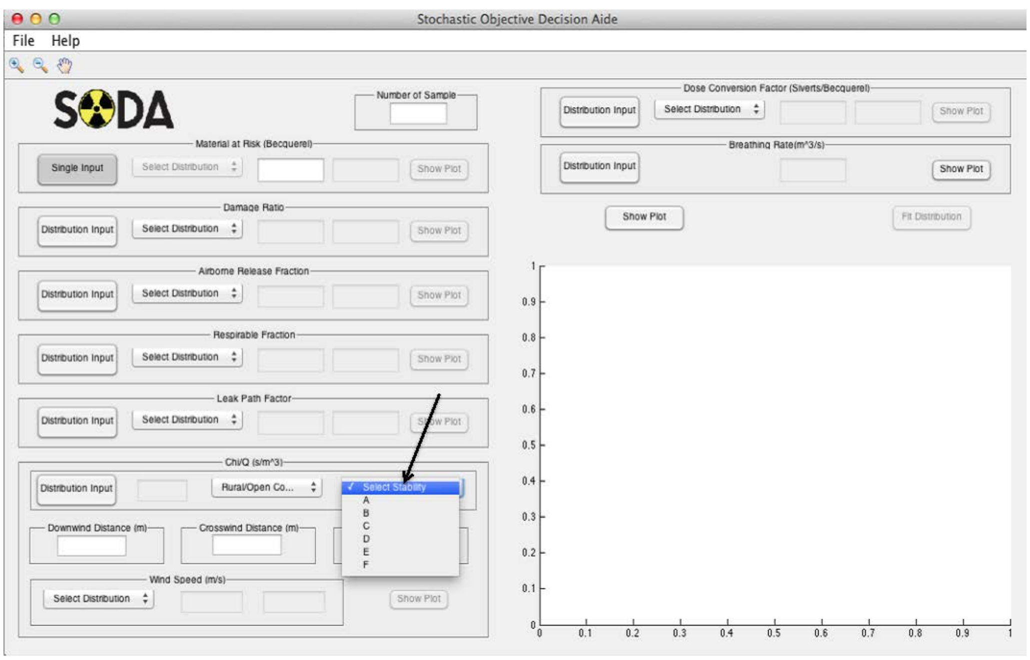


For a single input, enter the number in Becquerel's

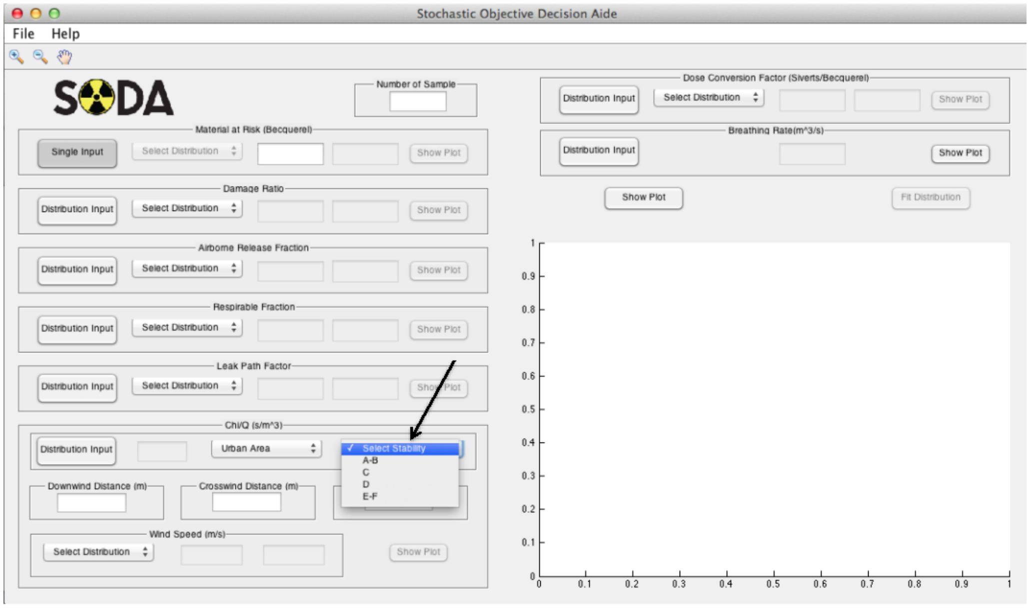
A.6.7. Select Terrain for Chi/Q



For Chi/Q distribution input, choose the terrain (Open country or Urban area)

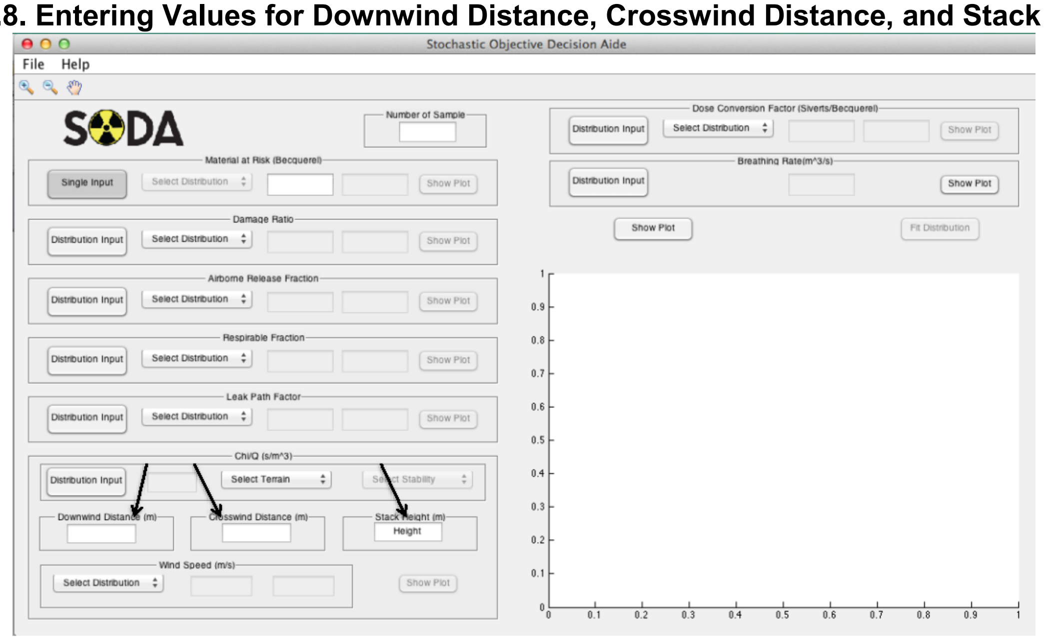


For Chi/Q distribution input, if you choose Open country, then choose the stability (A, B, C, D, E, F)



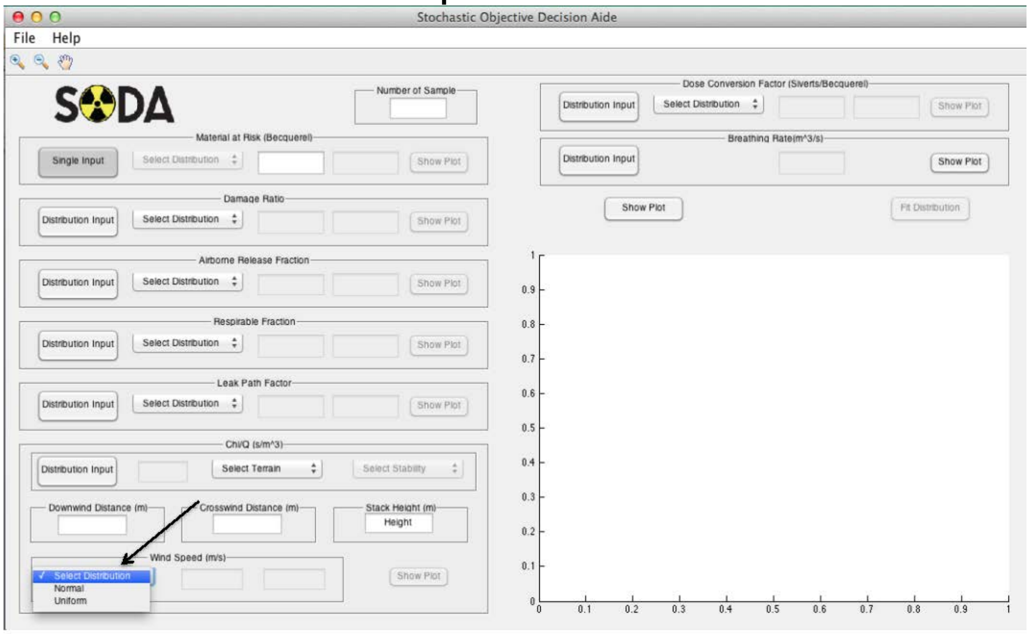
For Chi/Q distribution input, if you choose Urban Area, then choose the stability (A-B, C, D, E-F)

A.6.8. Entering Values for Downwind Distance, Crosswind Distance, and Stack Height



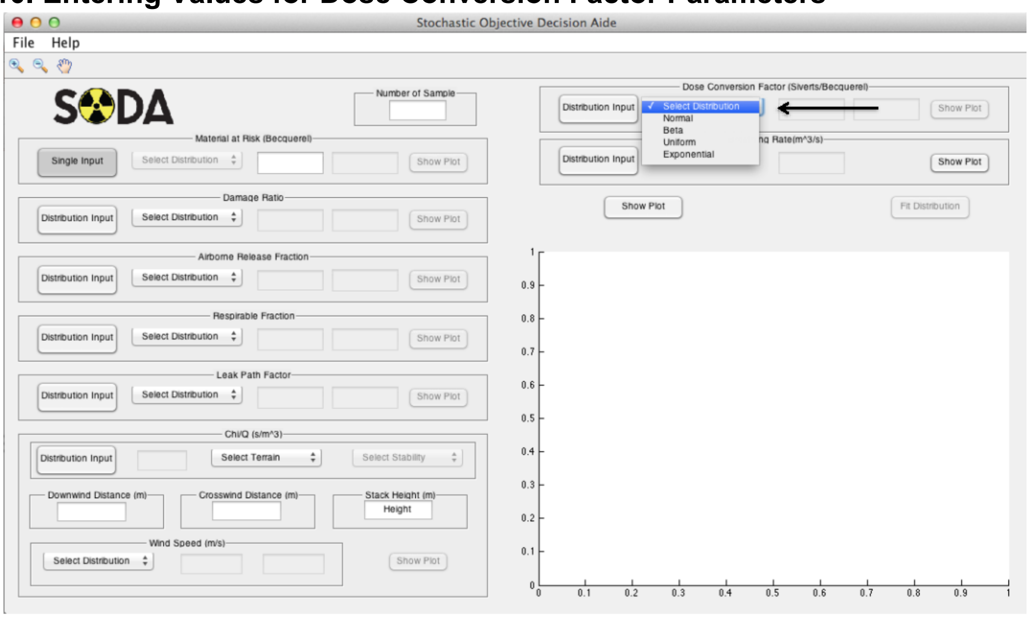
After that, enter the numbers of Downwind Distance, Crosswind Distance, and Stack Height. (All in meter)

A.6.9. Select Distribution for Wind Speed

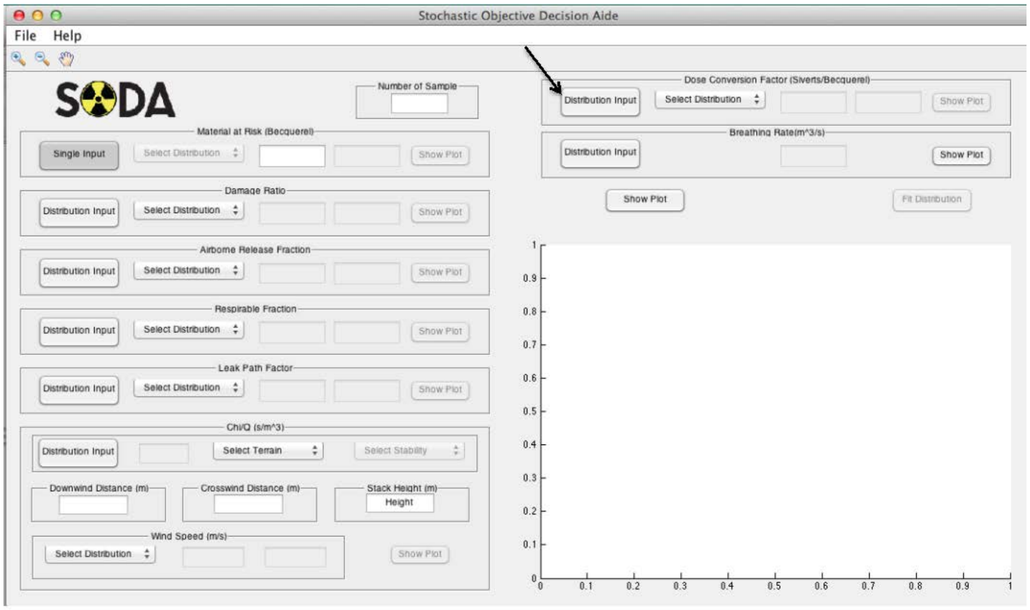


For Wind speed, select the distribution

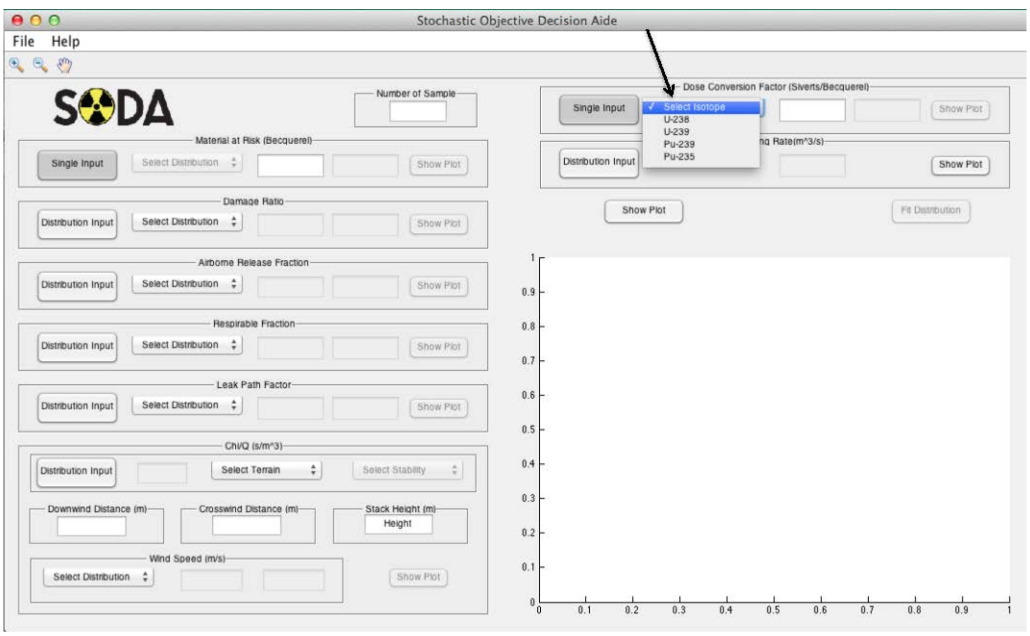
A.6.10. Entering Values for Dose Conversion Factor Parameters



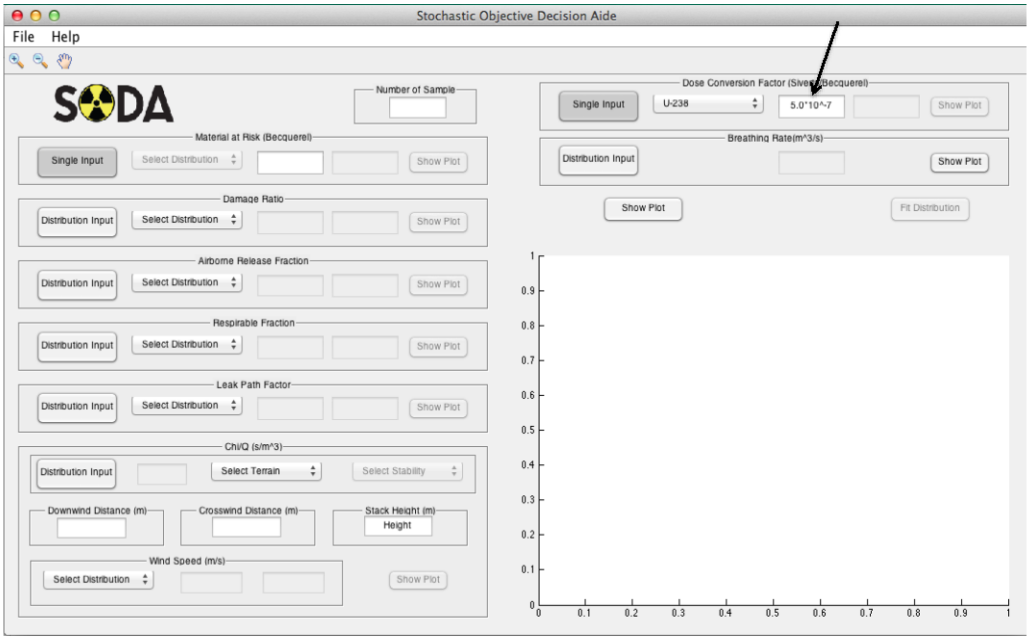
In Dose Conversion Factor, for a distribution input, choose the distribution type
(Normal, Beta, Uniform, Exponential)



To switch from distribution to single input, click on "Distribution Input"

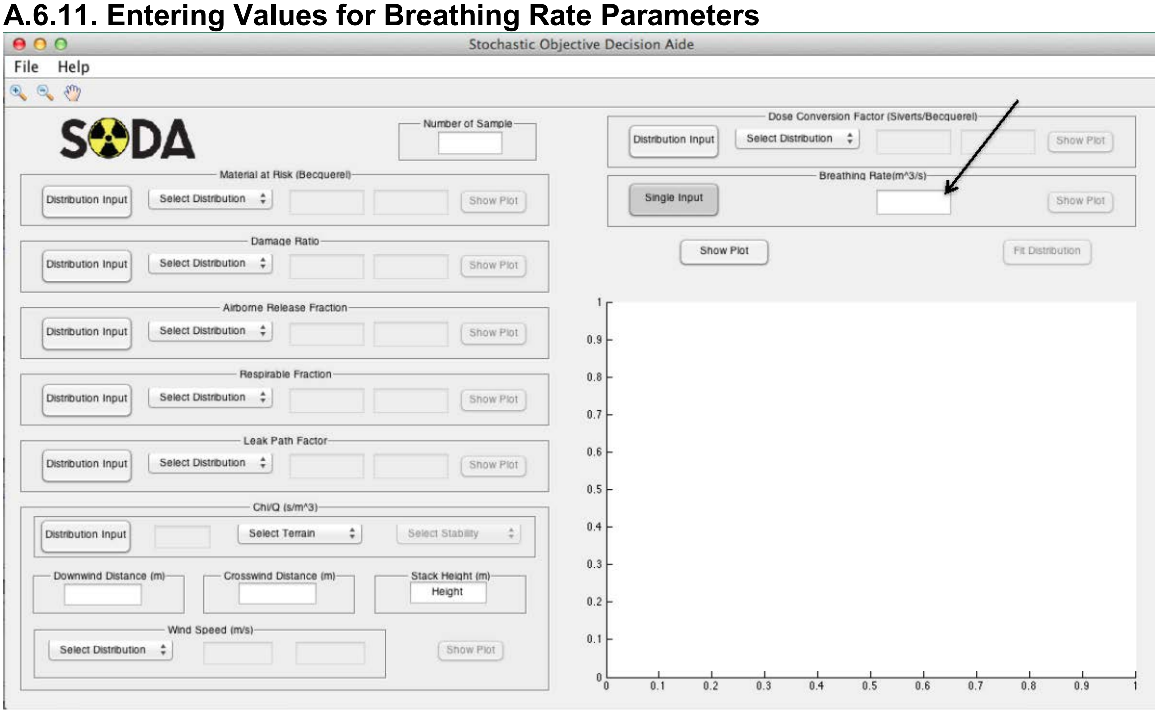


For a single input, select the isotope

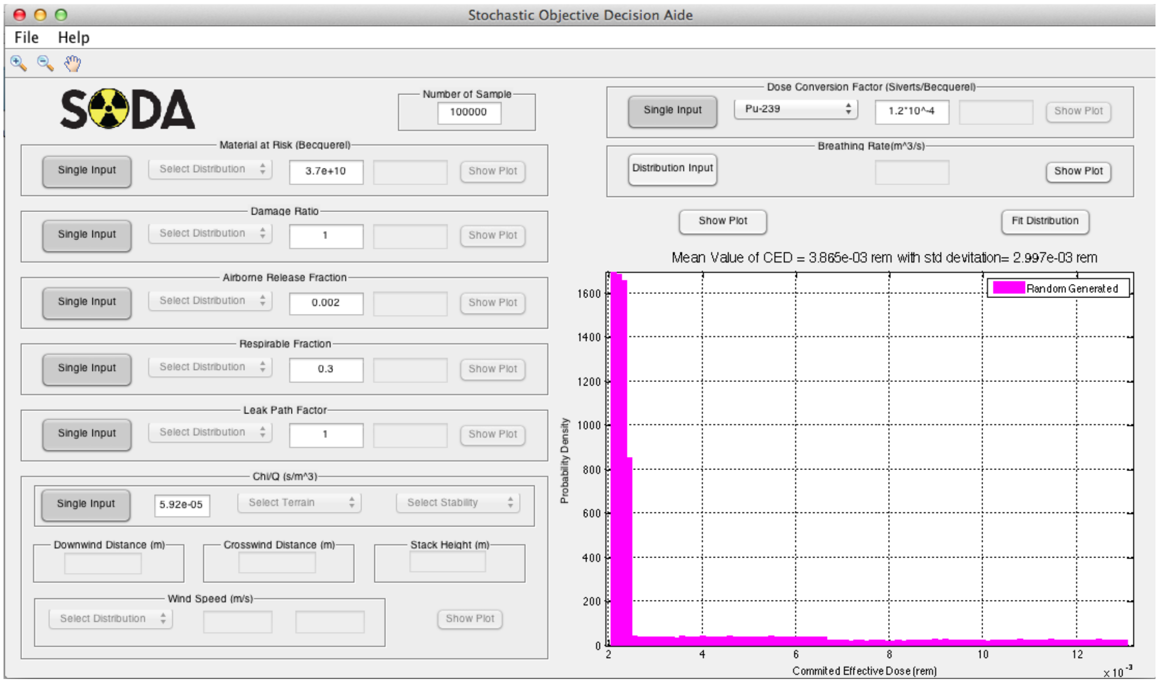


For example, select U-238 and its Dose Conversion Factor will show

A.6.11. Entering Values for Breathing Rate Parameters



For Breathing rate, enter the number in (m^3/s)



th assumptions of 24 hours

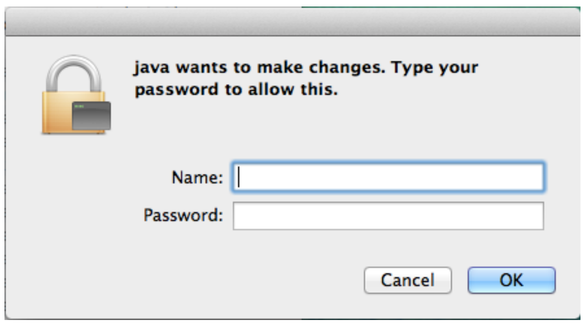
Breathing rate

APPENDIX B:
Installation Instructions

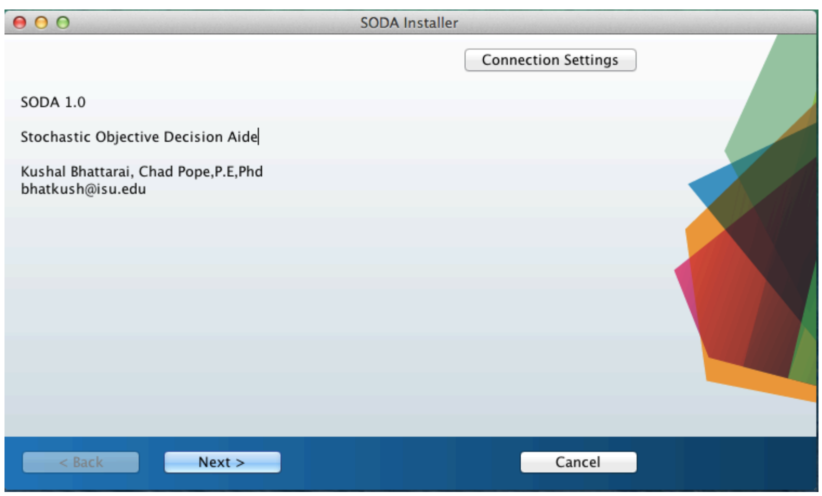
SODA installation instruction for both Mac and Windows PCs are provided below.

Mac Installation

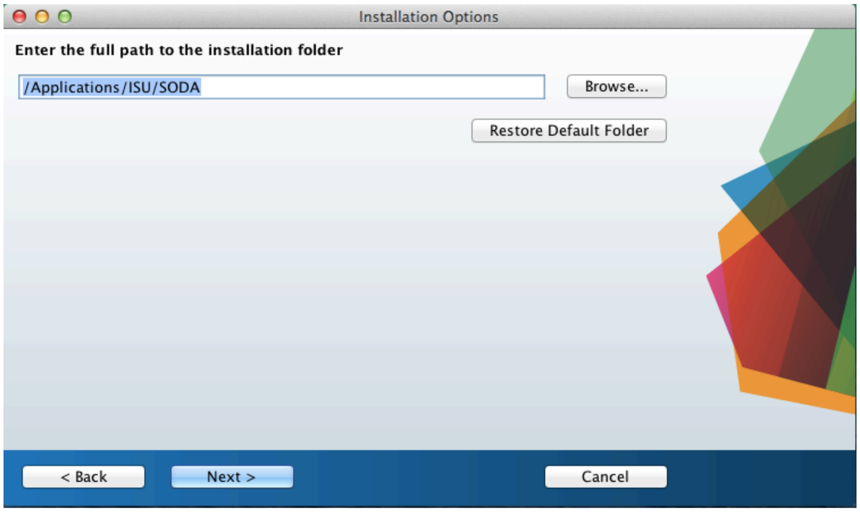
1. Download the program, and the download file will be in the downloads folder. Click to open the installation program.
2. The install application will take a few minutes to load. Please wait then put in your name and password.



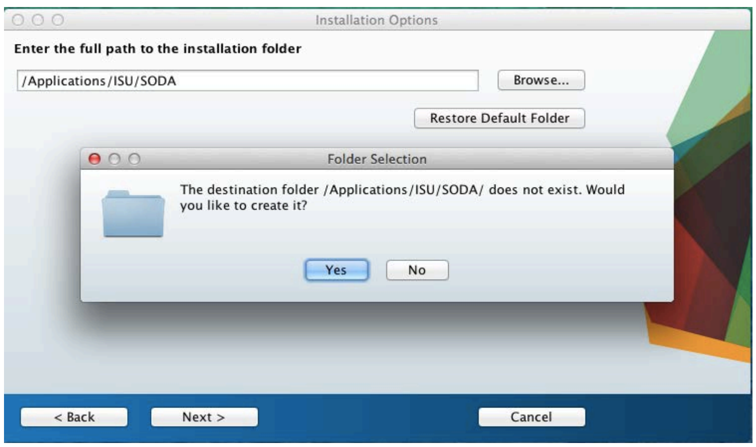
3. When you see the NSRD installer window, click “next” to continue Installation.



4. Please wait for the Contacting Math Works to prepare the installation; then you will see the ‘’Installation Options’’ window.



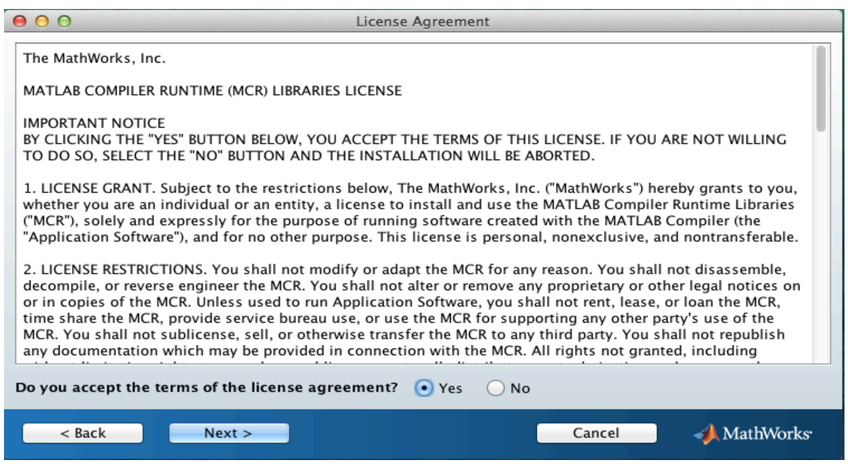
5. Click on the browse button to set a location for the program to be installed.



6. Click “next” to continue installation.

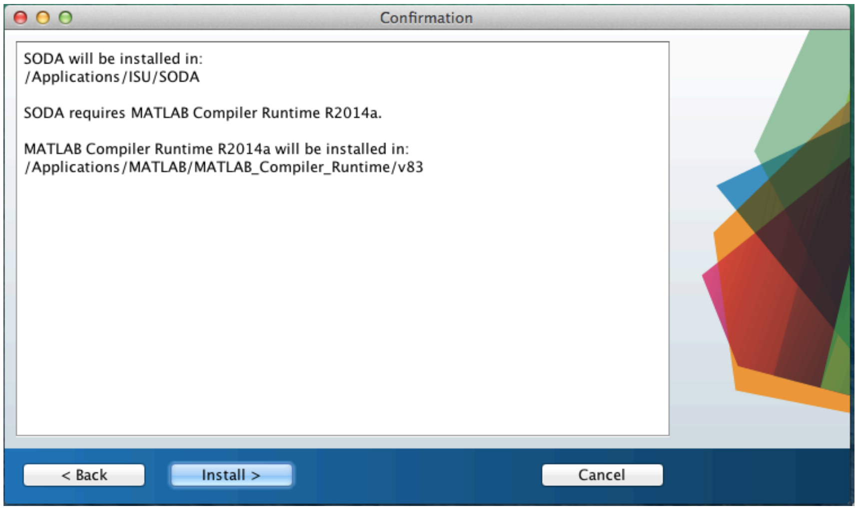
7. ‘’Required software’’ window will show up; click on the browse button to set a location for the program to be Installed.

8. Click next to continue installation.
9. When you see the ‘’License agreement’’ window, click the Yes button to accept the License Agreement. Note: If you check No the program will cancel.

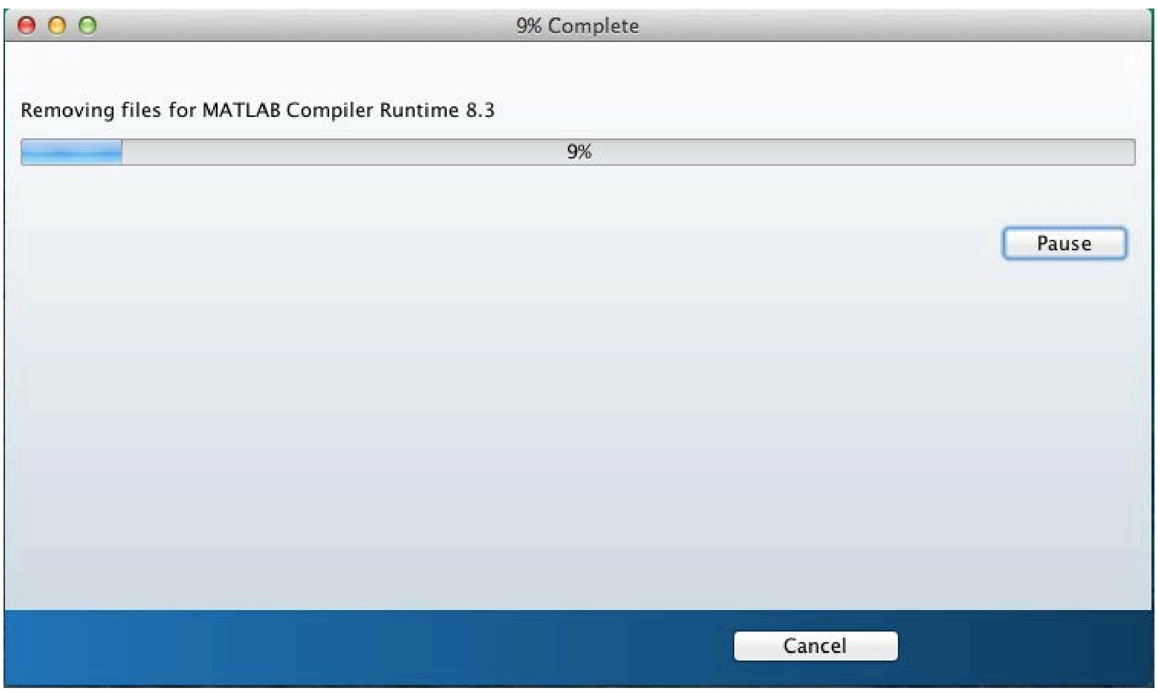


10. Click Next to continue installation.

11. Confirmation window will show up; click ''install'' to start downloading the program.



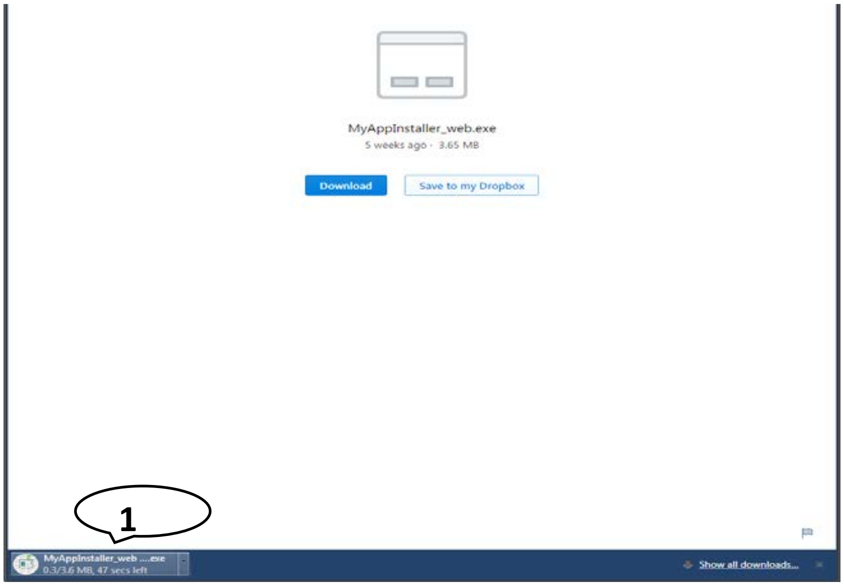
12. The download will keep track of the percentage downloaded 0%-100%. If you need to pause the program for any reason, press the pause button. Once the program reaches 100%, the ''Installation complete'' window will show.



13. Click finish; the program can be found in the location that you chose.

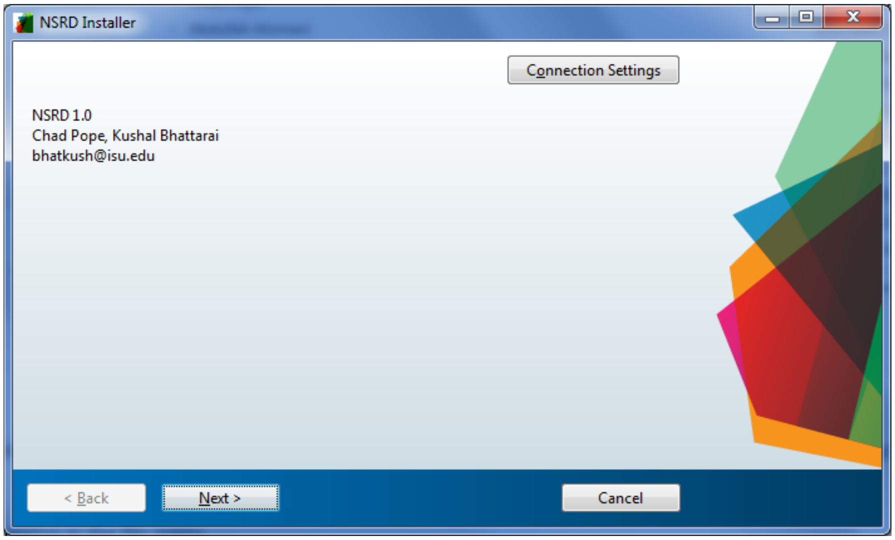
Windows Installation

1. Download the program; the download file will be at the bottom of your Internet browser. Click to open the installation program.

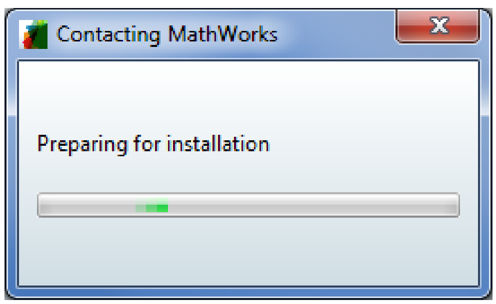


2. The install application will take a few minutes to load. Please just let it run until you see the SODA Installer window.

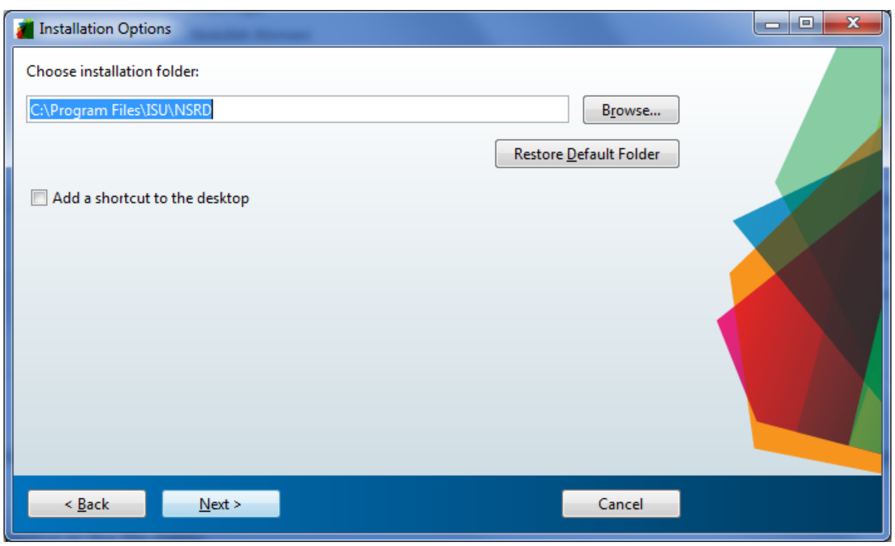
3. Click next button to continue Installation.



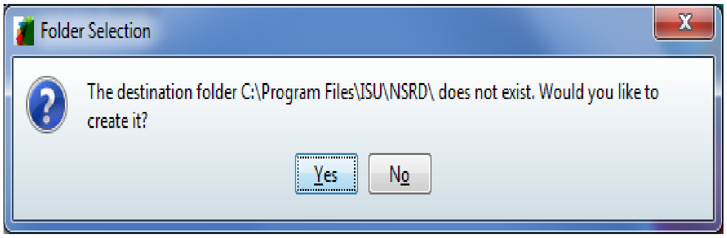
4. Please wait for Contacting Math Works to prepare the installation; then you will see the installation Options window.



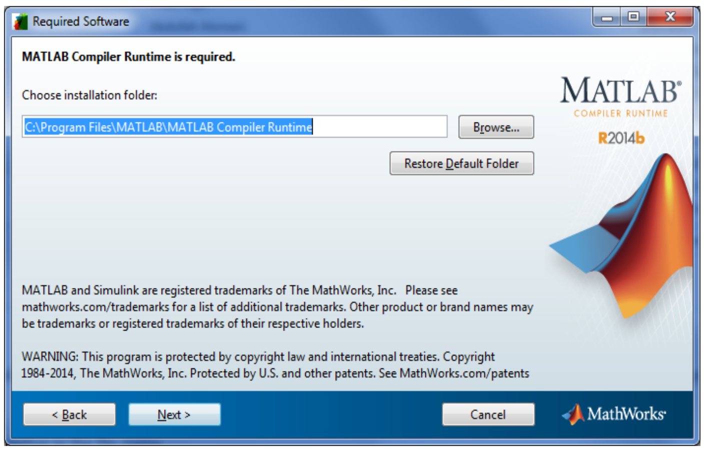
5. Click on the browse button to set a location for the program to be installed.



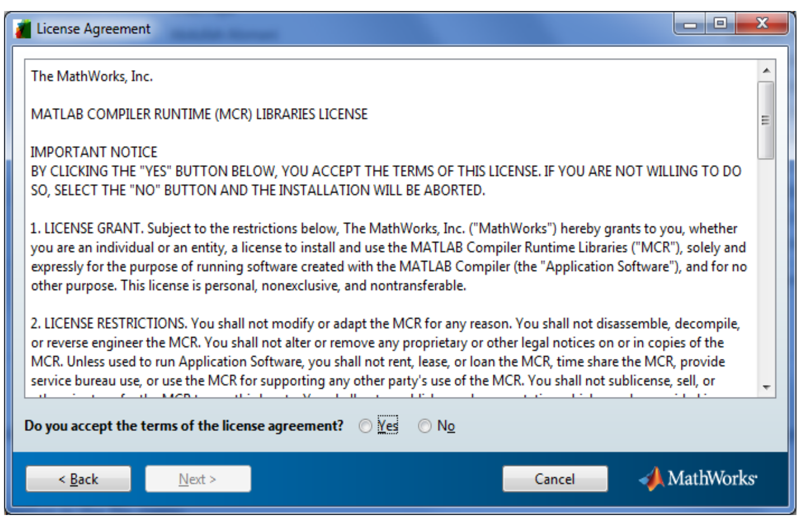
6. Click on the next button to continue installation; then you will see the folder selection.



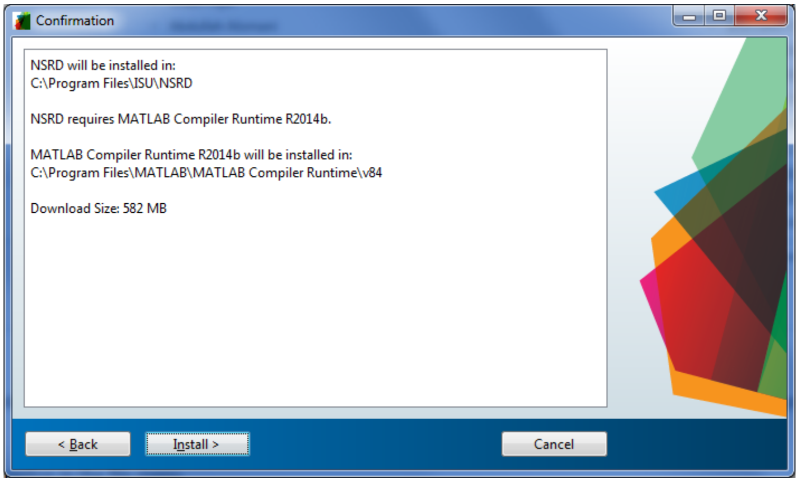
7. Click the Yes or No to either accept or change folders for Installation.
8. Choose an installation folder for the MATLAB compiler runtime.



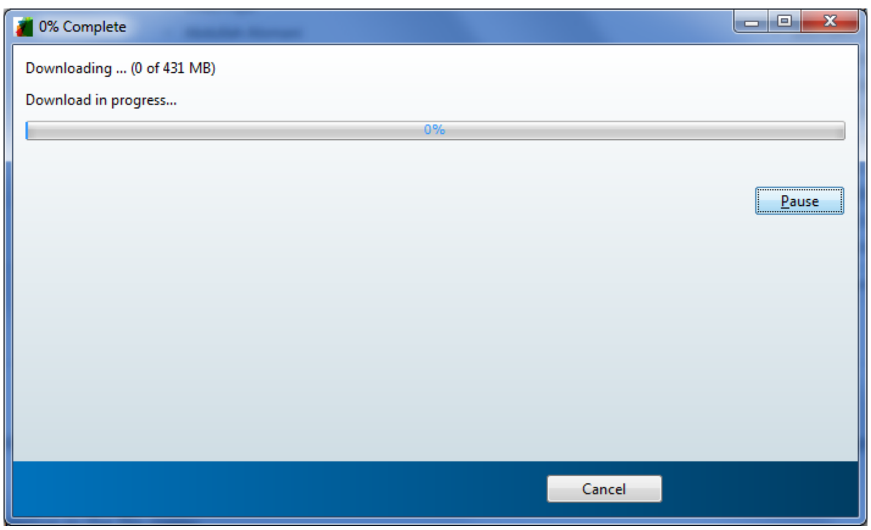
9. Check the Yes or No button to accept the License Agreement. Note: if you check No, the program will cancel.



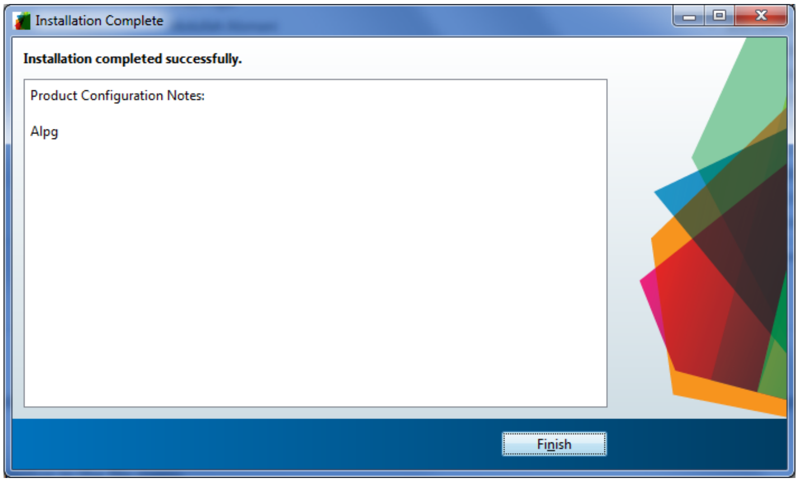
10. Confirmation window will show up; click the install button to start downloading the program.

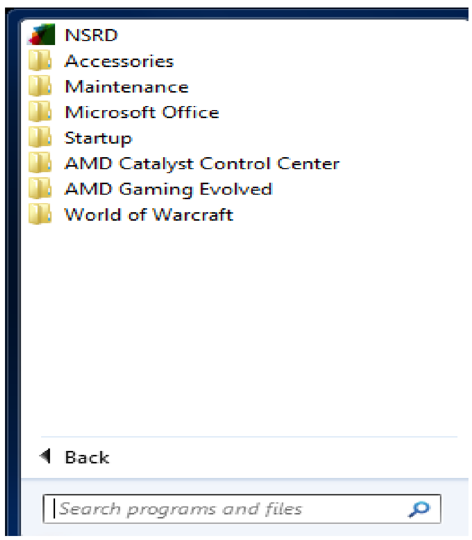


11. The download will keep track of the percentage downloaded 0%-100%. If you need to pause the program for any reason. Press the pause button. Once the program reaches 100% the ''installation complete'' window will show.



12. The program can be found in several places, Start Menu as pictured; desktop if you chose to allow desktop item; or installation folder that you picked in ‘’Installation options’’.





APPENDIX C:
MATLAB Code

```
function varargout = Project1(varargin)
% 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 10-Apr-2015 14:22:56

% Begin initialization code - DO NOT EDIT
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.
```

```
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');

%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_popup_dist,'String',{'Select Distribution';'Normal';...
   'Beta';'Uniform';'Exponential'},'Value',1);
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.
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.

sample = get(handles.num_sample_text,'String');%grab number from number of sample box
samplesize = str2num(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 = str2num(get(handles.rf_text1,'String')); %grab number from number of text box 1
num2 = str2num(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'
      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','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');
      title(['Respirable Fraction distribution plot with Normal Distribution \mu ='...
         num2str(mean(n)) ' and \sigma = ' num2str(std(n))])
   case 'Beta'
      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','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');
    title(['Respirable Fraction distribution plot with Beta distribution with \mu= '...
        num2str(mean(n)) ' and \beta = ' num2str(std(n))])
  case 'Uniform'
    if num1 < num2;
        % In unifrom distribution upper limt must be greater than lower
        % limit, if not show the error message
        errordlg('Upper Limit is less than lower limt','Uniform Distribution','modal')
        set(handles.rf_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','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');
        title(['Respirable Fraction distribution plot with Uniform distribution with \mu= '...
            num2str(mean(n)) ' and \sigma = ' num2str(std(n))])
    end
  case 'Exponential'
    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','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');
    title(['Respirable Fraction distribution plot with Exponential distribution with \mu= '...
      num2str(mean(n)) ' and \sigma = ' num2str(std(n))])
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.
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 stiring the user have selected.
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','')
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.
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)

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','')
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.
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)
sample = get(handles.num_sample_text,'String');
samplesize = str2num(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 = str2num(get(handles.dr_text1,'String'));
num2 = str2num(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'
      pd = makedist('Normal','mu',num1,'sigma',num2);
      t = truncate(pd,0.1,1);
      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');
      title(['DR distribution plot with Normal distribution'...
         '\mu=' num2str(mean(n)) ' and \sigma = ' num2str(std(n))])
   case 'Beta'
      pd = makedist('Beta','a',num1,'b',num2);
      t = truncate(pd,0.1,1);
      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');
    title(['DR distribution plot with Beta distribution'...
      '\mu=' num2str(mean(n)) ' and \sigma = ' num2str(std(n))])
  case 'Uniform'
    if num1 < num2;
      % In unifrom distribution upper limt must be greater than lower
      % limit, if not show the error message
      errordlg('Upper Limit is less than lower limt','Uniform Distribution','modal')
      set(handles.dr_pushbutton,'str','Show Plot','backg',col);
      return;
    else
      pd = makedist('Uniform','Upper',num1,'Lower',num2);
      t = truncate(pd,0.1,1);
      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');
      title(['DR distribution plot with Uniform distribution'...
        '\mu=' num2str(mean(n)) ' and \sigma = ' num2str(std(n))])
    end
  case 'Exponential'
    pd = makedist('Exponential','mu',num1);
    t = truncate(pd,0.1,1);
    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');
    title(['DR distribution plot with Exponential distribution'...
      '\mu=' num2str(mean(n)) 'and \sigma = ' num2str(std(n))])
```

```
end
set(handles.dr_pushbutton,'str','Show Plot','backg',col);
end

% --- Executes on button press in run_pushbutton.
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)
sample = get(handles.num_sample_text,'String');
samplesize = str2num(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 strcmp(sample,'') == 1 || samplesize < 0
      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 = str2num(get(handles.mar_text1,'String'));
   num2 = str2num(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 '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
            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
else
   mar = str2num(get(handles.mar_text1,'String'));
   if mar <= 0;
      errordlg('Material at Risk cannot 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 = str2num(get(handles.dr_text1,'String'));
   num2 = str2num(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 '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
            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);
    end
else
    dr = str2num(get(handles.dr_text1,'String'));
    if dr > 1 || dr <= 0;
        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 = str2num(get(handles.arf_text1,'String'));
    num2 = str2num(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,inf);
            arf = random(t,samplesize,1);
        case 'Beta'
            pd = makedist('Beta','a',num1,'b',num2);
            t = truncate(pd,0,inf);
            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
                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);
                arf = random(t,samplesize,1);
            end
        case 'Exponential'
            pd = makedist('Exponential','mu',num1);
            t = truncate(pd,0,inf);
            arf = random(t,samplesize,1);
    end
else
    arf = str2num(get(handles.arf_text1,'String'));
    if arf <= 0;
        errordlg('Airborne Release Factor cannot be less than 0', 'Error');
        set(handles.run_pushbutton,'str','Show Plot','backg',col);
```

```
        return
    end
end

%compute Respirable fraction

if d == 0 ;
    num1 = str2num(get(handles.rf_text1,'String'));
    num2 = str2num(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,inf);
            rf = random(t,samplesize,1);
        case 'Beta'
            pd = makedist('Beta','a',num1,'b',num2);
            t = truncate(pd,0,inf);
            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
                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);
                rf= random(t,samplesize,1);
            end
        case 'Exponential'
            pd = makedist('Exponential','mu',num1);
            t = truncate(pd,0,inf);
            rf = random(t,samplesize,1);
    end
else
    rf = str2num(get(handles.rf_text1,'String'));
    if rf <= 0;
        errordlg('Respirable Factor cannot be less than or equal 0', 'Error');
        set(handles.run_pushbutton,'str','Show Plot','backg',col);
        return
    end
end

%compute leak path factor

if e == 0;
    num1 = str2num(get(handles.lpf_text1,'String'));
    num2 = str2num(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 '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
                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);

    end
else
    lpf = str2num(get(handles.lpf_text1,'String'));
    if lpf > 1 || lpf <= 0;
        errordlg('Leak Path Factor cannot be less than or equal to 0 or greator than 1', 'Error');
        set(handles.run_pushbutton,'str','Show Plot','backg',col);
        return;
    end
end

%compute source term
st = mar.*dr.*arf.*rf.*lpf;

%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';

else
    br = str2num(get(handles.br_text1,'String'));
    if br <= 0;
        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 = str2num(get(handles.dcf_text1,'String'));
    num2 = str2num(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 '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
                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);
    end
else
    dcf = str2num(get(handles.dcf_text1,'String'));
    if dcf <= 0;
        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 = str2num(get(handles.distance_text1,'String'));
    distance1 = str2num(get(handles.distance_text2,'String'));
    pd = makedist('Normal','mu',0,'sigma',distance1);
    crossdistance = random(pd,samplesize,1);

    num1 = str2num(get(handles.windspeed_text1,'String'));
    num2 = str2num(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','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 = str2num(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.run_pushbutton,'str','Show Plot','backg',col);
                return;
        end
    case 'Select Terrain'
        switch stability
            case 'A'
                errordlg('Select terrain','Error','modal');
                set(handles.run_pushbutton,'str','Show Plot','backg',col);
                return;
            case 'B'
                errordlg('Select terrain','Error','modal')
                set(handles.run_pushbutton,'str','Show Plot','backg',col);
                return;
            case 'C'
                errordlg('Select terrain','Error','modal');
                set(handles.run_pushbutton,'str','Show Plot','backg',col);
                return;
            case 'D'
                errordlg('Select terrain','Error','modal');
                set(handles.run_pushbutton,'str','Show Plot','backg',col);
                return;
            case 'E'
```

```
                errordlg('Select terrain','Error','modal');
                set(handles.run_pushbutton,'str','Show Plot','backg',col);
                return;
            case 'F'
                errordlg('Select terrain','Error','modal');
                set(handles.run_pushbutton,'str','Show Plot','backg',col);
                return;
            case 'Select Stability Condition'
                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'
                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));

else
  cq = str2num(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
st = 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;
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);
   y = std(ced);
   nbins = max(min(length(ced)./10,100),50);
   xi = linspace(min(ced),max(ced),nbins);
   assignin('base','cedxi', xi);
   dx = mean(diff(xi));
   fi = histc(ced,xi-dx);
   fi = fi./sum(fi)./dx;
   assignin('base','cedfi2', fi);
   %     assignin('base','fi2', fi);
   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', [0.6 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);
end

% --- Executes on button press in rf_togglebutton.
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.
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.
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 = str2num(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 = str2num(get(handles.distance_text1,'String'));
distance1 = str2num(get(handles.distance_text2,'String'));
pd = makedist('Normal','mu',0,'sigma',distance1);
crossdistance = random(pd,samplesize,1);

num1 = str2num(get(handles.windspeed_text1,'String'));
num2 = str2num(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 = str2num(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')
title([ '\chi/Q'  ' Distribution with \mu= ' num2str(mean(c)) ' and \sigma= ' num2str(std(c))])
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.
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.
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.
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)
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 'U-238'
        set(handles.dcf_text1,'String','5.0*10^-7');
    case 'Select Isotope'
```

```
        set(handles.dcf_text1,'String','');
    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.
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)
sample = get(handles.num_sample_text,'String');
samplesize = str2num(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 = str2num(get(handles.dcf_text1,'String'));
num2 = str2num(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'
      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','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');
        title(['DCF distribution plot with Normal distribution \mu=' num2str(mean(n))...
            ' and \sigma = ' num2str(std(n))])
    case 'Beta'
        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','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');
        title(['DCF distribution plot with Beta distribution \mu=' num2str(mean(n))...
            ' and \sigma = ' num2str(std(n))])
    case 'Uniform'
        if num1 < num2;
            % In unifrom distribution upper limt must be greater than lower
            % limit, if not show the error message
            errordlg('Upper Limit is less than lower limt','Uniform Distribution','modal')
            set(handles.dcf_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','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');
            title(['DCF distribution plot with Uniform distribution \mu=' num2str(mean(n))...
                ' and \sigma = ' num2str(std(n))])
        end
    case 'Exponential'
        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','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');
        title(['DCF distribution plot with Exponential distribution \mu=' num2str(mean(n))...
            ' and \sigma = ' num2str(std(n))])
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)

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');
    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');  %

else
    set(hObject,'string','Distribution Input');
    set(handles.dcf_popup_dist,'String',{'Select Distribution';'Normal';...
        'Beta';'Uniform';'Exponential'},'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.
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 = str2num(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);
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.
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.
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)


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','')
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.
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 = str2num(get(handles.num_sample_text,'String'));
% samplesize = get(handles.num_sample_text,'String');
% if strcmp(samplesize,'') || num2str(samplesize) < 1
%
sample = get(handles.num_sample_text,'String');
samplesize = str2num(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 = str2num(get(handles.lpf_text1,'String'));
num2 = str2num(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'
        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');
    title(['LPF distribution plot with Normal distribution \mu=' num2str(mean(n))...
        ' and \sigma = ' num2str(std(n))])
  case 'Beta'
    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');
    title(['LPF distribution plot with Beta distribution \mu=' num2str(mean(n))...
        ' and \sigma = ' num2str(std(n))])
  case 'Uniform'
    if num1 < num2;
        % In unifrom distribution upper limt must be greater than lower
        % limit, if not show the error message
        errordlg('Upper Limit is less than lower limt','Uniform Distribution','modal')
        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');
            title(['LPF distribution plot with Uniform distribution \mu=' num2str(mean(n))...
                ' and \sigma = ' num2str(std(n))])
        end
    case 'Exponential'
        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');
        title(['LPF distribution plot with Exponential distribution \mu=' num2str(mean(n))...
            ' and \sigma = ' num2str(std(n))])
end
set(handles.lpf_pushbutton,'str','Show Plot','backg',col);
end


% --- Executes on button press in lpf_togglebutton.
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.
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)
sample = get(handles.num_sample_text,'String');
samplesize = str2num(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 = str2num(get(handles.arf_text1,'String'));
num2 = str2num(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'
        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','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');
```

```
    title(['ARF distribution plot with Normal distribution'...
        '\mu=' num2str(mean(n)) ' and \sigma = ' num2str(std(n))]);
case 'Beta'
    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','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');
    title(['ARF distribution plot with Beta distribution'...
        '\mu=' num2str(mean(n)) ' and \sigma = ' num2str(std(n))])
case 'Uniform'
    if num1 < num2;
        % In unifrom distribution upper limt must be greater than lower
        % limit, if not show the error message
        errordlg('Upper Limit is less than lower limt','Uniform Distribution','modal')
        set(handles.arf_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','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');
        title(['ARF distribution plot with Uniform distribution'...
            '\mu=' num2str(mean(n)) ' and \sigma = ' num2str(std(n))])
    end
case 'Exponential'
    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','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');
        title(['ARF distribution plot with Exponential distribution'...
            '\mu=' num2str(mean(n)) ' and \sigma = ' num2str(std(n))]);
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.
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)

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','')
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.
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 '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.
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 = str2num(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 = str2num(get(handles.mar_text1,'String'));
num2 = str2num(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'
        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', [0.6 1.02], 'HorizontalAlignment', 'center')

    case 'Beta'
        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', [0.55 1.02], 'HorizontalAlignment', 'center')
    case 'Uniform'
        if num1 < num2;
            % In unifrom distribution upper limt must be greater than lower
            % limit, if not show the error message
            errordlg('Upper Limit is less than lower limt','Uniform Distribution','modal')
            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', [0.55 1.02], 'HorizontalAlignment', 'center')
        end
     case 'Exponential'
        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', [0.55 1.02], 'HorizontalAlignment', 'center')
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


% filename = 'C:\Program Files\ISU\SODA\application\help.pdf';
% winopen('C:\Program Files\ISU\SODA\application\help.pdf')
%
% mac system
system('open /Applications/ISU/SODA/application/help.pdf')
% system('open help.pdf' );
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


% ------------------------------------------------------------------
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)
[filename,pathname] = uigetfile('*.mat','Load Work Space');
if isequal(filename,0)
   return
end

load(fullfile(pathname,filename),'userinput');

%load MAR
set(handles.num_sample_text,'String',userinput.num_sample_text);
set(handles.mar_togglebutton,'Value',userinput.mar_togglebutton);
if userinput.mar_togglebutton == 0;
   set(handles.mar_togglebutton,'String','Distribution Input');
   set(handles.mar_popup_dist,'Enable','on','Value',userinput.mar_popup_dist);
   if userinput.mar_popup_dist > 1 && userinput.mar_popup_dist < 5;
      set(handles.mar_text1,'Enable','on','String',userinput.mar_text1);
      set(handles.mar_text2,'Enable','on','String',userinput.mar_text2);
      set(handles.mar_pushbutton,'Enable','on');
   elseif userinput.mar_popup_dist == 5;
      set(handles.mar_text1,'Enable','on','String',userinput.mar_text1);
      set(handles.mar_pushbutton,'Enable','on');
   end
```

```
else
   set(handles.mar_togglebutton,'String','Single Input');
   set(handles.mar_popup_dist,'Enable','off','Value',1)
   set(handles.mar_text1,'Enable','on','String',userinput.mar_text1);
   set(handles.mar_text2,'Enable','off','String','');
   set(handles.mar_pushbutton,'Enable','off');
end
%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;
      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');
   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 ARF
set(handles.arf_togglebutton,'Value',userinput.arf_togglebutton);

if userinput.arf_togglebutton == 0;
   set(handles.arf_togglebutton,'String','Distribution Input');
   set(handles.arf_popup_dist,'Enable','on','Value',userinput.arf_popup_dist);
   if userinput.arf_popup_dist > 1 && userinput.arf_popup_dist < 5
      set(handles.arf_text1,'Enable','on','String',userinput.arf_text1);
      set(handles.arf_text2,'Enable','on','String',userinput.arf_text2);
      set(handles.arf_pushbutton,'Enable','on');
   elseif userinput.arf_popup_dist == 5;
      set(handles.arf_text1,'Enable','on','String',userinput.arf_text1);
      set(handles.arf_pushbutton,'Enable','on');
   end
else
   set(handles.arf_togglebutton,'String','Single Input');
   set(handles.arf_popup_dist,'Enable','off','Value',1)
   set(handles.arf_text2,'Enable','off','String','');
   set(handles.arf_text1,'Enable','on','String',userinput.arf_text1);
   set(handles.arf_pushbutton,'Enable','off');
end
%load RF
```

```
set(handles.rf_togglebutton,'Value',userinput.rf_togglebutton);

if userinput.rf_togglebutton == 0;
   set(handles.rf_togglebutton,'String','Distribution Input');
   set(handles.rf_popup_dist,'Enable','on','Value',userinput.rf_popup_dist);
   if userinput.rf_popup_dist > 1 && userinput.rf_popup_dist< 5
      set(handles.rf_text1,'Enable','on','String',userinput.rf_text1);
      set(handles.rf_text2,'Enable','on','String',userinput.rf_text2);
      set(handles.rf_pushbutton,'Enable','on');
   elseif userinput.rf_popup_dist == 5;
      set(handles.rf_text1,'Enable','on','String',userinput.rf_text1);
      set(handles.rf_pushbutton,'Enable','on');
   end

else
   set(handles.rf_togglebutton,'String','Single Input');
   set(handles.rf_popup_dist,'Enable','off','Value',1)
   set(handles.rf_text2,'Enable','off','String','');
   set(handles.rf_text1,'Enable','on','String',userinput.rf_text1);
   set(handles.rf_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;
      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');
   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 DCF
set(handles.dcf_togglebutton,'Value',userinput.dcf_togglebutton);

if userinput.dcf_togglebutton == 0;
    set(handles.dcf_togglebutton,'String','Distribution Input');
    set(handles.dcf_popup_dist,'Enable','on','Value',userinput.dcf_popup_dist);
    if userinput.dcf_popup_dist > 1 && userinput.dcf_popup_dist < 5;
        set(handles.dcf_text1,'Enable','on','String',userinput.dcf_text1);
        set(handles.dcf_text2,'Enable','on','String',userinput.dcf_text2);
        set(handles.dcf_pushbutton,'Enable','on');
    elseif userinput.dcf_popup_dist == 5;
        set(handles.dcf_text1,'Enable','on','String',userinput.dcf_text1);
        set(handles.dcf_pushbutton,'Enable','on');
    end
else
    set(handles.dcf_togglebutton,'String','Single Input');
    set(handles.dcf_popup_dist,'Enable','on','String',{'Select Isotope';'U-238';'U-239';'Pu-239';'Pu-235'}...
        ,'Value',userinput.dcf_popup_dist);
    set(handles.dcf_text2,'Enable','off','String','');
    set(handles.dcf_text1,'Enable','on','String',userinput.dcf_text1);
    set(handles.dcf_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_dist);
    set(handles.cq_text1,'Enable','off','String','');
    set(handles.height_text,'String',userinput.height_text);
    set(handles.distance_text1,'Enable','on','String',userinput.distance_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
end




% --------------------------------------------------------------------
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)
[filename,pathname] = uiputfile('*.mat','Save Work Spcae as');
if pathname == 0 %if the user pressed cancelled, then we exit this callback
    return
end

userinput.num_sample_text = str2num(get(handles.num_sample_text,'String'));

userinput.mar_togglebutton = get(handles.mar_togglebutton,'Value');
userinput.mar_popup_dist= get(handles.mar_popup_dist,'Value');
userinput.mar_text1 = str2num(get(handles.mar_text1,'String'));
userinput.mar_text2 = str2num(get(handles.mar_text2,'String'));


userinput.dr_togglebutton = get(handles.dr_togglebutton,'Value');
userinput.dr_popup_dist = get(handles.dr_popup_dist,'Value');
userinput.dr_text1 = str2num(get(handles.dr_text1,'String'));
userinput.dr_text2 = str2num(get(handles.dr_text2,'String'));

userinput.arf_togglebutton = get(handles.arf_togglebutton,'Value');
userinput.arf_popup_dist = get(handles.arf_popup_dist,'Value');
userinput.arf_text1 = str2num(get(handles.arf_text1,'String'));
userinput.arf_text2 = str2num(get(handles.arf_text2,'String'));

userinput.rf_togglebutton = get(handles.rf_togglebutton,'Value');
```

```
userinput.rf_popup_dist = get(handles.rf_popup_dist,'Value');
userinput.rf_text1 = str2num(get(handles.rf_text1,'String'));
userinput.rf_text2 = str2num(get(handles.rf_text2,'String'));


userinput.lpf_togglebutton = get(handles.lpf_togglebutton,'Value');
userinput.lpf_popup_dist = get(handles.lpf_popup_dist,'Value');
userinput.lpf_text1 = str2num(get(handles.lpf_text1,'String'));
userinput.lpf_text2 = str2num(get(handles.lpf_text2,'String'));

userinput.br_togglebutton = get(handles.br_togglebutton,'Value');
userinput.br_text1 = str2num(get(handles.br_text1,'String'));


userinput.dcf_togglebutton = get(handles.dcf_togglebutton,'Value');
userinput.dcf_popup_dist = get(handles.dcf_popup_dist,'Value');
userinput.dcf_text1 = str2num(get(handles.dcf_text1,'String'));
userinput.dcf_text2 = str2num(get(handles.dcf_text2,'String'));

userinput.cq_togglebutton = get(handles.cq_togglebutton,'Value');
userinput.distance_text1 = str2num(get(handles.distance_text1,'String'));
userinput.distance_text2 = str2num(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= str2num(get(handles.cq_text1,'String'));

userinput.windspeed_text1= str2num(get(handles.windspeed_text1,'String'));
userinput.windspeed_text2= str2num(get(handles.windspeed_text2,'String'));
userinput.height_text = str2num(get(handles.height_text,'String'));

save(fullfile(pathname,filename),'userinput')
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');
new_axes = copyobj(haxes, ftmp);
set(new_axes,'Units','normalized','Position',[0.1 0.1 0.8 0.8]);
```

```
saveas(ftmp, fullfile(pathname,filename));
delete(ftmp);
end



% ------------------------------------------------------------------
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.figure1);
   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.
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.
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 figure1 is resized.
function figure1_ResizeFcn(hObject, ~, handles)
% hObject    handle to figure1 (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.
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 figure1.
function figure1_CloseRequestFcn(hObject, ~, handles)
% hObject    handle to figure1 (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 figure1_DeleteFcn(hObject, ~, handles)
% hObject    handle to figure1 (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.
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
```

```
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
```

APPENDIX D:
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

3 m drop (pint)	Container Break	
	Yes	No
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

3 m drop (pint)	Container Break	
	Yes	No
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	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