MiSim Technical (Developer) Guide

Author: Santhosh Kumar Kasa &

Sumit Bafna

# Introduction:

MiSim is a Simulation tool written with Python. It is interfaced to Microsoft Excel as an external add-in. It exposes custom functions written in Python as Excel function. The user can model the simulation problem using these custom functions in the excel spreadsheet and then simulate it for a number of times.

This document aims to delve into the technical details of the design of this tool.

There are primarily two components to the tool:

1. The back-end which has python functions exposed to excel
2. The UI, which is displayed.

The user models the problem in excel using the MiSim Functions. User can also simulate for a set of inputs. User then enters a number of iterations for each simulation. MiSim simulates it and displays the output on UI. The calculations are done on the excel spread sheet itself.

# Backend:

Pyxll is the gateway between python and excel. More details about pyxll can be found [here](https://www.pyxll.com/). The pyxll provides an config file which can be used to set the pythonpath and log file path. The python path should contain the external python libraries and the python script which contains the functions.

Once, the pyxll is configured as an add-in on excel, functions can be written in the python script, and they appear as excel functions. The signature of the function can be specified in the script. More technical details can be found in the pyxll documentation.

Below are the stochastic functions which MiSim supports in version 1.0

* Binomial Function
* Uniform Function
* Exponential Function
* ChiSquare Function
* Triangular Function
* Log Normal Function
* Normal Function
* Poisson Function

All the functions are facilitated using the NUMPY module of python.

The excel sheet calculation mode can be set to Manual Mode by xlcCalcution function of the Pyxll.

MisOutput – is the function which is responsible for aggregating the output into a numpy array.

MisInputArray and MisInput require to access the excel spreadsheet cells. Pyxll enables excel application access using the win32com module.

For the user specified number of iterations the program refreshes the excel sheet using the xlcCalculateDocument() from pyxll and it collects the output in a numpy array.

Pyxll also enables controlling the excel spreadsheet VBA functions the same way as if it were written in VBA using the Win32com module. Using this feature the screen update is turned off during the calculation and turned on back after the completion of the calculation.

Once the calculation is complete, the output is collected in the numpy array. This could be two dimensional array if the user specified the simulation for a set of inputs.

UI is written in another python script – UI.py. But the python interpreter is being run inside the excel application. This will pause (hold) the excel application from user interaction until the UI is running. So, in order to avoid this we need to trigger the UI in a separate process running python.exe. Multithreading module of the python helps us achieve this.

Multithreading – ‘SetExecutable’ will allow the developer to specify the executable (.exe) used for executing the script.

‘Process’- can be used to specify the target script which has to be executed.

# UI:

The UI is designed using the QT framework of Python (PyQt4). The plots are plotted using the Matplotlib module.

The UI can be briefly divided into 3 sections:

1. The actual bar chart Plot
2. The user controlled settings for the plot
3. The statistics

**Bar chart**: this plots the data for a single input. Maplotlib Hist function is used for plotting the histograms. The number of bins by default are 10 and can be changed.

**User Settings**: this has various parameters such as Number of bins, Probability distribution function, and Cumulative probability distribution function. Based on the user entry, these values can be passed as parameters to the matplotlib hist function.

In order to calculate the percentage of values between the user entered limits, we divide the entire x values into 3 bins i.e. [-lowest x value, user entered lower limit] , [user entered lower limit, user entered higher limit] & [user entered higher limit, highest x value]

We then call NUMPY.HISTOGRAM with the above bins and that returns the number of values present in user entered range.

**Statistics**: The statistics which are shown are as below:

Maximum – using MAX(list)

Minimum – Using Min(list)

Average - Using Average(list)

Median – Using Numpy.median(nump.array)

Standard deviation – Using Numpy.std(numpy.array)

Variance – Using Numpy.var(numpy.array)

Skewness – Using Scipy.stats.skew(numpy.array)

Kurtosis – Using scipy.stats.kurtosis(numpy.array)

These statistics are present for all the various inputs which the user has entered using MisInputArray and MisInput

The EXE is bundled using Winpython and Inno (software to create exe files). Upon installation the Pyxll config file is the only file that needs manual change.

**Scenarios that require code change:**

Before trying to do anything, try to understand the “Simulate” method in the Main.py

1. **Adding new stochastic Function**: In this case, refer the code for existing functions such as normal, binomial in Main.py and add the new stochastic function.

2. **Adding a new statistical function on UI**: the create\_main\_frame method is responsible for creating the main frame. Add the code using numpy, scipy in built functions to calculate the statistic.

3. **Adding multiple output functions**: This could be a potential new addition to the existing functionality. Currently the tool can do multiple simulations with help of MiInputArray and MiInput.

Understand that completely, how different set of simulations are stored in a two dimensional output array where each row corresponds to simulation of one input value.

The MiOutput can currently take just one cell as an input variable. However, it needs to be modified to take cell and the output number (such as 1st output, 2nd output etc.)

Now, the two dimensional output array which is collecting all the output values will not change. Each row corresponds to one output number.

Carefully modify the simulate method so that each time the sheet is refreshed, the mioutput function appends the value to the corresponding output number row.

4. **Optimization each simulation**: In this scenario, we need to solve each instance of the sheet with the Solver. For this, we need to find out if there is a [Solver API](https://msdn.microsoft.com/en-us/library/ff628587%28v=vs.93%29.aspx) that can be accessed through Python.

Once the API/Interface is found out, modify / create another simulate method that will optimize each simulation.