User Manual for the Automated Optimizer for MCNP Input

Contents

Revision 1

Overview 1

Points of Contact 1

System Environment 2

Application Overview 2

Data Input 2

Data Output 3

Run Operation 3

User Required Code Alterations 4

Functions 5

Assumptions and Future Work 7

Revision

Release Version	Date	Description
1.0	14 Dec 2017	Initial Product

Overview

The Automated Optimizer for MCNP input decks, hence forth referred to as the application, optimizes the specified parameters inside an MCNP input deck in an attempt to match experimental data where the main variable is the radioactive source's position. This is accomplished by prioritizing the parameters indicated by the user. Additionally, once the application has determined what the best result is, it outputs a MCNP input deck for the user to use as needed.

The application is a local program that is hands free during its execution allowing the user to accomplish other tasks as memory permits. Since its main function is to optimize MCNP decks, this application is filed as a stand-alone utility. Additionally, it is currently operational for demonstration purposes.

Points of Contact

This application was created under the Air Force Institute of Technology as an academic exercise under the instruction of Captain Bevins. As such, support for this application is minimal to the extent of the Authors.

Author: Robert Torzilli, AFIT

Robert.Torzilli@afit.edu

Co-Author: Lieutenant Bryan Egner, AFIT

bryan.egner@afit.edu

System Environment

Operating System	Required Software	Memory Available
Windows 10	MCNP6	140 MB + Created Data
	Python 3.0 or Newer	

Application Overview

This application takes a variety of provided input files from the user from predefined locations and performs a brute force method of optimizing the primary file of interest, the MCNP input Deck. Once it has optimized the deck, it saves it with actual values in a predefined location. While performing this optimization, the application saves models and data summaries as a visual aid for the user to use to determine if the application is performing as expected.

These locations are outlined below in **User Required Code Alterations.**

Data Input

The following is a list of files the application expects in order for it to execute without error. These file must be provided by the user and will have extra requirements pending what the user is attempting to do.

1. A working input deck, with the interested parameters swapped with unique variable names.
    a. Example: geDensity
2. Experimental data that is being used as the reference the application is trying to duplicate.
    a. First column needs to be interested energy bins
    b. Second column needs to be efficiencies
    c. Third column needs to be uncertainty
3. A Variable Input text file with the following three columns in order.
    a. Interested parameters, matching the unique variable names from 1
    b. Default values
    c. Number of iterations to run between the min and max values, i.e. step
4. An Iteration text file with the following two columns in order.
    a. Interested parameters, matching the unique variable names from 1 followed by min and max
        i. Example: geDensityMin
    b. Value for the parameter

If wanting to run multiple sources or positions, the following must be done:

1. The input deck described above that will be split along its symmetrical data with all of the wanted decks.
    a. The name of this input deck is what will be used in the Code Alterations described below.
2. The non-symmetrical data of the positions each with a different file name

    a. The names of these halves will be part of the Code Alterations described below.

Data Output

The following describes the data output that is recorded by the application in the form of input decks, text files, and PNGs.

1. An MCNP output file for every run is created by the application and placed in the user Defined folder for storage named after the input Deck and concatenated with the parameter it is iterating over. The first run is labeled as the default run and the final run is simply the. name of the input deck. These files are text files and thus can be opened normally.
2. The final optimized deck is plotted and saved as a PNG file to serve as a visual aid. It shows the provided data plotted on the same graph as the analytical data. Currently, the application also saves the analytical data used for the plot so that it can be recreated if need be.
3. A summary of the data for the given energy bins is extracted for ease of comparison. This summary file includes the Average Error, Chi Squared value, and the Relative Error for each bin compared to the experimental data and is saved as a text file.
4. A runnable input deck is saved as a .i file once the application has finished its optimization routine so that the user can run different configurations to determine likely experimental results.

Run Operation

This application can be run as package by simply executing the StartAutomaton batch file; if the user has the aforementioned required software installed on their local drive. Otherwise, this application will need to be manually run by the use of an IDE or inside the command prompt.

Once the application has started it will go through the following process until it has completed running each iteration step, the default run, and the determined optimized run.

The first run, the default run of the code does not need to create input values as it uses the defined ones in the Variable_Input file. The final run has an additional step where it creates a plot and saves the input deck.

1. Create the Input Values

   This application creates its input values by replacing the keyword's value in the application created dictionary with the current iteration value. It accomplishes this by using the editDictionary function.

2. Alter the Model

   In order to alter the model, the editFile function is called whereupon it searches for the parameter keywords in the MCNP input deck and replaces them with the values it has associated in its provided dictionary.

3. Run MCNP

In order to get data, the application makes use of MCNP which it calls by using a batch file. Once the batch file starts, the application waits until MCNP is finished before moving to the next step.

4. Compare Values

   In order to compare the values the application needs to get the data out of the output file that was created by the last step. Once that data is gathered, it can then compare the results to the user provided data. If the resulting chi squared value is less than one, the program exits the current position optimization routine and moves on to running the final optimized run. If the chi squared value does not meet the break out condition, it is compared to the previous chi squared value where upon if the result is better, it stores it for use in the next parameter run.

5. Record, Clean, and Reset

   Before repeating the loop at the next iteration or the next parameter, the data used for comparison as well as the comparison results are saved in text file that denotes which run as well as which parameter the results relate to. In order to save the MCNP output file as a whole the application renames it and places it in its corresponding position folder. Once all the iterations for the current parameter have been completed the application saves the best result as the new one for the other parameters. Hence the application will only use the best known parameter while iterating over values for the interested parameter.

User Required Code Alterations

1. inputEditor.py

   mcnpModel: This variable dictates the name of the input deck that will be ran by the MCNP program. Thus, it must match the name provided to the runMCNP batch file as the input file.

   mcnpBest: This is the name of the final optimized input deck saved for the user.

   variablesToAdjust: This indicates what text file to use an input for the variables that the user has placed inside their edited input deck. As such, if the user wants to use a different file, that must be reflected here.

   valuesToIterateOver: This indicates what text file to use when determining the minimum and maximum range for the application to iterate over. As such if the user wants to change the name of the file that must be reflected here.

   sourceMcnp: This is the numerical value of how many surfaces the user has instructed MCNP to use during the run. Since this value causes the total energy to become a ratio, it is necessary for it to be used when calculating the difference between the MCNP results and the experimental ones. Note, that this is not the same as the amount of energies that the user is actually interested in comparing.

mcnpOut: This is the file name that the application will look for once MCNP has created a new out file. If the user wants to use a different name, the runMCNP batch file must be altered to match so that the file is created with the name and can be found by the application. Note that the extension in this case does not matter as the application will create the extension later during the cleanup process.

dimensionKeys: This is the list of parameters in order of importance that the user wants the program to optimize, iterate over. If the user originally has 10 values and does not wish to alter the text file, the user should be able to simply remove it from this list without breaking the program.

fileNames: This is the list of files the program will attempt to merge when running multiple different source positions. The first value must be the main file that has the same lines across the different positions.

If the user is running a single position this can be ignored as long as the user is using the preexisting inputEditor_for_single_poistion_.py instead of the main application code.

compareValues: This is the name of the text file that will be accessed to determine the values for comparison. It should only have values and those values should be in order as described above in the Data Input section.

currentPositionFolder: This is the name of the directory where the output files will be stored. This folder must exist and it must by inside that parent directory of the application. It is also used to determine the name of the plot PNG that is created.

dataOutLoc: This list is used to determine which directories to store the summary data that is created. Again, these directories must exist and must be within the parent directory of the application.

2. StartAutomaton.bat

   This file should only need to be edited if the user wants to rename the application core file or if the user wants to swap to the single position core file.

3. runMCNP.bat

   The user must place their path to their MCNP bin directory as well as the path to their MCNP data directory. This will allow the batch file to set the path so MCNP can be called as needed.

   Additionally the input and output lines will need to be edited if the user decides to change the names of the output file created or the name of the MCNP input deck the user provides.

Functions

editFile: This function is used to find the parameters indicated by the user and replace them with a numerical value, either the default provided or one of the iterations.

mergeFile: This function merges the files where the first input parameter of the function is the top of the file and the second parameter of the function corresponds to the bottom half. Once created, it is saved as the indicated file that was the third parameter it accepts.

createFile: This function is used to record the data collected while the application is running and saving it in a readable text file. It requires a dictionary of the collected data and the values of each calculated error in order to be used as intended.

createPlotFile: This function is used to create a numpy readable text file that will be passed to the plot module in order to plot the MCNP analytical values.

createDictionary: Takes in a text file and uses the first column as the keyword and the second column as the value associated with that keyword. It is delimited by a space.

getThirdCol: This was created to supplement the createDictionary function without introducing additional error. It does the same thing but instead looks for the third column to associate with the keyword. This can be merged in the future with the createDictionary function.

getData: This function searches for the interested values by using a dictionary keyword. Once the keyword is found it strips the line to get the results needed for comparison. This function is currently very specific to the energy bins of the MCNP out file and should be used with caution if used on a different file with a potentially different format.

relativeErr: This function calculates the relative error using the following equation,

$$E = \frac{|De - Da|}{|De|}$$

Where E is the relative error of each pair of points, Da is the analytical derived point created by the application, and De is the captured experimental data that the application is trying to match.

chiSquared: This function calculates the relative error using the following equation,

$$\mathrm{X}^2 = \sum_{i=0}^{N} \frac{(Da - De)^2}{\sigma^2}$$

Where X^2 ,chi squared, is equal to the sum over all the data points of interest, Da is the analytical derived point created by the application, De is the captured experimental data that the application is trying to match, and σ is the uncertainty of that experimental data. The closer chi squared is to one, the better fit it is to the experimental data.

recreateDimFile: This function is currently not used since the editDictionary is used. It would edit a given variable file with new values for the interested parameter. Thus, if the user wanted to know the given values used in the input deck, this file could save them in human readable format without having to open the MCNP output file.

editDictionary: This function takes a supplied dictionary and replaces the key value for a given keyword.

Assumptions and Future Work

1. The final run is not necessarily the best run since parameters could have correlation and that the application does not currently take this into account since it relies on the user to determine when dictating what parameters to change and in what order.
2. This application does not currently employ the use of standard file safety on every single file it reads. Thus, if a file does not exist, the application will crash. Additionally, if the application encounters a file that already exists when attempting to rename an older file, it will also crash.
3. This application assumes the user is only asking it to iterate over possible ranges that won't break the MCNP geometry.
4. This application currently has functions that could be merged to simplify the code and minimize the SLOC.