
FOQUS Documentation

Release 3.22.dev0

CCSI team

Apr 10, 2024

CONTENTS

1	FOQUS Installation and Running	1
2	Introduction	9
3	Flowsheets and Settings	13
4	Optimization	37
5	Uncertainty Quantification (UQ)	51
6	Optimization Under Uncertainty (OUU)	117
7	Surrogate Modeling	125
8	Sequential Design of Experiments (SDOE)	151
9	Robust Optimality-Based Design of Experiments (ODoE)	251
10	Heat Integration	281
11	PYOMO-FOQUS	285
12	IDAES-FOQUS	287
13	FOQUS-MATLAB	293
14	Simulation Standard Interface (SimSinter)	307
15	Surrogate Model Based Optimizer	359
16	Debugging	375
17	Developer Documentation	377
18	Vector Variables Support Capability	381
19	References	401
20	Contact and Support	403
21	Copyright and License	405
22	FOQUS	407

FOQUS INSTALLATION AND RUNNING

This chapter covers how to install and run FOQUS as well as how to install other optional components for use within FOQUS.

1.1 Quick Start

Note: If you are installing on Apple silicon please use the sub-sections as this quick start will not work.

For those familiar with the details, here is a summary of how to install and run FOQUS:

- Download and install [Anaconda](#).
- In a terminal, to setup and install:

```
conda create --name ccsi-foqus -c conda-forge python=3.10 pywin32=306
conda activate ccsi-foqus
pip install ccsi-foqus
foqus --make-shortcut # Create Desktop shortcut (Windows only)
```

-
- In a terminal, to run:

```
conda activate ccsi-foqus
foqus
```

- On Windows, double-click the Desktop shortcut made above.

For a detailed explanations see the following sub-sections.

1.2 Contents

1.2.1 Install Python

Python version 3.8 up through 3.12 is required to run FOQUS.

We recommend using either the [Miniconda](#) or [Anaconda](#) Python distribution and package management system. The choice of Miniconda or Anaconda is up to the user, with Miniconda being smaller and quicker to download while Anaconda is larger but more self-contained. For Windows users, Anaconda is likely a better choice as it also comes with the “Anaconda Prompt” which is a command terminal already set up for working with Anaconda. The primary

advantage of using Miniconda or Anaconda is being able to isolate and customize a python environment specifically for FOQUS without having to modify your existing system python environment. It does this by allowing the ordinary user the ability to create self-contained python environments without any need for administrator privileges. These separate environments can have different set of packages, isolating version dependencies when working with multiple python projects.

If you have a working version of Python 3.8 through 3.12, which you prefer over Anaconda, you can skip these steps.

Anaconda or Miniconda Install and Setup

1. Download one of [Miniconda](#) or [Anaconda](#).
2. Install the above package following the [install instructions](#) for your operating system.
3. Create a `ccsi-foqus` conda environment; this environment will be referred to as `ccsi-foqus` in the installation documentation, but you can use any name you like. If you would like to install multiple version of FOQUS (for example a stable version and the latest development version), this can be done by running the following command multiple times with different environment names after the `--name` flag in the below command. In a terminal (or on Windows in the Anaconda Prompt) type:

```
conda create --name ccsi-foqus -c conda-forge python=3.12 pywin32=306
```

Then follow the prompts. This will create a new conda environment with a minimal set of packages. To use a different version of python, change the version specified after `python=` in the command.

Note: The command above installs the `pywin32` Conda package immediately after creating the Conda environment. The `pywin32` package is strictly required to run FOQUS on Windows, and should be installed with Conda from the `conda-forge` channel or errors might occur. For other platforms (Linux, macOS), the `pywin32` package is not required. However, the `pywin32` package itself is still available, and therefore the command above is compatible with all platforms for which FOQUS is supported.

4. Activate the environment on Linux in a terminal type:

```
conda activate ccsi-foqus
```

If you create an environment in which to install FOQUS, you will need to ensure that environment is active before installing FOQUS. On Windows, once FOQUS is installed a batch file is created that will activate the proper environment when running FOQUS. On Linux or Mac, you will need to activate the appropriate environment before running FOQUS.

1.2.2 Install FOQUS

Note: In previous releases we instructed you to download the FOQUS code and install it in place. As of version 1.5.0, this is no longer required unless you are running on Apple silicon. The below `pip install` method is now the preferred method to install FOQUS.

To install FOQUS, open the Anaconda prompt (or appropriate terminal or shell depending on operating system and choice of Python), and run the following commands:

```
conda activate ccsi-foqus
pip install ccsi-foqus
foqus --make-shortcut # Windows only
```

This will install FOQUS and all the required packages into the `ccsi-foqus` conda environment. The last command there will create a Desktop shortcut for easier, non-terminal, startup of FOQUS (Windows only, for now).

1.2.3 For Apple silicon

Open the Anaconda prompt and run the following commands:

```
conda activate ccsi-foqus
conda install pyqt
git clone https://github.com/CCSI-Toolset/FOQUS
cd FOQUS
pip install -e .
```

1.2.4 Install FOQUS Examples

Note: In previous releases the examples were packaged inside the main `ccsi-foqus` package. Since version 3.5.0, they are no longer part of the `ccsi-foqus` package, but instead distributed as a separate archive. The steps described below are now the correct method to install the FOQUS examples.

To obtain the FOQUS examples, go to the “Releases” section of the FOQUS code repository on GitHub at <https://github.com/CCSI-Toolset/FOQUS/releases>, and locate the release of interest based on the version number.

Then, expand the “Assets” section. The examples are packaged in a ZIP archive named `ccsi-foqus-X.Y.Z-examples.zip`, where `X.Y.Z` is the FOQUS version number.

Finally, download the archive containing the example, and extract it to a directory of your choice on your system. Throughout the rest of this documentation, we will refer to this directory as the **Examples directory**.

1.2.5 Run FOQUS

The specific command to launch FOQUS depends on the operating system.

To launch FOQUS, open the Anaconda prompt (or appropriate terminal or shell depending on operating system and choice of Python), and run the following commands:

```
conda activate ccsi-foqus
foqus
```

Alternatively on Windows you can start FOQUS by double-clicking on the “ccsi-foqus” Desktop shortcut created when FOQUS was first installed. That shortcut can be recreated at any time by opening a terminal, as described above, and starting FOQUS with the “make shortcut” option:

```
foqus --make-shortcut
```

Note: The first time FOQUS is run, it will ask for a working directory location. This is the location FOQUS will put any working files. This setting can be changed later.

Note: Files passed as command line arguments to FOQUS will be relative to where FOQUS is run. Once FOQUS starts, file paths will be relative to the FOQUS working directory.

Running FOQUS without a graphical interface (“batch” or “headless” mode)

The default usage mode for FOQUS is through its graphical interface, or GUI. However, it’s still possible to use FOQUS in situations where a graphical interface is not available and/or practical, such as batch computing (e.g. in an HPC cluster) or automation (e.g. within a script).

To enable this mode, set the QT_QPA_PLATFORM environment variable to one of the supported values *before* starting FOQUS, e.g., for the Bash shell:

```
export QT_QPA_PLATFORM=minimal
```

Note: If FOQUS is not configured to enable batch/headless mode as described above, the following error messages might occur when starting FOQUS:

```
PyQt5 or Qt not available
```

or:

```
qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "" even though it was
↳ found.
This application failed to start because no Qt platform plugin could be initialized.
↳ Reinstalling the application may fix this problem.

Available platform plugins are: eglfs, linuxfb, minimal, minimalegl, offscreen, vnc,
↳ wayland-egl, wayland, wayland-xcomposite-egl, wayland-xcomposite-glx, webgl, xcb.
```

If, on the contrary, a graphical interface *is* desired but the errors above occur, it is possible that the system is not yet configured to support graphical applications. In this case, try installing the `libgl1-mesa-glx` and/or `libxkbcommon-x11-0` packages using the package manager appropriate for your Linux distribution (i.e. `apt-get install` on Ubuntu).

1.2.6 Install Optional Software

There are several optional pieces of software which are not written in Python and not easily installed automatically. There are a couple packages which most users would want to install. The first is PSUADE, which provides FOQUS UQ functionality. The second is TurbineLite which requires Windows and is used to interface with Excel, Aspen, and gPROMS software.

Other software listed below will enable additional features of FOQUS if available.

Install PSUADE-Lite (current version: 1.9.0)

PSUADE (Problem Solving environment for Uncertainty Analysis and Design Exploration) is a software toolkit containing a rich set of tools for performing uncertainty analysis, global sensitivity analysis, design optimization, model calibration, and more.

PSUADE-Lite is now available as a Conda package. To install just follow the steps below:

```
conda activate ccsi-foqus
conda install --yes -c conda-forge -c CCSI-Toolset psuade-lite=1.9
psuade --help # quickly test that the psuade executable has been installed correctly
```

The psuade executable should now be available within the Conda environment's folders, i.e. at the path \$CONDA_PREFIX/bin/psuade (Linux, macOS) or %CONDA_PREFIX%\bin\psuade.exe (Windows). Once you set the full path in the corresponding field in the FOQUS GUI "Settings" tab, you should be able to use it normally within FOQUS.

Install Turbine and SimSinter (Windows Only)

Note: You will need to install Turbine and SimSinter in order to run FOQUS locally with Aspen and Excel. The Turbine installation will install the Turbine Web API Windows Service and applications that manage Aspen and Excel processes. The SimSinter installation will install the SinterConfigGUI application and libraries for interacting with Aspen and Excel through COM Server interfaces. The programs "SimSinter" and "TurbineLite" should appear on the local installed programs list.

- Install [Microsoft SQL Server Compact 4.0](#).
- Download and install the latest releases of [SimSinter](#) and [TurbineLite](#).
- Install SimSinter first, then TurbineLite.
- After the install the Turbine Web API Service Will start automatically when Windows starts, but it will not start directly after the install. Do one of these two things (only after install):
 - Restart computer, or
 - Start the "Turbine Web API service":
 1. open Task Manager
 2. go to the "Services" tab
 3. click the "Services" button (in the lower right corner)
 4. right-click "Turbine Web API Service" from the list, and
 5. click "Start"
- Configure the location of the executables in FOQUS
 - SimSinter
 1. In "Settings" (see Figure 1 below) go to the "FOQUS" tab
 2. Modify the "SimSinter Home" field to point to the directory you installed SimSinter
 - TurbineLite
 1. In "Settings" (see Figure 1 below) go to the "Turbine" tab

2. In the “TurbineLite (local)” section modify the “TurbineLite Home” field to point to the directory you installed TurbineLite

Install ALAMO

ALAMO (Automated Learning of Algebraic Models for Optimization) is a software toolkit that generates algebraic models of simulations, experiments, or other black-box systems. For more information, go to the [ALAMO Home Page](#). Download ALAMO and request a license from the [ALAMO download page](#).

Install NLOpt

NLOpt is an optional optimization library, which can be used by FOQUS. NLOpt is not available to be installed with pip, but can be installed with conda as follows:

```
conda activate ccsi-foqus
conda install -c conda-forge nlopt
```

For more information, see the [NLOpt Installation Instructions](#).

Install SnobFit

SnobFit is an optional optimization library, which can be used by FOQUS for unconstrained optimization. The python package can be installed with pip with:

```
conda activate ccsi-foqus
pip install SQSnobFit
```

The plugin has been developed for FOQUS versions 2.1 and greater. For further details on the available versions and installation, see the [SQSnobFit PyPI package page](#).

Once the python package is downloaded, navigate the path to “SQSnobFit” folder (likely `$CONDA_PREFIX/lib/python3.7/site-packages/SQSnobFit/`) and modify the `_snobfit.py` file making the following changes:

Comment out or remove the following code lines just below `def minimize(...)` function definition:

```
if budget <= 0:
    budget = 100000
```

Then replace:

```
return Result(fbest, xbest), objfunc.get_history()
```

with:

```
return (request, xbest, fbest)
```

in the `def minimize()` function.

Install R

R is a software toolbox for statistical computing and graphics. R version 3.1+ is required for the ACOSSO and BSS-ANOVA surrogate models and the Basic Data's SolventFit model.

- Follow instructions from the [R website](#) to download and install R.
- Open R and type the following to install and load the prerequisite packages:

```
install.packages('quadprog')
library(quadprog)
install.packages('abind')
library(abind)
install.packages('MCMCpack')
library(MCMCpack)
install.packages('MASS')
library(MASS)
q()
```

- The last command exits R. When asked to save workspace image, type “y”.
- Open FOQUS, go to the “Settings” tab, and set the “RScript Path” to the proper location of the R executable.

1.2.7 The FOQUS “Settings” Tab

Use the FOQUS “Settings” tab to set the optional software configuration described in this section:

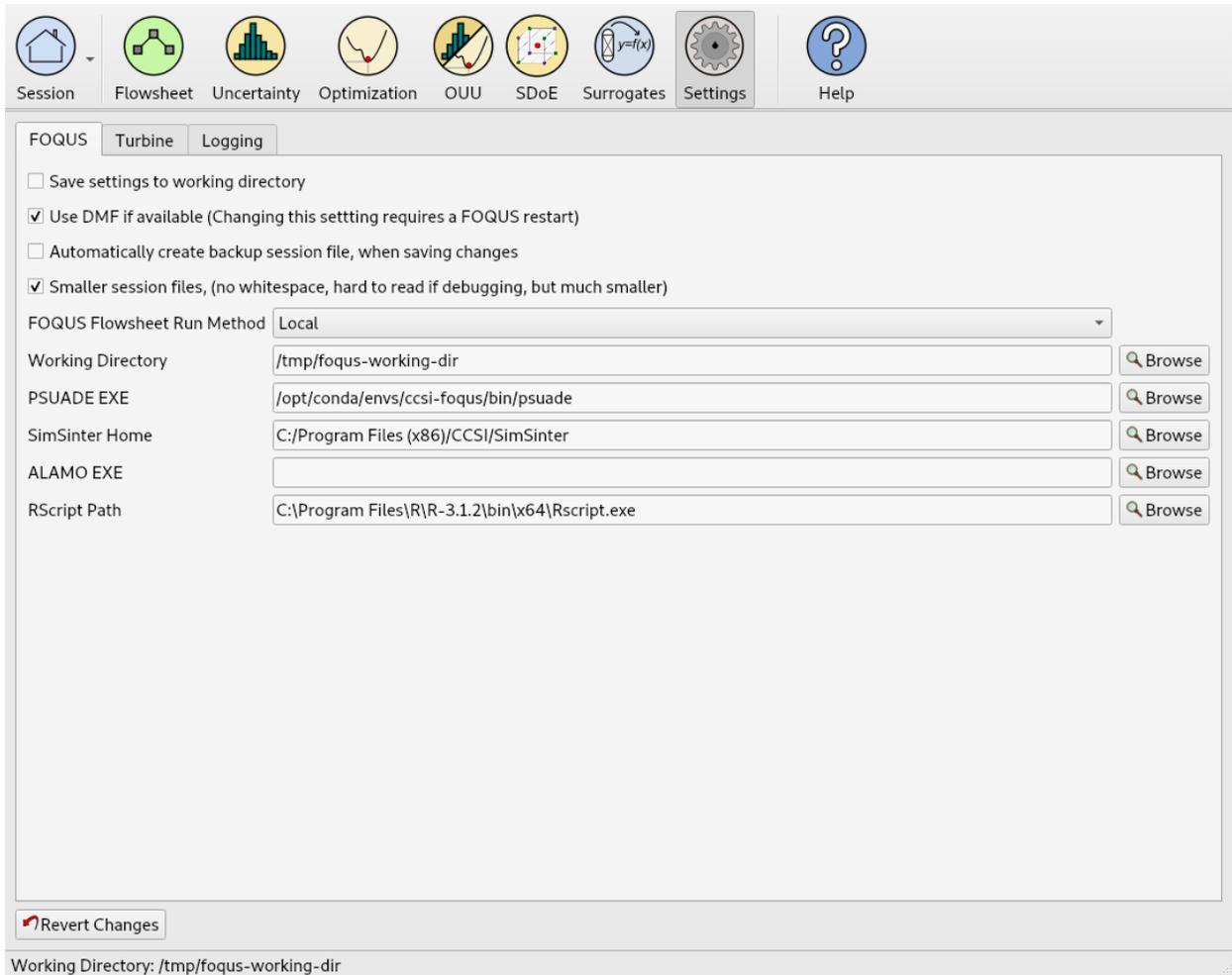


Fig. 1: Figure 1: The FOQUS “Settings” tab

INTRODUCTION

The Framework for Optimization, Quantification of Uncertainty, and Surrogates (FOQUS) software provides a graphical interface and standard platform for several Carbon Capture Simulation Initiative (CCSI) tools. The primary feature of FOQUS is its ability to interact with commonly-used chemical engineering process modeling software. Models constructed using a variety of software can be combined into a larger composite model. The CCSI tool SimSinter provides connectivity to external process simulation software. SimSinter also provides a standard library to enable interfacing with other software. The CCSI FOQUS Cloud on Amazon Web Services (AWS) is fully compatible with the FOQUS UI, and can be used to submit flowsheets with Aspen ACM/AspenPlus simulations to be run remotely and in large batch submissions to take advantage of high parallelism enabled through cloud architecture including on-demand Cloud resource scaling.

In FOQUS, simulations can be connected in a meta-flowsheet, which enables parts of a process to be modeled using the most appropriate software and combines them into a single large model, possibly including recycle streams. For example, in studying a carbon capture system for a coal-fired power plant: a power plant may be modeled in Thermoflex; a solvent-based carbon capture system may be modeled in Aspen Plus; and a compression system may be modeled in gPROMS. To optimize the entire system, these models can be combined into a single large model. The resulting meta-flowsheet can be used for simulation-based optimization, uncertainty quantification (UQ), or generation of surrogate models.

This section provides brief overview and motivating examples, for different uses of FOQUS.

2.1 Simulation Based Optimization

Simulation-based optimization considers a process simulation to be a black box model, which is a model where the mathematical details are not known. In this case, models are evaluated using process simulation software; multiple models can be combined to form larger models. Due to the long run times and the limitations of the methods used, a limited set of optimization variables (usually less than 30) is considered. Simulation-based optimization has some advantages and disadvantages, compared to equation-based optimization methods. With simulation-based optimization, there is no need to provide simplified algebraic models, problem formulation is relatively simple, and a good solution can usually be obtained; however, a provably-global optimum cannot be found and it is impractical to deal with very large numbers of variables. Large numbers of variables may be found in superstructure and heat integration problems where the structure of a process is being optimized. Both simulation and equation-based optimization methods are used in CCSI.

Capture of CO₂ from a pulverized coal-fired power plant involves several very different systems including: a boiler, steam cycle, flue gas desulfurization, carbon capture, and CO₂ compression. It is convenient to separate many of these processes into smaller, more reliable simulations. The different processes may also be better simulated in different software packages. Although some process simulation software contains optimization features, there are several reasons these may not be practical for a large composite system. It may be hard to develop a large model of the entire system that reliably converges. Many optimization methods have a difficult time dealing with simulation errors, and many black box derivative free optimization solvers are better able to handle occasional simulation failures. It may not be

practical to simulate the entire process accurately using a single tool. Derivatives are also difficult to estimate for many systems when models do not provide exact derivatives, making derivative-free methods a good option.

The motivating example used to demonstrate the optimization framework is fairly simple. The system consists of a series of bubbling fluidized bed (BFB) CO₂ adsorbers and regenerators modeled in Aspen Custom Modeler (ACM). The details of the BFB system are described in the CCSI BFB model documentation. A cost analysis for a 650 MW power plant and capture system is presented in an Excel spreadsheet. The simulation and spreadsheet files are provided in the examples directory in the FOQUS installation directory (see the tutorial in Section [ref{tutorial.sim.flowsheet}](#) for more information). The spreadsheet contains capital cost as well as operating and maintenance cost estimates, which are used to estimate the cost of electricity.

In this example, the objective function is the cost of electricity; the decision variables are design and operating variables in the ACM model. The cost of electricity is minimized while maintaining a 90 CO₂ percent capture rate. The BFB system model and the cost of electricity are contained in separate models connected in a FOQUS flowsheet, which enables the cost of electricity to be calculated in Excel, using data acquired from the ACM model. See Sections [ref{tutorial.sim.flowsheet}](#) and [ref{sec.opt.tutorial}](#) for more information about the optimization problem.

2.2 Uncertainty Quantification

The Uncertainty Quantification (UQ) module of FOQUS encompasses a rich selection of mathematical, statistical, and diagnostic tools for application users to perform UQ studies on their simulation models. The PSUADE tool provides most of the UQ functionality available in FOQUS (*Tong 2011*). The recommended systematic multi-step approach consists of the following steps:

1. Define the objectives of the analysis (e.g., identify the most important sources of uncertainties).
2. Specify a simulation model to be studied. Acquire the model input files and the executable that runs the simulation (i.e., an executable that uses the specified inputs and generates model outputs). Identify the outputs of interest, identify all relevant sources of uncertainties, and ensure that they can be used as input variables to the simulation model.
3. Select some or all input parameters that have uncertainty attributed. Characterize the prior probability distribution of these selected parameters by specifying the upper/lower bounds. For non-uniform prior distributions (e.g., Gaussian), additional information (e.g., mean and standard deviation) is required to define the shape of the prior distribution. This prior distribution represents the user's best initial guess about the selected parameters' uncertainties.
4. Identify, if available, relevant data from physical experiments that can be used for model parameter calibration. Model calibration is a process that applies the observational data to update the prior distribution. The model calibration correlates the observational data to predict a distribution as a result.
5. Select a sample scheme and sample size. From this information, a set of input values are sampled from the prior distribution. The choice of sampling scheme (which affects how the samples populate the input space) depends on the UQ objective(s) specified in the first step.
6. "Run" the input samples. Running the input samples is the process where each sampled input value is fed to the simulation executable (specified in Step 2) and the corresponding output value is returned.
7. Analyze the results and make decisions on how to proceed.

Steps 1-4 are often done through expert knowledge elicitation and/or literature search. Steps 5-7 can be achieved through software provided in the FOQUS UQ module.

The FOQUS UQ module provides a number of sampling and analysis methods, including:

- Parameter screening methods: computes the importance of input parameters to identify which are important (to be kept in subsequent analyses) and which to ignore (to be weeded out).

- Response surface (used interchangeably with ‘surrogate’) construction: approximates the relationship between the input samples and their outputs via a smooth mathematical function. This response surface or surrogate can then be used in place of the actual simulation model to speed up lengthy simulations.
- Response surface validation methods: evaluates how well a given response surface fits the data. This is important for choosing different response surfaces.
- Basic uncertainty analysis: propagates input uncertainty to output uncertainty.
- Sensitivity analysis methods: quantifies how much varying an input value can impact the resulting output value.
- Bayesian calibration: applies observational data to refine the estimate of input uncertainties.
- Visualization tools: views computed distributions and response surfaces.
- Diagnostics tools (mainly in the form of scatter plots): checks samples and model behaviors (e.g., outliers).

The adsorber 650.1 subsystem process model is used to demonstrate the UQ framework. The A650.1 process model was developed and is continuously refined by our Process Synthesis and Design Team. The model is based on their design and optimization of an initial full-scale design of a solid sorbent capture system for a net 650 MW (before capture) supercritical pulverized coal power plant. The A650.1 model describes a solid sorbent-based carbon capture system that uses the NETL-32D sorbent. NETL-32D is a mixture of polyethyleneamine (PEI) and aminosilanes impregnated into the mesoporous structure of a silica substrate. CO₂ removal is achieved through chemical reactions between the amine sites within the sorbent. The A650.1 model is implemented in Aspen Custom Modeler (ACM) and contains many components (e.g., adsorbers, regenerators, compressors, heat exchangers). For the UQ analyses, this manual focuses on the adsorber units, which are responsible for the adsorption of CO₂ from the input flue gas.

In its original form, the A650.1 model consists of a deterministic simulation model, which means to consider all the parameters (e.g. chemical reaction parameters, heat and mass transfer coefficients) to have a fixed value (most likely fixed to a mean value, lower or upper bound for robustness). With the FOQUS UQ module, the model uncertainties can be addressed. Thus, UQ analysis of the A650.1 model would help to develop a robust design by addressing the following questions: * How accurately does each subsystem model predict actual system performance (under uncertain operating conditions)? * Which input parameters should be examined to improve prediction accuracy? * What is each input parameters’ contribution to prediction uncertainty?

2.3 Optimization Under Uncertainty

The Optimization Under Uncertainty (OUU) module in FOQUS is an extension of simulation-based optimization by including the contribution of model parameter uncertainties in the objective function. OUU is useful when inclusion of uncertainties may significantly alter the optimal design configurations. A straightforward approach to include the effect of uncertainty is to replace the objective function with its statistical mean on an ensemble drawn from the probability distributions of the continuous uncertain parameters (other options are available in FOQUS). Alternatively, users can provide a set of ‘scenarios’, where each scenario is associated with a probability. The latter case is often called ‘scenario optimization.’ The FOQUS OUU accommodates both continuous and scenario-based uncertain parameters. OUU makes use of the flowsheet for evaluations of the objective function. Naturally, OUU requires more computational resources than deterministic optimization. However, the ensemble runs can be launched in parallel so ideally, the turnaround time remains about the same as that of deterministic optimization if high performance computing capability (such as the CCSI FOQUS Cloud) is used in conjunction with FOQUS.

2.4 Surrogate Models

Process simulations are often time consuming and occasionally fail to converge. For mathematical optimization, it is sometimes necessary to replace a simulation with a surrogate model, which is a simplified model that executes much faster. FOQUS contains tools for creating and quantifying the uncertainty associated with surrogate models.

2.4.1 ALAMO

While simulation based optimization can often do a good job of providing optimal design and operating conditions for a predetermined flowsheet, it cannot provide an optimal flowsheet. To obtain a more optimal flowsheet, a mixed integer nonlinear program must be solved. These types of problems cannot generally be solved using simulation based optimization. A solution is to generate relatively simple algebraic models that accurately represent the high fidelity models. FOQUS currently provides an interface for ALAMO (*Cozad et al. 2014*), which builds surrogate model that are well suited for superstructure optimization.

2.4.2 ACOSSO

The Adaptive Component Selection and Shrinkage Operator (ACOSSO) surface approximation was developed under the Smoothing Spline Analysis of Variance (SS-ANOVA) modeling framework (*Storlie et al. 2011*). As it is a smoothing type method, ACOSSO works best when the underlying function is somewhat smooth. For functions which are known to have sharp changes or peaks, etc., other methods may be more appropriate. Since it implicitly performs variable selection, ACOSSO can also work well when there are a large number of input variables. To facilitate the description of ACOSSO, the univariate smoothing spline is reviewed first. The ACOSSO procedure also allows for categorical inputs (*Storlie et al. 2013*).

2.4.3 BSS-ANOVA

The Bayesian Smoothing Spline ANOVA (BSS-ANOVA) is essentially a Bayesian version of ACOSSO (*Reich 2009*). It is Gaussian Process (GP) model with a non-conventional covariance function that borrows its form from SS-ANOVA. It tackles the high dimensionality (of inputs) on two fronts: (1) variable selection to eliminate uninformative variables from the model and (2) restricting the level of interactions involved among the variables in the model. This is done through a fully Bayesian approach which can also allow for categorical input variables with relative ease. Since it is closely related to ACOSSO, it generally works well in similar settings as ACOSSO. The BSS-ANOVA procedure also allows for categorical inputs (*Storlie et al. 2013*).

FLWSHEETS AND SETTINGS

This chapter provides general information about using FOQUS and constructing flowsheets. The FOQUS flowsheet provides the basis for other analysis tools.

3.1 Contents

3.1.1 Reference

Getting Started

Follow the installation instructions provided in the *FOQUS Installation and Running* chapter.

The first time FOQUS is started, the user is prompted to specify a working directory. The working directory preference is stored in %APPDATA%\foqus.cfg on Windows (APPDATA is an environment variable). On Linux or OSX, the working directory is specified in \$HOME/.foqus.cfg. Additionally the user can override the working directory when starting FOQUS by using the `--working_dir <working dir>` or `-w <working dir>` command line option. Log files, user plugins, and files related to other FOQUS tools are stored in the working directory. The working directory can be changed at a later time from within FOQUS. A full list of FOQUS command line arguments is available using the `-h` or `--help` arguments.

Home Menu

Session Information Display

FOQUS flowsheet information and settings are stored in a session. The session screen displays information about the current session. A menu is available by clicking the **Session** drop-down menu. The figure below shows the Home window.

Fig. 1: Home Screen

1. The buttons displayed at the top of the Home window, excluding **Help**, are tab-like buttons that change the window when selected. The depressed button indicates the currently displayed window.
 - A. **Session** displays the Session window, which contains a description of the session that is currently open. **Session** has a drop-down menu that displays the Session menu.
 - B. **Flowsheet** displays the meta-flowsheet editing window.
 - C. **Uncertainty** displays the interface for PSUADE and UQ visualization.

- D. **Optimization** displays the simulation-based optimization interface.
 - E. **OOU** displays the optimization under uncertainty interface.
 - F. **Surrogates** displays the surrogate model generation window.
 - G. **DRM-Builder** displays the dynamic reduced model builder, which can be used to develop reduced models for dynamic simulations.
 - H. **Settings** displays the main FOQUS settings window.
2. **Help** toggles the Help browser. The Help browser contains HTML help, licensing and copyright information, log messages, and debugging console.
 3. The main Session window displays information about the current session and is divided into three tabs:
 - **Metadata** displays information about the current FOQUS session. The **Session Name** provides a descriptive name for the session. This name is used by the data management framework and when running flowsheets remotely, so a name is required. Entering a name should be the first step in creating a FOQUS flowsheet. **Version** number can be used to keep track of changes to a FOQUS session. **Confidence** describes whether the FOQUS session is expected to produce reliable results or not. **ID** is a unique identifier to identify a particular saved version of the session. **Creation Time** is the date and time that the flowsheet was first saved. **Modification Time** is the time and date that the flowsheet was last saved.
 - **Description** displays a detailed explanation of the purpose of the current session file, the problem being solved, and other useful information provided by the creator of the session file.
 - **Change Log** displays a record of changes made to the file. If the **Automatically create backup session file, when saving changes** checkbox is selected in FOQUS **Settings**, a backup file should exist for entries in the **Change Log**. The backup can be matched to the **Change Log** by the unique identifier appended to the file name.

Session Menu

The figure below illustrates the **Session** menu.

Fig. 2: Home Window, Session Drop-Down Menu

1. **Add Current FOQUS Session to Turbine...** * upload the current FOQUS session to Turbine. This can be used run a flowsheet in parallel with turbine.
2. **Add/Update Model to Turbine** enables additional models to be uploaded to Turbine. Turbine provides simulation job queuing functionality so models cannot be run in FOQUS until they have been added to the Turbine server.
3. **New Session** clears all session information so that a new session can be started.
4. **Open Recent** shows a list of recently open FOQUS sessions that can be quickly reloaded for convenience.
5. **Open Session** opens a session that was previously saved to a file.
6. **Save Session** saves the current session with the current session file name. If the session has not been previously saved, the user will be prompted to enter a file name. **Save Session** commands the user to save two session files: (1) a file with the selected name and (2) if backup option is enabled, a backup file with a name constructed from the **Session Name** and **ID**. The Session **ID** is shown on the **Session, Metadata** tab. The backup file is saved to the working directory. This system prevents accidental saving over an important file. It also enables the user to open any previously saved session.
7. **Save Session As** is similar to **Save Session**; however, the user is prompted for a new file name.
8. **Exit FOQUS** exits FOQUS. The user is asked whether to save the current session before exiting.

Adding or Changing Turbine Simulations

Before running any flowsheet where a node is linked to a simulation, the simulation must be uploaded to Turbine. To use a simulation at least two things are required: (1) the simulation file (e.g., Aspen Plus file, Excel file) and (2) the SimSinter configuration. The SimSinter configuration file is a JavaScript Object Notation (JSON) formatted file that specifies the simulation, input, and output. Any additional files required to run the simulation must also be uploaded.

Fig. 3: Turbine Upload Dialog Box

1. **Create/Edit** enables use of the SimSinter configuration Graphical User Interface (GUI) to create a SimSinter configuration file. See the *SimSinter documentation* for more information.
2. **Browse** displays a file browser, which can be used to select an existing SimSinter configuration file. Once a SimSinter configuration file is selected, the **Application** type is filled in. The SimSinter **Configuration File** and simulation file are automatically added to the file upload table.
3. **Simulation Name** enables entry of a new name if uploading a new simulation. An existing simulation can be selected from the drop-down list if an existing simulation is being modified. After selecting a SimSinter configuration file, the simulation name is guessed from the SimSinter configuration file name, but it can be edited.
4. **Application** displays the application that will be used to run the simulation. This is filled in automatically based on information in the SimSinter configuration file, and cannot be edited.
5. **Add Files** enables uploading of any auxiliary files that may be required by the simulation. Multiple files may be selected at once.
6. **Remove Files** enables added files to be removed from the list of files to upload.
7. **File Table** displays a list of files to be uploaded to Turbine.
8. **Delete** allows the simulation with the name currently displayed in the **Simulation Name** drop-down list to be deleted from Turbine. Only simulations that have not been run can be deleted.
9. **Resource Relative Path** enables the user to set the path of resource files relative to the simulation working directory. To set the directory, select files in the **File Table**. Multiple files can be selected. Click **Resource Relative Path**, and type the relative path to assign to the selected resource files.
10. **Turbine Gateway Selection** enables the user to select where to upload the simulation (local TurbineLite or AWS FOQUS Cloud). **Current** is the select configuration to run simulations. **Remote** is configured AWS FOQUS Cloud. **Local** is the TurbineLite instance installed on the local computer. **Remote + Local** allows simulations to be uploaded to both the local (TurbineLite) and the AWS FOQUS Cloud. **Multiple/Custom** allows simulations to be uploaded to other Turbine instances by selecting Turbine configuration files.

Settings

The settings screen shows FOQUS settings that are related to the general FOQUS setup, and are unlikely to change between sessions. The settings screen is accessible by clicking the **Settings** button at the top of the Home window. The FOQUS settings can be stored in two locations: (1) “%APPDATA%.foqus.cfg” on Windows or “\$HOME/.foqus.cfg” on Linux or OSX, (2) “foqus.cfg” in the working directory.

The Settings screen displays settings grouped into tabs. Figure *Settings, FOQUS Tab* shows **Settings, FOQUS** tab.

Fig. 4: Settings, FOQUS Tab

Options in the **Settings, FOQUS** tab are described below.

1. **Save settings to working directory**, when checkbox is selected the settings file will be read from the specified working directory. This setting is useful for running multiple copies of FOQUS to ensure the settings do not conflict. When starting additional copies of FOQUS, it is best to start them from the Working Directory command line giving each copy of FOQUS its own independent working directory. If FOQUS is started more than once from the Windows start menu, each copy will use the same working directory. Starting FOQUS multiple times with the same working directory may cause unusual behavior in FOQUS.
2. **Use DMF if available**, when checkbox is selected the Data Management Framework (DMF) module will be loaded and the DMF options will be shown in the **Session** menu.
3. **Automatically create backup session file**, when checkbox is selected each time a FOQUS session is saved it will be saved twice. A backup copy with a universally unique identifier appended to the file name will be saved. This will allow the user to load any previous save point of the session.
4. **Smaller session files**, when checkbox is selected significant storage space is saved by excluding formatting from the session file; this makes the session files less human readable. A more readable session file can be useful for debugging.
5. **FOQUS Flowsheet Run Method** enables the user to select between running simulations on the same computer as FOQUS, or on the AWS FOQUS Cloud. Running simulations remotely on the cloud allows parallel execution. The default setting is “Local”. If the user switches from “Local” to “Remote”, a warning message will appear. The user will be informed that the models that have been uploaded to the Local Turbine may not be available in the AWS FOQUS Cloud. Therefore, the user may need to upload these models into the Cloud in order to run the models remotely.
6. **Working Directory** is the path to the FOQUS working directory. The **Working Directory** is where FOQUS reads and writes files needed to function. When running multiple copies of FOQUS, the **Working Directory** can also be specified from the command line using the “-w” or “-workingDir” options. After changing the **Working Directory**, FOQUS should be restarted.
7. **PSUADE EXE** is the path to the PSUADE executable. PSUADE provides FOQUS’s UQ features.
8. **SimSinter Home** is the path to the SimSinter interface for creating Sinter configuration files for simulations to be run with FOQUS. This setting is not required but it allows easy access to the SimSinter configuration GUI when uploading simulation to Turbine.
9. **iREVEAL Home** is the path the iREVEAL installation. This is required to use the iREVEAL surrogate model module.
10. **ALAMO EXE** is the path to the ALAMO executable. This is required to use the ALAMO surrogate model module.
11. **RScript Path** is the path to the RScript executable. This is required for surrogate model modules that use R as a platform.
12. **Java Home** is the path to the Java installation. The DMF and the iREVEAL surrogate modules require Java.
13. **Revert Changes** The settings changes are applied when the user navigates away from the settings screen. To undo changes made to settings the revert button can be clicked before changing to another screen.

The **Turbine** tab contains settings for configuring the local and remote instance of Turbine. Figure *Settings, Turbine Tab* shows the FOQUS Turbine settings.

Fig. 5: Settings, Turbine Tab

The first section in the **Turbine** tab is **TurbineLite (Local)**. This section contains settings related to the local installation of Turbine, and is only applicable when running FOQUS on the windows platform.

1. **Test** tests the connection to the local Turbine server to make sure it is configured and running properly.

2. **Start Service** starts the Turbine server service on Windows. The user must have permission to start services to use this button.
3. **Stop Service** stops the Turbine server service on Windows. The user must have permission to stop services to use this button.
4. **Change Port** can reconfigure the local Turbine server service on Windows to use a different port. This may be necessary if Turbine conflicts with another service.
5. **Aspen Version**, Aspen 7.3 is still in common use but the API differs slightly from newer versions. This option allows FOQUS to be used with Aspen 7.3.
6. **TurbineLite Home** is the location of the TurbineLite installation. For local simulation runs FOQUS needs to know where TurbineLite is installed so it can launch Turbine consumers to run simulations. This setting is not needed if simulations are only run remotely.
7. **Turbine Configuration (local)** is the path to the TurbineLite gateway configuration file for running simulations locally. If simulations are only run remotely, this setting is not needed. **New/Edit** displays a form to create or edit a Turbine configuration file. Having a setting for both local and remote Turbine allows easy switching between run methods.

The second section in the **Turbine** tab is **Turbine Gateway (Remote)**. This section contains settings related to a remote instance of Turbine.

1. **Test** tests the connection to the remote Turbine server to make sure it is configured and running properly.
2. **Turbine Configuration (remote)**, is the path to the Turbine gateway configuration file for running simulations remotely. If simulations are only run locally, this setting is not needed. **New/Edit** displays a form to create or edit a Turbine configuration file. Having a setting for both local and remote Turbine allows easy switching between run methods.
3. **Check Interval (sec)** is the number of seconds between checking the remote Turbine server for job results. This number should not be set too low to avoid overwhelming the Turbine server with requests.
4. **Number of Times to Resubmit Failed Jobs** is the number of times to resubmit jobs that fail. Jobs occasionally fail due to software bugs. This allows a job to be retried.

The **Logging** tab contains settings related to the FOQUS log files, which provide debugging information. The FOQUS log files are stored in the logs directory in the working directory. Figure [Settings, Logging Tab](#) show the FOQUS log settings. There are two log files (1) FOQUS and (2) Turbine Client.

Fig. 6: Settings, Logging Tab

1. The level sliders indicate how much information to send to the logs.
2. The **Log Files** section enables the user to specify where the log information is sent. The **File Out** checkboxes turn on or off the file output of logs. The **Std. Out** checkboxes enable or disable the output to the screen.
3. **Format** allows the format of the log messages to be changed. See the documentation for the Python 2.7 logging module for more information.
4. **Rotate Log Files** turns on or off log file rotation. When a log file reaches a certain size, a new log file is started and the contents of the old log are moved to a new file. There currently seems to be a bug in the log file rotation which occasionally makes the log file output stop; therefore, the **Rotate Log Files** option is labeled as an experimental feature.

Flowsheet

The meta-flowsheet defines connections between simulations. The flowsheet defines the order that simulations are performed and what data is transferred between them. Simulations are represented as nodes in the flowsheet. These simulations may be links to external simulation software through the SimSinter/Turbine, or custom simulations or simulation wrappers written in Python. Directed edges in the flowsheet connect nodes. The edges also specify which variables in the simulations are equivalent.

If the flowsheet contains cycles, they are solved iteratively. Tear streams are selected by FOQUS based on two criteria: (1) minimize the maximum number of times any cycle is torn and (2) minimize the total number of tear edges (which only is considered when two tear sets have the same value for the first criteria).

FOQUS currently has two methods available for solving flowsheets with recycle: (1) direct substitution and (2) Wegstien *Wegstein 1958*. FOQUS will solve strongly connected components in the order they are encountered in the flowsheet. FOQUS flowsheets are generally not very complicated, so if a strongly connected component contains more than one tear stream, they are solved simultaneously. More advanced solution options will be added if a need arises. Figure *Flowsheet Recycle* shows how a simple flowsheet with recycle would be solved.

Fig. 7: Flowsheet Recycle

Flowsheet Editor

Figure *Flowsheet Editor* illustrates the main **Flowsheet Editor** screen and a description of the pieces. The toolbar on the left contains various flowsheet tools.

Fig. 8: Flowsheet Editor

The first three buttons are mouse mode buttons. The current mouse mode is shown by the depressed button. The remaining buttons on the toolbar perform an action. The flowsheet editing toolbar and flowsheet are described in detail below.

1. **Selection mode** enables the user to select nodes and edges. Multiple items may be selected by holding down the Shift key. To deselect everything, click an empty area of the flowsheet while not holding the Shift key. Selected items can be moved by dragging them. To move multiple items, hold down the Shift key while dragging. The last item selected becomes the current object to be edited in the **Node** or **Edge Editor**.
2. **Add node mode** enables the user to add a node by clicking anywhere on the flowsheet. Once a location is clicked, a dialog box opens where the new node name can be entered. If **Cancel** is selected, no node is added. The new node name cannot be “graph” and cannot match any existing node name.
3. **Add edge mode** enables edges to be added by selecting the node that the edge originates from, followed by the node the edge terminates at.
4. **Center flowsheet in display** centers the display on the flowsheet.
5. **Delete selected** deletes all selected nodes and edges. If a node is deleted, all edges connecting to that node are also deleted.
6. **Run a simulation** starts a single simulation run. This is primarily used to test a simulation before running optimization or UQ.
7. **Stop a simulation** is enabled when a simulation is running and stops any running simulation. The simulation may take several seconds to stop.
8. **Set inputs to defaults** returns all of the inputs to their default values.

9. **Determine tear edges** makes it easier to see where initial guesses are needed and makes it possible to edit the tear set before running the flowsheet. If tear streams are needed but not specified before running a flowsheet, they will be automatically specified, however inputs that will be used for the initial guess will not be known before running.
10. **Flowsheet solver settings** contains options related to tear solvers.
11. **Toggle node editor display** displays or hides the **Node Editor**. The user can change the node being edited by selecting from **Name** in the **Node Editor** or selecting it on the flowsheet in selection mode.
12. **Toggle edge editor display** displays or hides the edge editor. The user can change the edge being edited in the **Edge Editor**, or by selecting it in selection mode.
13. **Show results from all flowsheet runs** displays the results of all flowsheet runs in a table view. This can be exported to a spreadsheet.
14. **Node** represents a simulation or calculations.
15. **Edge** connects simulation data, represents data transfer between two nodes.

Node Editor

The **Node Editor** enables the assignment of simulations to a node, and editing variables. Figure *Node Editor Window* shows the Node Editor window with the input variables section of the toolbox displayed.

Fig. 9: Node Editor Window

1. **Apply** immediately applies any changes made in the **Node Editor**. This is not usually needed. Changes are applied when the current node is changed, the **Node Editor** is closed, or some other action is taken that requires the flowsheet, such as running the flowsheet.
2. **Revert** sets the node back to the version where the changes were last applied. This is usually the original state of the node when the editor was opened.
3. **Run** can be used to run the simulation represented by this node only. This can be used for testing to make sure the node is properly configured without running the whole flowsheet.
4. **Stop Run** is active when a simulation is currently running. It stops a single node run or a flowsheet run.
5. There are three tabs in the **Node Editor**: (1) **Variables** tab, shown in Figure *Node Editor Window*, (2) **Position** tab displays the coordinates of the node, and (3) **Node Script** tab enabling the entry of Python code to be executed after the simulation is run.
6. **Name** displays the name of the node currently being edited. The current node can be changed by selecting from existing nodes in the drop-down menu.
7. **Code** displays the error status code for the node.
8. **Message** displays a more detailed description of the error status of the node.
9. **Type** enables the user to select the type of model to run. The model types are none, Turbine, DMF Lite, DMF Server, or Python Plugin. None allows no model to be assigned to the node; this is useful when the node only executes a script entered directly into FOQUS. Turbine is used to execute Aspen, gPROMS, or Excel simulations. Python plugins are custom simulations or wrappers written by the user. Surrogate model methods may also produce Python plugin models.
10. **Model** enables selection of the models available on Turbine or loaded Python plugins.
11. **Input Variables** enables viewing and editing the node's input variables. Most of these variables are added automatically when a simulation is selected.

- a. **Add variable** enables the addition of an input variable. There are two reasons to add an input: (1) to use a variable to pass information to another simulation (even if the variable is not used in any node calculation, it can receive data from the previous simulation and be passed on to the next simulation) and (2) to use in a node script. For example, a variable could be added that provides output in different units of measure.
 - b. **Remove variable** removes variables. If an input variable is removed that originally came from a Turbine simulation, the simulation will run with the default value.
 - c. **Tags** displays a tag browser that lists commonly used variable tags.
 - d. **Input Variables** table displays information about variables. Most attributes can be edited, except for the **Name** column within the **Input Variables** table. The rows for input variables are color coded depending on whether they are set by an edge from results in another node. White rows are not connected. Yellow rows are set by a tear edge. These variables serve as initial guesses but their value may change once the simulation has run. Red rows are set by an edge that is not a tear edge. The value set for these inputs does not matter and it may change once the simulation has run.
12. **Output Variables** is a variable table similar to the **Input Variables** table for node output variables. This area is displayed by clicking **Output Variables**.
 13. **Settings** displays simulation settings. A description is provided for each setting. The available settings vary depending on simulation.

Node Variables

Variables in the node editor are grouped into two sections, inputs and outputs. The input and output variable tables are accessible as described in the previous section. The contents of the variable tables are described here.

The columns in the input variable list are:

- **Name** is the name of the variable,
- **Value** is the current value,
- **Unit** is the unit of measure,
- **Type** is the data type (float, int, or str),
- **Default** is the default value,
- **Min** is the minimum value,
- **Max** is the maximum value,
- **Description** is a description string,
- **Tag** is a list of strings that can be used to attach additional information to a variable
- **Distribution** is a distribution type,
- **Param1** is the first parameter of a parametric distribution the exact meaning depends on the selected distribution, and
- **Param2** is the second parameter of a parametric distribution the exact meaning depends on the selected distribution.

The minimum and maximum values are not enforced when running simulations. A value can be given outside the range. Optimization and UQ features make use of these values to set upper and lower bounds on decision variables or sampling. The distribution information is used when setting up sampling for UQ. In the future, this may also be used for things like optimization under uncertainty. Integer and string type variables cannot currently be used as optimization decision variables, or sampled with the UQ tool.

The rows of the input variable table are color coded. Some of the input variables may be set by connections to other nodes. White rows are variables whose values are not set by a connection. The variables that are red have values set by a connection, and the value given will be overwritten and does not matter. The values that are colored yellow are inputs set by a connection that is a tear stream. The values of these variables serve as an initial guess for solving recycles.

The output variable table is similar to the input table, however it only contains the columns: Name, Value, Unit, Type, Description, and Tags. The value of the outputs may not correspond to the inputs until the simulation has been run.

Node Script

There are three types of **Node Script** that can be used: (1) **Pre** runs before a node simulation, (2) **Post** runs after a node simulation, and (3) **Total** scripts how a node runs the simulation.

Figure *Node Script Tab* illustrates the **Node Script** tab of the **Node Editor** with calculations for an optimization test problem.

Fig. 10: Node Script Tab

Node scripts can be any valid Python code. The input and output variables for node scripts are stored in dictionaries *x* and *f*. The dictionary keys are the variable names. The *f* dictionary is used to update the node variables after the calculations are executed.

Edge Editor

The **Edge Editor** is illustrated in Figure *Edge Editor*. The **Edge Editor** can be used to set connections between node variables.

Fig. 11: Edge Editor

1. **Index** is the index of the current edge. The current edge can be changed by selecting an index from the drop-down menu, but since the index is not a very meaningful identifier it is usually more convenient to select the edge to edit with the graphical selection tool.
2. **Origin Node** is the node where an edge starts. This may be edited by selecting a different node from the drop-down menu.
3. **Destination Node** is the node to which the edge goes.
4. **Curve** can be a positive or negative number. The greater the magnitude of number, the more curved an edge will appear in the flowsheet. This setting is used to keep edges from overlapping in the flowsheet display.
5. **Tear** marks this edge as a tear. Before a simulation is run, if a valid tear set is not specified, FOQUS locates one.
6. **Active** specifies whether the edge is active or not. This allows connections to be temporarily disabled.
7. **Variable Connections** table displays which variables are connected. Inputs or outputs in the origin node can be connected to inputs in the destination node.
8. **Add connection** adds a new connection.
9. **Remove connection** deletes the selected connections.
10. **Auto** automatically connects variables having the same name. For example, in connecting a simulation to a spreadsheet to calculate costs there are a large number of variables for which it makes sense that the variables have the same name in the simulation and spreadsheet. **Auto** should be used with great care. Connecting variables with the same name is often not what is wanted. For example two simulations may have a variable named FlowAIn;

however, it is very unlikely that they should be connected. It is more likely FlowAOut should be connected to FlowAIn.

Sample Results

Flowsheet evaluations that have been run in a FOQUS session can be viewed by clicking the table button in the flowsheet toolbar (#13 in Figure *Flowsheet Editor*). The results are displayed in a table, and the contents can be copied and pasted into a spreadsheet or exported to a CSV file. Figure *Flowsheet Results Table Window* show the Flowsheet Results Table window.

Fig. 12: Flowsheet Results Table Window

1. **Menu** contains a menu with four sub menus.
 1. **Import** data from files or the clipboard.
 2. **Export** data to files or the clipboard.
 3. **Edit** or delete data.
 4. **View** options for the table.
2. The **Current Filter** drop-down list enables the user to select a data filter, which can be used to filter and sort data.
3. **Edit Filters** enables the user to create or edit data filters.

Error Codes

Error codes are listed in the **Flowsheet Results** table for the whole flowsheet and for individual nodes. Table *Flowsheet Error Codes* shows the flowsheet error codes and Table *Node Error Codes* shows the node error codes. The most common flowsheet error is 1, a node calculation failed. The most common node error is 7, Turbine simulation error. These errors are typically caused by a simulation that fails to converge or has some other calculation error (e.g., ACM does not converge or an Excel spreadsheet simulation with a division by 0 error).

Table 1: Flowsheet Error Codes

Code	Meaning
-1	Did not run or finish
0	Success
1	A simulation/node failed to solve
2	A simulation/node failed to solve while solving tears
3	Failed to create a worker node
5	Unknown tear solver
11	Wegstein failed, reached iteration limit
12	Direct failed, reached iteration limit
16	Presolve node error
17	Postsolve node error
19	Unhandled exception during evaluation (see log)
20	Flowsheet thread terminated
21	Missing session name
40	Error connecting to Turbine
50	Error loading session or inputs
100	Single node calculation success
201	Cycle in determining calculation order (invalid tear set)

Table 2: Node Error Codes

Code	Meaning
-1	Did not run or finish
0	Success
1	Simulation error (see log)
3	Exceeded maximum wait time
4	Failed to create Turbine session ID
5	Failed to add Turbine job
6	Exceeded maximum run time
7	Turbine simulation error
8	Failed to start Turbine job
10	Failed to get Turbine jobs status
11	Flowsheet thread terminated
20	Error in node script
23	Could not convert Numpy value to list
27	Cannot read variable result (see log)

3.1.2 Tutorial

Tutorial 1: Creating a Flowsheet

The Basics

This tutorial provides information about the basic use of FOQUS and setting up a very simple flowsheet. A single node flowsheet will be created that performs a simple calculation using a square root so that simulation errors can be observed when a negative input value is provided.

This tutorial will show the user the procedure for creating a flowsheet in FOQUS. However, if the user is interested, the finished flowsheet is available in: `examples/tutorial_files/Flowsheets/Tutorial_4`

Note: The **examples/** directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

1. Start FOQUS (see Section *Getting Started*).
2. In the session form enter the **Session Name** as “Simple_Flow” (Figure *Setting the Session Name*).

Fig. 13: Setting the Session Name

3. Set the session description.
 - a. Select the **Description** tab (Figure *Setting the Session Description*).
 - b. Type the description shown in Figure *Setting the Session Description*. The buttons above the **Description** tab box can be used to format the text.

Fig. 14: Setting the Session Description

4. Click the **Flowsheet** button at the top of the Home window (Figure *Flowsheet, Input Variables*).
5. Add a node named “calc.”
 - a. Click the **Add Node** button in the toolbar on the left side of the Home window.
 - b. Click a location on the gridded flowsheet area.
 - c. Enter the node name “calc” in the dialog box.
6. Click the **Select Mode** button in the toolbar.
7. Open the Node Editor by clicking the **Node Editor** button in the toolbar.
8. Add input variables to the node. (When linking a node to an external simulation the input and output variables are populated automatically, and this step is not necessary.)
 - a. Click + above the **Input Variables** table.
 - b. Enter x1 in the variable **Name** dialog box. Enter 1 for the variable size, -2 for the min, 2 for the max, and 1 for the value.
 - c. Click + above the **Input Variables** table.
 - d. Enter x2 in the variable **Name** dialog box. Enter 1 for the variable size, -1 for the min, 4 for the max, and 4 for the value.

Fig. 15: Flowsheet, Input Variables

9. Add an output variable to the node. (When linking a node to an external simulation the input and output variables are populated automatically.)
 - a. Click **Output Variables** to show the **Output Variables** table (Figure *Flowsheet, Output Variables*).
 - b. Click + above the **Output Variables** table to add a variable.
 - c. Enter z in the output **Name** dialog box.

Fig. 16: Flowsheet, Output Variables

In this example, the node is not linked to any external simulation. The FOQUS nodes contain a section called node script, which can be used to do calculations before, after or instead of a simulation linked to the node. The node script can be used for things such as unit conversion, simple calculations, or simulation convergence procedures. The node scripts are written as Python. The **Input Variables** are contained in a dictionary named `x` and the **Output Variables** are contained in a dictionary named `f`. The dictionary keys are the variables names shown in the input and output tables. Only **Output Variables** can be modified by a node script.

10. Add a calculation to the node.
 - a. Click the **Node Script** tab (Figure *Node Calculation*).
 - b. Enter the following code into the Python code box:


```
f['z'] = x['x1']*math.sqrt(x['x2'])
```
11. Click the **Variables** tab.
12. Click the **Run** button (Figure *Node Calculation*).

The flowsheet should run successfully and the output value should be 2. Rerun the flowsheet with a negative value for `x2`, and observe the result. The simulation should report an error.

Fig. 17: Node Calculation

13. Save the FOQUS session.
 - a. Click the **Session** drop-down menu at the top of the Home window (Figure *Save Session*).
 - b. Click **Save**. The exact location of save in the menu depends on whether or not the data management framework is enabled.
 - c. The **Change Log** entry can be left blank.
 - d. The default file name is the session name. Change the file name and location if desired.

Fig. 18: Save Session

Automatically running FOQUS for a set of user-defined input conditions

This procedure requires the Uncertainty Tab.

Therefore, the instructions for this procedure can be found in the documentation under:

Uncertainty Quantification / Tutorial / Simulation Ensemble Creation and Execution / Automatically running FOQUS for a set of user-defined input conditions

The link for these instructions is shown below:

https://foqus.readthedocs.io/en/latest/chapt_uq/tutorial/sim.html

Tutorial 2: Creating a Flowsheet with Linked Simulations

Note: This tutorial requires the user to have Aspen Custom Modeler installed on their machine.

Note: This tutorial utilizes SimSinter and Turbine, two optional software packages that integrate with FOQUS to run Aspen and Excel models. The installation instructions are located at *Install Turbine and SimSinter (Windows Only)*.

This tutorial is referenced by other tutorials. **Save the flowsheet in a convenient location for future use.**

This tutorial demonstrates how to link simulations to nodes, and how to connect nodes in a flowsheet. Two models are used: (1) a bubbling fluidized bed model in ACM and (2) a cost of electricity (COE) model in Excel. The COE model estimates the cost of electricity for a 650 MW (net before adding capture) supercritical pulverized coal power plant with solid sorbent post combustion CO₂ capture process added.

The files for this tutorial is located in: `examples/test_files/Optimization/Model_Files`.

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

1. Start FOQUS. The Session window displays (Figure *Session Setup*).
2. Enter “BFB_opt” in **Session Name** (without quotes).
3. Click the **Description** tab. The problem description box displays and is shown in (Figure *Session Description*).
4. In the problem description box enter information about the problem being solved in the FOQUS session; this information can be more extensive than what is shown in the example.
5. Save the session file. Click **Save Session** from the **Session** drop-down menu. Enter change log information and a file name when prompted. The **Creation Time** in metadata page will be the time the session is first saved. The **Modification Time** will be the last time the session was saved. The **ID** is a unique identifier that changes each time the user saves the simulation. The **Change Log** tab provides a record of the changes made each time the session is saved.

Fig. 19: Session Setup

Fig. 20: Session Description

There are two models needed for this optimization problem: (1) the ACM model for the BFB capture system and (2) the Excel cost estimating spreadsheet. These models are provided in the `examples/test_files/Optimization/Model_Files` directory. There are two SimSinter configuration files: (1) `BFB_sinter_config_v6.2.json` for the process model and (2) `BFB_cost_v6.2.3.json` for the cost model. The next step is to upload the models to Turbine.

6. Open the **Add/Update Model to Turbine** dialog box (Figure *Open Upload to Turbine Dialog*).
7. In this case, the SimSinter configuration files have already been created. If a SimSinter configuration file needs to be created for the simulation, **Create/Edit** displays the SimSinter configuration GUI (see Figure *Upload to Turbine Dialog*). See the SimSinter documentation or Chapter *Simulation Standard Interface (SimSinter)* for more information.
8. Click **Browse** to select a SimSinter configuration file (Figure *Upload to Turbine Dialog*). Once the SimSinter configuration file is selected, the simulation file and sinterconfig file is automatically added to the files to upload.

The application type is entered automatically. If there are additional files required for the simulation, those files can be added by clicking **Add File**.

9. Enter the simulation name in **Simulation Name**. This name is determined by the user, but will default to the SimSinter configuration file name. For this tutorial use BFB_v6_2.
10. Click OK to upload the simulation.
11. Repeat the upload process for the cost model. Name the model BFB_v6_2_Cost.

Fig. 21: Open Upload to Turbine Dialog

Fig. 22: Upload to Turbine Dialog

The next step is to create the flowsheet. Figure *Flowsheet Editor* illustrates the steps to draw the flowsheet.

12. Click **Flowsheet** at the top of the Home window.
13. Click **Add Node mode**.
14. Add two nodes to the flowsheet. Name the first node “BFB” and the second node “cost”.
15. Click **Add Edge mode**.
16. Click the BFB node followed by the cost node.
17. Click **Selection mode** and select the BFB node.
18. Click **Toggle Node Editor**. The Node Editor displays as illustrated in Figure *Node Editor*.

Fig. 23: Flowsheet Editor

Each node must be assigned the appropriate simulation. Use the Node Editor to set the simulation type and the simulation name from simulation uploaded to Turbine. The Node Editor is illustrated in Figure *Node Editor*

19. Under **Model** and **Type**, set the simulation **Type** to Turbine. This indicates that the simulation is to be run with Turbine.
20. Under **Model**, set the simulation of the BFB node to BFB_v6_2.
21. The **Variables** and **Settings** are automatically populated from the SimSinter configuration file. Variable values, **Min/Max**, and descriptions can be changed; however, for this problem, the values taken from the SimSinter configuration should not be changed.
22. Repeat the process for the cost node, assigning it the BFB_v6_2_cost simulation.

The connections between variables in the BFB simulation and the cost estimation spreadsheet must be set, so that required information can be transferred from the BFB simulation to the cost simulation.

23. Click **Toggle Node Editor** to hide the Node Editor (Figure *Flowsheet Editor*).
24. Select the edge on the flowsheet with the **Selection** tool.
25. Click **Toggle Edge Editor** to show the Edge Editor. The Edge Editor is shown in Figure *Edge Editor*.
26. For convenience, all of the variables that should be connected from the ACM model to the Excel spreadsheet have been given the same names in their SimSinter configuration files. To connect the variables click **Auto** in the Edge Editor. **Auto** connects variables of the same name. Since this is often not desired, the **Auto** button should be used carefully. There should be 46 connected variables.

Node Edit

Variables Position Node Script

Name: BFB Visible

Error Status

Code:

Message: Did not finish

Model:

Type: Turbine Model: BFB_v6_2

Input Variables

	Name	Value	Unit	Type	Default	Min	Max	Description	Tags
1	adsDt	15.0	m	float	15.0	9.0	15.0	Adsorption units diameter	<input type="checkbox"/>
2	adsdx	0.0275	m	float	0.0275	0.0175	0.03	Adsorption units HX tubes diameter	<input type="checkbox"/>
3	adslhx	0.4	m	float	0.4	0.0075	0.55	Adsorption units HX tubes spacing	<input type="checkbox"/>
4	adsN	15.0		float	15.0	4.0	15.0	Number of parallel adsorption trains	<input type="checkbox"/>
5	BFBadsB.Lb	4.2	m	float	4.2	2.8	4.2	Bottom adsorber bed depth	<input type="checkbox"/>
6	BFBadsM.Lb	4.2	m	float	4.2	2.8	4.2	Middle adsorber bed Depth	<input type="checkbox"/>
7	BFBadsT.Lb	4.2	m	float	4.2	2.8	4.2	Top adsorber bed depth	<input type="checkbox"/>
8	BFBrgnB.Lb	4.2	m	float	4.2	2.8	4.2	Bottom regenerator bed depth	<input type="checkbox"/>
9	BFBrgnT.Lb	4.2	m	float	4.2	2.8	4.2	Top regenerator bed depth	<input type="checkbox"/>
10	GHXfg.GasOut.T	40.0	degC	float	40.0	25.0	40.0	Flue gas cooler outlet temperature	<input type="checkbox"/>
11	rgnDt	12.0	m	float	12.0	9.0	12.0	Regeneration units diameter	<input type="checkbox"/>
12	rgndx	0.0225	m	float	0.0225	0.014	0.026	Regeneration units HX tubes diameter	<input type="checkbox"/>

Legend: Not Connected Tear Connected Connected

Output Variables

Settings

Fig. 24: Node Editor

Fig. 25: Edge Editor

The flowsheet should now be ready to run. Test the flowsheet by executing a single evaluation before setting up the optimization problem.

27. Click **Run** in the Flowsheet Editor (Figure *Flowsheet Editor*).
28. The flowsheet may take a few minutes to run. The BFB simulation takes a significant amount of time to open in ACM. While running optimization, the evaluations take less time because the simulation remains opened. The simulation should complete successfully. A message box displays when the simulation is done. The status bar also indicates the simulation is running.
29. While the simulation is running, **Stop** is enabled.
30. Once the simulation runs successfully, **Save** the FOQUS session again, and **keep it for use in later tutorials**.

Tutorial 3: Flowsheets with Recycle

This section provides a tutorial on working with flowsheets containing recycle. Sections *Tutorial 1: Creating a Flowsheet* and *Tutorial 2: Creating a Flowsheet with Linked Simulations* provide tutorials for creating flowsheets, in this section a pre-constructed flowsheet is used.

The file for this tutorial is `Mass_Bal_Test_02.foqus`, and this file is located in `examples/tutorial_files/Flowsheets/Tutorial_3`.

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

1. Open FOQUS.
3. Open the `examples/tutorial_files/Flowsheets/Tutorial_3/Mass_Bal_Test_02.foqus` file.
 1. Open the **Session** drop-down menu on the right side of the **Session** button (Figure *Flowsheet with Recycle*).
 2. Select **Open Session** from the drop-down menu.
 3. Locate `Mass_Bal_Test_02.foqus` in the file browser, and open it.
4. Click **Flowsheet** button from the toolbar at the top of the Home window.

The flowsheet is shown in Figure *Flowsheet with Recycle*. The flowsheet consists of two reactors in recycle loops. The flowsheet contains mixers, reactors, separators, and splitters. Each node uses a set of simple calculations in the node script section. The tear edges are shown in light blue.

Fig. 26: Flowsheet with Recycle

5. Inspect a node.
 1. Make sure the Selection tool is selected (Figure *React_01 Node*).
 2. Open the Node Editor by clicking the **Node Edit** button in the left toolbar in the Flowsheet view.
 3. Click the “React_01” node.
 4. Click **Input Variables** table. Note: Some input rows are colored red. This denotes that their values are set by output of the previous flowsheet node by the edge connecting “Mix_01” to “React_01.”
 5. Click the **Node Script** tab.
 6. Note the equations. **Input Variables** are stored in the `x` dictionary and **Output Variables** are stored in the `f` dictionary.

6. Click the gear icon in the left toolbar (see Figure *React_01 Node*. The tear solver settings are shown in Figure *Tear Solver Settings*.

Fig. 27: React_01 Node

Fig. 28: Tear Solver Settings

7. Remove the tear edges.
 1. Close the Node Editor.
 2. Open the Edge Editor. Click the **Edge Editor** icon in the left toolbar (see Figure *Edge Edit*.
 3. Click the edge between “React_01” and “Sep_01.”
 4. In the Edge Editor, clear the **Tear** checkbox.
 5. Repeat for the other tear edge.
8. Close the Edge Editor.

Fig. 29: Edge Edit

There should now be no tear edges in the flowsheet. The user can select tear edges or FOQUS can automatically select a set. If there is not a valid set of tear edges marked when a flowsheet is run, tear edges will automatically be selected.

9. Automatically select a tear edge set by clicking the **Tear** icon in the left toolbar (see Figure *Edge Edit*).
10. Open the Node Editor and look at node “Sep_01.” In the Input Variables table, notice that some of the input lines are colored yellow. The yellow inputs serve as initial guesses for the tear solver. The final value will be different from the initial value.
11. Click the **Run** button on the left toolbar. The flowsheet should solve quickly.
12. The results of the completed run are in the flowsheet. An entry will also be created in the Flowsheet Results data table (see Section *Tutorial 4: Flowsheet Result Data*).

Tutorial 4: Flowsheet Result Data

Flowsheet evaluation results are stored in a table in the FOQUS session. This data can be used for many purposes. The flowsheet evaluations may be single runs, part of an optimization problem, or part of a UQ ensemble. This tutorial provide information about sorting, filtering, and exporting data.

The FOQUS file for this tutorial is `Simple_flow.foqus`, and this file is located in `examples/tutorial_files/Flowsheets/Tutorial_4`.

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

The `Simple_flow.foqus` file is similar to the one created in the tutorial Section *Tutorial 2: Creating a Flowsheet with Linked Simulations*, but it has been run an additional 100 times using a UQ ensemble (see *Tutorial 1: Simulation Ensemble Creation and Execution*).

1. Open FOQUS.

2. Open the `examples/tutorial_files/Flowsheets/Tutorial_4/Simple_flow.foqus` session from the example files.
3. Click the **Flowsheet** button from the Home window.
4. Click **Flowsheet Data** in the toolbar on the left side of the Home window.

Fig. 30: Flowsheet Results Data Table, All Data

A data table should be displayed like the one shown in the figure below. There are 102 flowsheet evaluations. The first two evaluations are single runs, as can be seen in the **SetName** column, and the remaining 100 evaluation are from a UQ ensemble. The **Error** column shows several of the evaluations resulted in an error from a negative number being passed to the square root function.

This tutorial is broken up into mini-tutorials in the remaining subsections, which can be done independently. They each use the example data file described above.

Sorting Data

1. Open FOQUS.
2. Open the `Simple_flow.foqus` session from the example files.
3. Click **Flowsheet** in the main toolbar at the top of the FOQUS Home window.
4. Click **Flowsheet Data** in the toolbar on the left side of the Home Window.
5. Click **Edit Filters**.
6. Click **New Filter**.
7. Enter "Sort1" as the new filter name.
8. Click **New Filter**.
9. Enter "Sort2" as the new filter name.
10. Select "Sort1" from the **Filter** drop-down list.
11. Enter ["-result"] as the **Sort by Column**. Include the square brackets. The square brackets indicate that there is a list of sort terms, although in this case there is only one. If multiple search terms are given, the additional terms will be used to sort results having the same value for the previous terms. The "-" in front of **result** indicates the results should be sorted in reverse. The names of the sort terms come from the column headings, and are case sensitive.
12. Click **Done** to save the filters and return to the results table.

Fig. 31: Sort1 Data Filter

14. Select "Sort1" from the **Current Filter** drop-down list.
15. The results are shown in below. The data should be sorted in reverse alphabetical order by **result**. Some of the columns are hidden to make the relevant results easier to see.

Fig. 32: Sort1 Data Filter Results

16. Click **Edit Filters**.

17. Select “Sort2” from **Filter** drop-down list.
19. Enter ["err", "-result"] in the **Sort Term** field. This will sort the data first by **Error** code then by **result** in reverse alphabetical order.
20. Click **Done**.

Fig. 33: Sort2 Data Filter

21. Select “Sort2” in the **Current Filter** drop-down list.
22. The results are shown in below. The data should be sorted so all **Error** code zero results are first then sorted in reverse alphabetical order by **result**.

Fig. 34: Sort2 Data Filter Result

Filtering Data

1. Open FOQUS.
2. Open the Simple_flow.foqus session from the example files.
3. Click the **Flowsheet** button in the Home window.
4. Click the Results Data button (Table icon in left toolbar).
5. In the data table dialog, click **Edit Filters**.
6. Click **New Filter** and enter “Filter1” in the **Filter** field as the new filter name.

The filter expression is a Python expression. The `c("Column Name")` function returns a numpy array containing the column data. The expression should evaluate to a column of booleans where rows containing `True` will be included in the filtered results and rows containing `False` will be excluded. If combining multiple logical expressions the numpy logical functions <https://docs.scipy.org/doc/numpy-1.15.1/reference/routines.logic.html> should be used. Numpy is imported as `np`

8. In this example, results without errors in the “Single_runs” should be selected. In the filter expression field enter `np.logical_and(c("err") == 0, c("set") == "Single_runs")`
10. Click **Done**.

Fig. 35: Filter1 Data Filter

11. In the data table dialog, select “Filter1” from the **Current Filter** drop-down list.
12. The result is displayed in the Figure below.

Fig. 36: Filter1 Data Filter Result

Exporting Data

This tutorial uses a spreadsheet program such as Excel or Open Office. The exported data is subject to the selected filter. See the previous tutorials in this section for more information about sorting and filtering data to be exported.

Clipboard

FOQUS can export data directly to the Clipboard. The data can be pasted into a spreadsheet or as text. Copying data to the Clipboard eliminates the need for an intermediate file when creating spreadsheets.

1. Open FOQUS.
2. Open a spreadsheet program.
3. Open the Simple_flow.foqus session from the example files.
4. Click the **Flowsheet** button in the Home window.
5. Click the Results Data button (Table icon in left toolbar).
6. Click on the **Menu** drop-down list in the data table dialog.
7. Select “Export” from the **Menu** drop-down list.
8. Click **Copy Data to Clipboard**.
9. Select Paste in the spreadsheet program. The data table in FOQUS should paste into the spreadsheet. Filters can be used to sort or reduce the exported data.

CSV File

CSV (comma separated value) files can be read by almost any spreadsheet program, and are common formats readable by many types of software. FOQUS exports CSV files using the column headings from the data table as a header.

1. Open FOQUS.
2. Open a spreadsheet program.
3. Open the Simple_flow.foqus session from the example files.
4. Click the **Flowsheet** button in the Home window.
5. Click the Results Data button (Table icon in left toolbar).
6. Click the **Menu** drop-down list.
7. Select “Export” from the **Menu** drop-down list.
8. Click **Export to CSV File**.
9. Enter a file name in the file dialog.
10. In the spreadsheet program, open the CSV file exported in the previous step.

Tutorial 5: Using the AWS FOQUS Cloud

The AWS FOQUS Cloud may be used instead of TurbineLite. TurbineLite, used by default, runs simulations (e.g., Aspen Plus) on the user's local machine. The AWS FOQUS Cloud has several potential advantages over TurbineLite, while the main disadvantage is the effort required for installation and configuration. Some reasons to run a on the AWS FOQUS Cloud are:

- Users don't have to install Aspen on their local machine.
- Users with Apple Macintosh or Linux-based systems can run flowsheets with Aspen simulations on the AWS FOQUS Cloud.
- Simulations can be run in parallel. The AWS FOQUS Cloud can scale up to hundreds of virtual machines configured to run FOQUS flowsheet consumers. FOQUS consumers are basically additional instances of FOQUS running on remote systems which can run a FOQUS flowsheet.
- Simulations can be run on machines other than the user's, so as not to tie-up the user's machine running simulations.

Running on AWS FOQUS Cloud

The steps below demonstrate how to set up FOQUS to run flowsheets remotely if the user would like to run FOQUS in parallel in AWS (see Figure *Remote Turbine Settings*).

1. Obtain a user name, password, and URL. Ask on the ccsi-support list
2. Open FOQUS.
3. Click **Settings** at the top right of the Home window (Figure *Run Method Settings*).
4. Select "Remote" from the **FOQUS Flowsheet Run Method** drop-down list. A message box will appear. The user will be warned that the models that have been uploaded to Turbine Local may not be available in Turbine Remote Gateway, which means that the user may need to upload the models into Turbine again (please see Step 7).
5. Click the **Turbine** tab; this displays the Turbine settings shown in Figure *Remote Turbine Settings*.

Fig. 37: Run Method Settings

6. Create a Turbine configuration file; this contains your password in plain text, so it is very important that if you are allowed to choose your own password, you choose one that is not used for any other purpose.
 - a. Click **New/Edit** next to the **Turbine Configuration (remote)** field. The Turbine Configuration window displays (see Figure *Remote Turbine Settings*).
 - b. Select "Cluster/Cloud" from the **Turbine Gateway Version** drop-down list in the Turbine Configuration window.
 - c. Enter the AWS FOQUS Cloud URL in the **Address** field.
 - d. Enter the **User** name and **Password**.
 - e. Click **Save as** and enter a new file name.
 - f. Set the remote Turbine configuration file. Click **Browse** next to the **Turbine Configuration (remote)** field. Select the file created in Step 6e.

Fig. 38: Remote Turbine Settings

At this point FOQUS is ready to use the AWS FOQUS Cloud. The last step is to ensure that all simulations referenced by flowsheets to be run are uploaded to the AWS FOQUS Cloud.

7. Upload any necessary simulations (see Section *Adding or Changing Turbine Simulations* and the tutorial in Section *Tutorial 2: Creating a Flowsheet with Linked Simulations*)

Once all settings are specified there is no apparent difference between running flowsheets locally or on the AWS FOQUS Cloud, and FOQUS can readily be switched between the two.

OPTIMIZATION

4.1 Contents

4.1.1 Reference

The simulation based optimization tool provides a plug-in system where different derivative free optimization (DFO) solvers can be used with FOQUS. Several solvers are provided with FOQUS. The CMA-ES solver (*Hansen 2006*) is a good global derivative free optimization (DFO) solver. The NLOpt library provides access to several DFO solvers (*Johnson 2015*). SLSQP and BFGS from the Scipy module are also provided (*Jones et al. 2015*). Since FOQUS does not generally have access to derivative information the Scipy solvers rely on finite difference approximations, and should only be used with well-behaved functions. Due to convergence tolerances in process simulators, finite difference approximations may not be good for many of FOQUS's intended applications.

CMA-ES offers a restart feature, which can be used to resume an optimization if it is interrupted for any reason. Other solvers may use an auto-save feature, which does not provide the ability to restart, but will allow optimization to start from the best solution found up to the point the optimization was interrupted. Samples making up the population in CMA-ES can be run in parallel. The NLOpt and Scipy plugins do not offer parallel computing for standard optimization. For any solver, parallel computation can be used for parameter estimation and optimization under uncertainty, where multiple flowsheet evaluations go into an objective function calculation.

Problem Set Up

See Chapter *Flowsheets and Settings* for information about setting up a flowsheet in FOQUS. Once the flowsheet has been set up and tested, an optimization problem can be added. FOQUS allows multiple flowsheet evaluations to be used to calculate a single objective function value. This allows FOQUS to do parameter estimation and scenario based optimization under uncertainty. There are three types of variables used in the optimization problem: (1) fixed variables do not change during the optimization, (2) decision variables are modified by the optimization algorithm to find the best value of the objective function, and (3) sample variables, which are used to construct the multiple flowsheet evaluations that can go into an objective calculation. If no sample variables are defined, each objective function value will be based on a single flowsheet evaluation. Figure *Optimization Variable Selection* shows the **Variables** tab selection form.

Fig. 1: Optimization Variable Selection

1. The **Variables** tab contains the form for variables selection.
2. The **Variable** column shows the name of input variables in the flowsheet. If a variable is set by a connection to another variable through an edge, it is not shown in the table. The format for a variable name is {Node Name}.{Variable Name}.
3. The **Type** column allows the variables to be assigned as one of three types (1) fixed, (2) decision, or (3) sample.

4. The **Scale** column allows the scaling method to be set for each variable. Decision variables must be scaled. Scaling is ignored for other variables. In the FOQUS example files, there is a scaling spreadsheet that provides a demonstration of the different scaling methods. The upper and lower bound are used in the scaling calculations. Regardless of the scaling method, the optimizer sees the decision variables as running from 0 at their minimum to 10 at their maximum.
5. The **Min** and **Max** columns are used to define the upper and lower bounds for the variables. FOQUS requires that all optimization problems be bounded.
6. The **Value** column provides the starting point for the optimization. How the starting point is used depends on the optimization method. The starting point for sample variables is irrelevant. Fixed variables will remain at their starting point during the optimization.

The sample variables define a set of samples that will be used to calculate an objective function. For each objective function, the decision variables are fixed at values set by the optimization solver, and the flowsheet is evaluated for each row on the sample table. The results of the samples can be used to calculate the objective function. Using the **Samples** tab is optional. If no sample variables are set, each objective function value will be based on a single simulation. Figure *Optimization Sample Table* shows the Samples table form.

Fig. 2: Optimization Sample Table

1. The **Samples** tab contains the table used to define samples for objective function calculations. If there are no sample variables, the table should be empty.
2. **Add Sample** adds a row to the Samples table.
3. **Delete Samples** deletes the selected rows from the Samples table.
4. **Generate Samples** opens a dialog box that provides a selection of methods to generate samples or read samples from a file.
5. **Clear Samples** clears the Samples table.

Once the variables and (optionally) samples have been selected, the objective function and constraints can be defined. FOQUS is set up to handle multi-objective optimization, but no multi-objective optimization plug-ins are currently provided in the FOQUS installer, so some of the options may seem to be extraneous. There are two methods for entering the objective function and constraints into FOQUS: (1) Simple Python expressions and (2) a more extensive Python function. Python expressions are easier and sufficient for most cases. If the objective function is complicated it may be necessary to write a Python function, which can be as complex as needed.

The variables used in the Python code for the objective function or constraints are stored in two Python dictionaries, “f” for outputs and “x” for inputs. There are two ways to index the dictionaries depending on whether or not sample variables are used. For an input variable with sampling, the indexing is `x[Sample Index][Node Name][Variable Name][Time Step Index]`. If no sample variables are defined, the sample index is not needed, so the indexing would be, `x[Node Name][Variable Name][Time Step]`. Node Name and Variable Name are strings so they should be in quotes. The sample and time step indexes are integers. For steady state simulations, the time step should be 0.

Figure *Optimization Simple Objective Function* shows the form for entering the objective function and constraints as Python expressions.

Fig. 3: Optimization Simple Objective Function

1. The **Objective/Constraints** tab contains the form used to enter the objective function and constraints.
2. The drop-down list enables the selection of either the “Simple Python Expression” or “Custom Python” form of the objective function.

3. + adds an objective function to the table. The solvers currently available are single objective and will only use the first objective function.
4. - removes the selected objective from the table.
5. The Python expression for the objective function can be entered in the **Expression** column.
6. The **Penalty Scale** column is intended for use with multi-objective solvers and allows the constraint violation penalty to be applied differently to objective functions with different magnitudes.
7. The **Value for Failure** column contains the value to be assigned to the objective function if the objective cannot be evaluated for any reason. The value should be higher than the expected highest value for a successful objective.
8. + adds an inequality constraint.
9. - removes selected inequality constraints.
10. The inequality constraints are in the form $g(\mathbf{x}) \leq 0$. The **Expression** column contains the Python expression for $g(\mathbf{x})$.
11. The **Penalty Factor** contains the coefficient a used in calculating the penalty for a constraint violation, see Equations (4.1) to (4.3).
12. The **Form** column contains a selection of different methods to calculate a constraint penalty.
13. **Check Input** checks the problem for any mistakes that can be detected before running the optimization.
14. **Variable Explorer** enables the user to browse the variables in the simulation. They can be copied and pasted into the Python expression. The variables are provided without the sample index.

The calculations for each type of constraint penalty are given in Equations (4.1) to (4.3).

$$\text{Linear penalty form: } p_i = \begin{cases} 0 & \text{if } g_i(\mathbf{x}) \leq 0 \\ a \times g_i(\mathbf{x}) & \text{if } g_i(\mathbf{x}) > 0 \end{cases} \quad (4.1)$$

$$\text{Quadratic penalty form: } p_i = \begin{cases} 0 & \text{if } g_i(\mathbf{x}) \leq 0 \\ a \times g_i(\mathbf{x})^2 & \text{if } g_i(\mathbf{x}) > 0 \end{cases} \quad (4.2)$$

$$\text{Step penalty form: } p_i = \begin{cases} 0 & \text{if } g_i(\mathbf{x}) \leq 0 \\ a & \text{if } g_i(\mathbf{x}) > 0 \end{cases} \quad (4.3)$$

If the Simple Python Expression method of entering the objective function does not offer enough flexibility, the Custom Python method can be used. The Custom Python method enables the user to enter the objective calculation as a Python function, which also should include any required constraint penalties.

Figure *Custom Objective Function* shows the Custom Python objective form. The top text box provides instructions for writing a custom objective function. The bottom text box provides a place to enter Python code. The numpy and math modules have been imported and are available as numpy and math. To use the Custom Python objective, the user must define a function called “objfunc(x, f, fail).” The three arguments are: (1) “x” is the dictionary of input variables, (2) “f” is the dictionary of output variables, and (3) “fail” is a boolean vector that indicates whether a particular sample calculation has failed. The “objfunc” function should return three values: (1) a list of objective function values for multi-objective optimization (in most cases with single objective optimization this will be a list with one value), (2) a list of constraint violations, and (3) the total constraint penalty. The constraint violation and penalty information are only used for debugging, so they are not required. It is safe to return [0] and 0 for the constraint information regardless of whether a constraint penalty has been added to the objective.

Fig. 4: Custom Objective Function

The code in Figure *Objective Function Code* provides an example of a custom objective function for parameter estimation. The objective function minimizes the sum of the differences between simulation and empirical data. In this case

the decision variables would be model parameters. The first line defines a function with three arguments. The “x” and “f” arguments are the input and output variables. The variable indexing is explained in the simple objective function section. The “fail” argument is a boolean vector where element “i” is true if sample “i” failed. If there are no sample variables, “fail” will only have one element.

The “if” in the function determines if any flowsheet evaluation failed, and assigns a bad objective function value if so. If all the flowsheet evaluations were successful, the results are used to calculate the objective function. In the objective function calculation, Python list comprehension is used to calculate the sum of squared errors. In this case, no constraint penalty is needed. The objective function is returned as a list with only one element. The last two return values are debugging information for constraints. In this case, the “zeros” are just place holders and have no real utility.

Listing 1: Objective Function Code

```
def objfunc(x, f, fail):
    if any(fail): # any simulation failed
        obj = 1000000
    else: #simulations successful
        obj=sum([(f[i]['Test']['y'][0] - x[i]['Test']['ydata'][0])**2\
                for i in range(len(f))])
    return [obj], [0], 0
```

Solver Options

The **Solver** tab in the **Optimization** button tool enables the selection of the DFO method and setting of solver parameters. Figure *Optimization Solver Form* illustrates the solver form.

Fig. 5: Optimization Solver Form

Elements of the solver form are:

1. **Select Solver** drop-down list, which enables the user to select from available DFO solvers.
2. **Description** text box provides a description of the selected DFO solver.
3. **Solver Options** table contains the solver settings and a description of each option. The settings depend on the selected plug-in.

Running Optimization

The optimization monitor is displayed under the **Run** tab in the **Optimization** button tool. The optimization monitor, illustrated in Figure *Optimization Monitor Form*, is used to monitor the progress of the optimization as it runs.

Fig. 6: Optimization Monitor Form

Elements of the optimization monitor are:

1. **Start** starts the optimization.
2. **Stop** stops the optimization. The best solution found when optimization is stopped is stored in the flowsheet.
3. **Update delay** is how often the user interface communicates with the optimization thread to update the display.
4. **Optimization Solver Messages** displays output from the optimization solver.

5. **Best Solution Parallel Coordinate Plot** displays the values of the decision variables scaled. This plot is helpful in identifying when variables are at, or near, their bounds.
6. **Objective Function Plot** displays the objective function value at each iteration.
7. **Status Box** displays the current iteration, how many samples have been run, how many sample were successful, and how many failed.
8. **Clear** deletes solver messages from the solve message box.

As the optimization runs, the FOQUS flowsheet is updated to include the best solution found. If sampling is used, the first sample in the best objective function is stored in the flowsheet. If for any reason the optimization terminates, the best solution found is available in the flowsheet. The results for all flowsheet evaluations done for the optimization are available in the Results table in the Flowsheet Editor.

4.1.2 Tutorial

Tutorial 1: Optimization

This tutorial is a step-by-step walk through of simulation-based optimization. This tutorial builds on the tutorial in Section *Tutorial 2: Creating a Flowsheet with Linked Simulations*.

The files for this tutorial are located in: `examples/test_files/Optimization/Model_Files`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

1. Open FOQUS.
2. Load the FOQUS session from the tutorial “Creating a Flowsheet with Linked Simulations” in Section *Tutorial 2: Creating a Flowsheet with Linked Simulations* or if that tutorial has not yet been completed, complete it first.

Problem Set Up

If the simulation runs successfully and the results are reasonable, proceed to define the optimization problem. There are four steps to setting up the optimization problem: (1) select the variables, (2) define samples (optional), (3) define the objective function, and constraints and (4) select and configure the solver.

3. Select the **Optimization** button from the toolbar at the top of the Home window (Figure *Optimization Problem Variables*). Select the **Variables** tab.
4. Select “Decision” from the drop-down list in the **Type** column as the variable type for all 17 variables shown. If more than 17 variables are shown, the edge connecting the “BFB” node to the “Cost” node was most likely not configured properly. The scale will automatically change to linear, which is acceptable for most problems.
5. The **Min**, **Max**, and **Value** columns can be changed. The **Min** and **Max** columns define the lower and upper bounds. The **Value** column specifies the initial point. For this example the defaults are acceptable.

Fig. 7: Optimization Problem Variables

If more than one flowsheet evaluation is used in the objective function calculation (e.g., parameter estimation or optimization under uncertainty), the next step is to setup the samples under the **Samples** tab. In this case only one evaluation is used to calculate an objective function value, so the sample setup is not needed. The next step is to define the objective function and constraints using the form under the **Objective/Constraints** tab as shown in Figure *Optimization Problem Objective*.

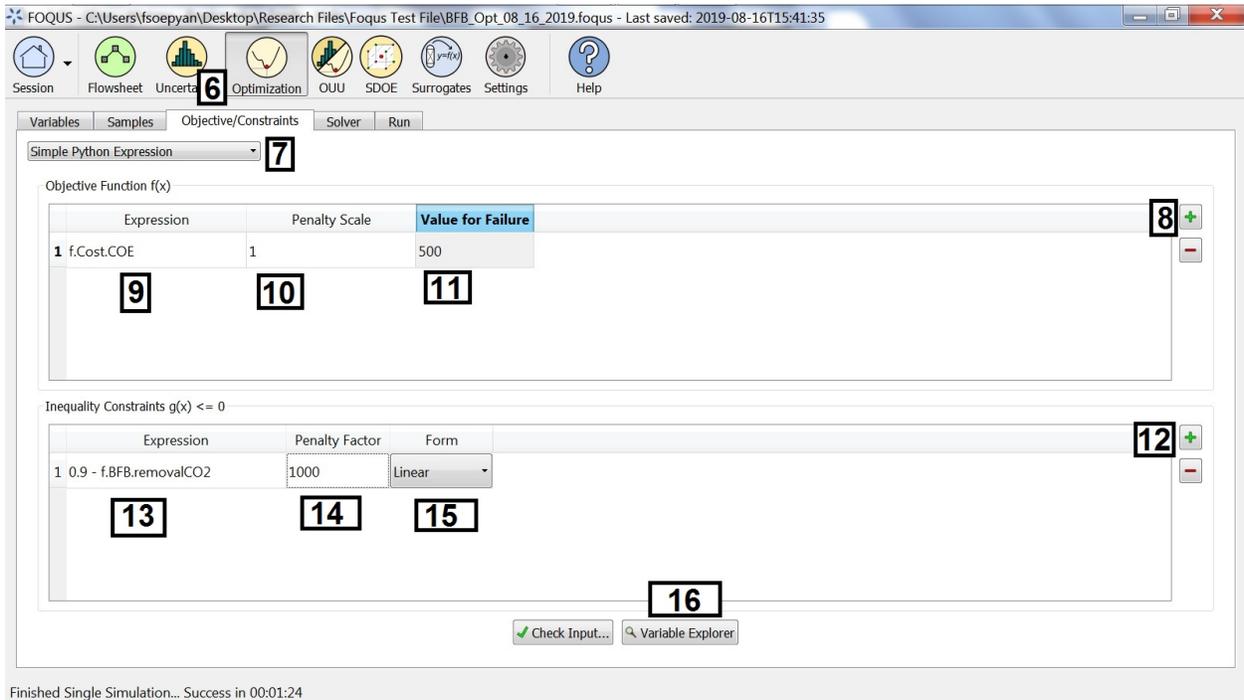


Fig. 8: Optimization Problem Objective

6. Select the **Objective/Constraints** tab (see Figure *Optimization Problem Objective*).
7. In the drop-down list, verify “Simple Python Expression” is selected.
8. Add an objective function by clicking + to the right of the Objective Function table.
9. The objective function is the cost of electricity from the cost spreadsheet. Enter:
 - f.Cost.COE
 - in the **Expression** column.
10. Enter 1 in the **Penalty Scale** column. This setting is used mostly for multi-objective optimization to apply the constraint penalty to different objectives.
11. Enter 500 in the **Value for Failure** column. This should be worse than the objective for any non-failed simulations.
12. Add a constraint by clicking + next to the Inequality Constraints table.
13. The constraint is that the fraction of CO₂ captured must be greater than or equal to 0.9. The constraint is in the form $g(x) \leq 0$; therefore, in the **Expression** column enter:
 - 0.9 - f.BFB.removalCO2.
14. Enter 1000 for the **Penalty Factor**.
15. The constraint penalty **Form** should be linear.
16. The **Variable Explorer** button can be used to help select flowsheet variables.

Solver Settings

The last step before running the optimization is to select and configure the solver. The solver configuration form is shown in Figure *Optimization Solver Setup*.

Fig. 9: Optimization Solver Setup

17. Select the **Solver** tab (see Figure *Optimization Solver Setup*).
18. Select “OptCMA” from the **Select Solver** drop-down list.
19. The default options are acceptable. Solver options are described in the Solver Options table.

Running Optimization

The optimization run form is shown in Figure *Optimization Monitor*.

Fig. 10: Optimization Monitor

20. Click the **Run** tab to display the optimization run form (see Figure *Optimization Monitor*).
21. Click **Start**.
22. Once the optimization has run for while click **Stop**.

As the optimization run, the best result found is stored in the Flowsheet. If an optimization is run with sample variables the first sample in the set with the best objective function will be stored in the flowsheet. All simulation results can be viewed in the Flowsheet Results table.

The run form displays some diagnostic information as the optimization runs. The parts of the display labeled in Figure *Optimization Monitor* are described below.

23. The Optimization Solver Messages window displays information from the solver.
24. The **Best Solution Parallel Coordinate Plot** shows the value of the scaled decision variables, which is useful to see where the best solution is relative to the variable bounds.
25. The **Objective Function Plot** shows the best value of the objective function found as a function of the optimization iteration or sample number.
26. While the optimization is running, the status bar shows the amount of time that has elapsed since starting the optimization.

Tutorial 2: Parameter Estimation

Note: The NLOpt solvers are used for the tutorial, but are an optional to the installation. See the install instructions for more information about installing NLOpt.

This tutorial provides a very simple example of using the sampling with optimization. Sampling can be used to do optimization under uncertainty where there are several scenarios with differing values of uncertain parameters. Sampling can also be used to do parameter estimation, where estimated values must be compared against several data points. This tutorial will focus on parameter estimation.

At any point in this tutorial, the FOQUS session can be saved and the tutorial can be started again from that point.

The model is given by Equation (4.4). The unknown parameters are a , b , and c . The x and y data are given in Table *x-y Data*.

$$y = ax^2 + bx + c \quad (4.4)$$

Table 1: x-y Data

Sample	1	2	3	4	5
x	0	1	2	3	4
y	1	0	3	10	21

The first step is to create a flowsheet with one node. The node will have the input variables: a , b , c , x , and $ydata$; and output variable y .

1. Open FOQUS.
2. In the **Session Name** field, enter “PE_tutorial” (see Figure *Session Setup*).
3. Click the **Flowsheet** button in the top toolbar.

Fig. 11: Session Setup

4. Add a node to the flowsheet named “model.”
 1. Click **Add Node** in the left toolbar (see Figure *Adding Node and Inputs*).
 2. Click anywhere on the gridded flowsheet area.
 3. Select “model” in the **Name** drop-down list and then click **OK**.
5. Click the **Selection Mode** icon in the left toolbar (see Figure *Adding Node and Inputs*).
6. Click the **Node Editor** icon in the left toolbar (see Figure *Adding Node and Inputs*).
7. In the Node Edit input table, add the variables a , b , c , x , and $ydata$. The $ydata$ variable will be used as an input for the known y sample point data, later in the tutorial.
 1. Click the **Add Input** icon (see Figure *Adding Node and Inputs*).
 2. Enter “ a ” for the variable name in the **Name** column.
 3. Enter -10 and 10 for the min and max in the **Min** and **Max** columns for a , b , c , and x .
 4. Repeat for all of the inputs.
 5. Enter 1 for the value of a , b , and c in the **Value** column.
 6. Enter 2 for the value of x in the **Value** column.
 7. The **Value**, **Min**, and **Max** for $ydata$ do not matter.
8. Click **Output Variables** (see Figure *Adding Outputs*).
9. Add the output variable y .
 1. Click the **Add Output** icon (see Figure *Adding Outputs*).
 2. Enter “ y ” for the variable name in the **Name** column.
10. Add the model equation to the node.
 1. Click the **Node Script** tab.

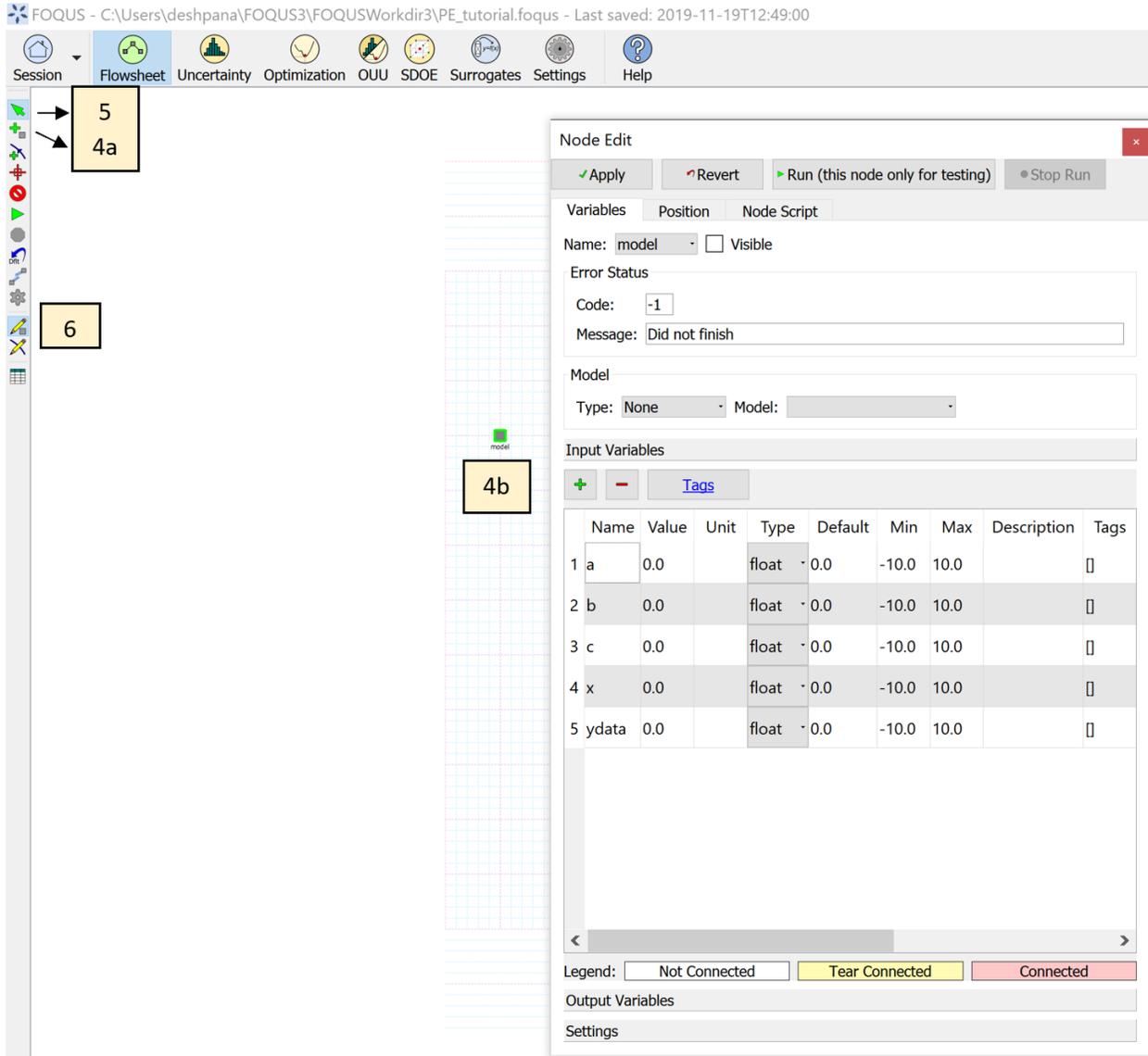


Fig. 12: Adding Node and Inputs

Fig. 13: Adding Outputs

2. Enter the following code in the calculations box:

```
f['y'] = x['a']*x['x']**2\
+ x['b']*x['x'] + x['c']
```

Fig. 14: Adding Node Calculation

11. Return to the Output Variables table in the Node Editor, by clicking on the **Variables** tab, and selecting **Output Variables**.
 12. Click **Run** in the left toolbar in the FOQUS Home window, to test a single flowsheet evaluation and ensure there are no errors.
 13. When the run is complete, there should be no error and the value of y should be 7 in the Output Variables table.
- The next step is to setup the optimization. The objective function is to minimize the sum of the squared errors between the estimated value of y and the observed value of y. There are five data points in Table *x-y Data*, so there are five flowsheet evaluations that need to go into the calculation of the objective.
14. Click the **Optimization** button in the top toolbar of the Home window (see Figure *Optimization Variables*).
 15. Select “Decision” in the **Type** column drop-down lists for “model.a,” “model.b,” and “model.c.” The **Scale** column will automatically be set to linear.
 16. Select “Sample” in the **Type** column drop-down lists for “model.x” and “model.ydata.”

Fig. 15: Optimization Variables

The decision variables in the optimization problem will be changed by the optimization solver to try to minimize the objective, and the sample variables are used to construct the samples that will go into the objective function calculation.

17. Select the **Samples** tab (see Figure *Optimization Samples*).
18. Click **Add Sample** five times to add five samples.
19. Enter the data from Table *x-y Data* in the Samples table.
20. For larger sample sets, **Generate Samples** has an option to load from a CSV file. The CSV file must be saved as “CSV (MS-DOS)” as the file type, as follows:

The objective function is the sum of the square difference between y and ydata for each sample in Table *x-y Data*. The optimization solver changes the a, b, and c to minimize the objective.

21. Click the **Objective/Constraints** tab.
22. Click the **Add Objective** icon on the right side of the Objective Function table (see Figure *Objective Function*).
23. In the **Expression** column, enter the following (without any line break):

```
sum([(ff.model.y - xx.model.ydata)**2 for (ff,xx) in zip(f,x)])
```

The above expression uses Python list comprehension to calculate the sum of squared errors.

The keys for x (the inputs) and f (the outputs) are:

- Dummy variable name for index (i.e., ff for outputs and xx for inputs)
- Node name (i.e., model)
- Variable name (i.e., y and ydata)

	A	B	C	D
1		model.x	model.ydata	
2	1	0	1	
3	2	1	0	
4	3	2	3	
5	4	3	10	
6	5	4	21	

Fig. 16: Sample Variable data (csv file)

Fig. 17: Optimization Samples

Then, the user will need to specify which of the dummy index corresponds to outputs, and which of the dummy index corresponds to inputs. In this case, ff is for the outputs, and xx is for the inputs. Hence, we have “for (ff,xx) in zip(f,x)” (without the quotes).

24. Enter 1 for the **Penalty Scale**.
25. Enter 100 for the **Value for Failure**.
26. No constraints are required.

Once the objective is set up, a solver needs to be selected and configured. Almost any solver in FOQUS should work well for this problem with the default values.

27. Click the **Solver** tab (see Figure *Optimization Samples*).
28. Select “NLOpt” from the **Select Solver** drop-down list. NLOpt is a collection of solvers that share a standard interface (*Johnson 2015*).
29. Select “BOBYQA” under the Solver Options table in the **Settings** column drop-down list.
30. Click the **Run** tab (see Figure *Running Optimization*).
31. Click the **Start** button.
32. The Optimization Solver Messages window displays the solver progress. As the solver runs, the best results found is placed into the flowsheet.
33. The **Best Solution Parallel Coordinate Plot** shows the scaled decision variable values for the best solution found so far.
34. The **Objective Function Plot** shows the value of the objective function as the optimization progresses.

The best result at the end of the optimization is stored in the flowsheet. All flowsheet evaluations run during the optimization are stored in the flowsheet results table.

35. Once the optimization has completed, click **Flowsheet** in the top toolbar.

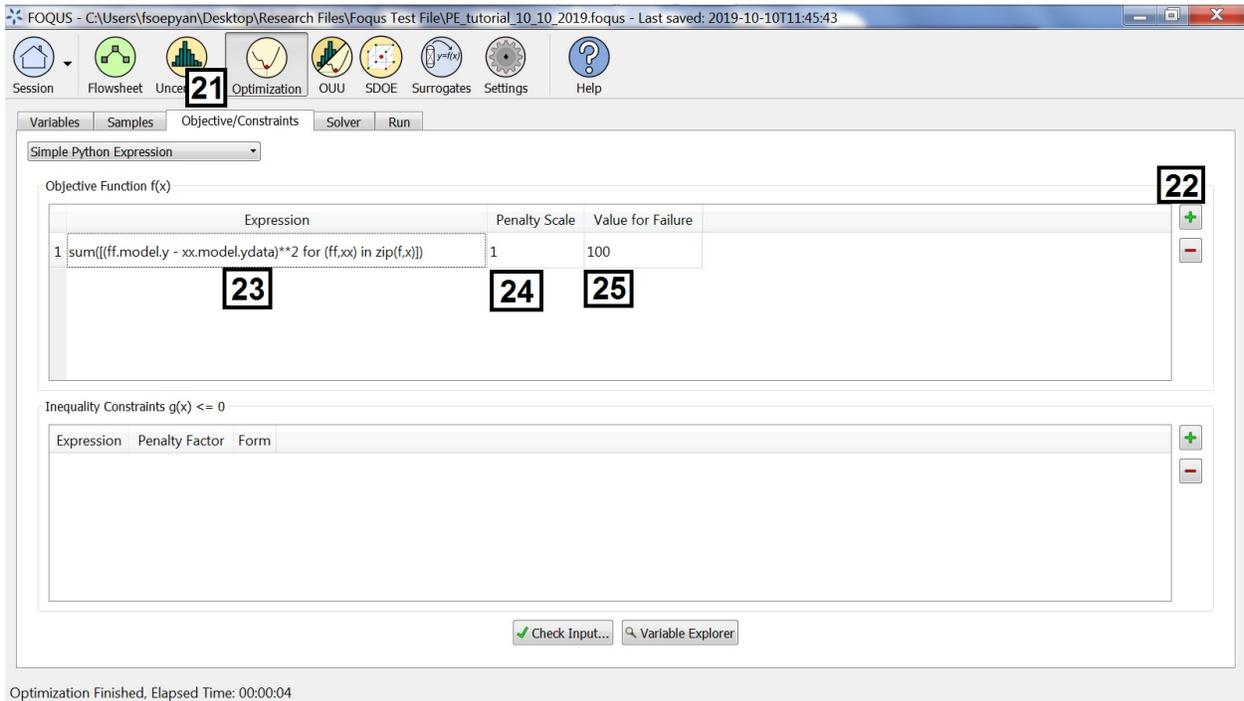


Fig. 18: Objective Function

Fig. 19: Optimization Samples

Fig. 20: Running Optimization

36. Open the **Node Editor** and look at the **Input Variables** table. The approximate result should be $a = 2$, $b = -3$, and $c = 1$ (see Figure *Flowsheet, Input Variables Results*).

Fig. 21: Flowsheet, Input Variables Results

UNCERTAINTY QUANTIFICATION (UQ)

5.1 Contents

5.1.1 Reference

The Uncertainty Quantification (UQ) module of FOQUS provides a multitude of analysis and visualization capabilities to facilitate the understanding of uncertainty's impact on a given system. In a generic UQ study, the workflow is usually comprised of the following steps:

1. Define the objectives of the analysis.
2. Specify and acquire the simulation model, which implements an input-to-output mapping from inputs to outputs.
3. Select the inputs that have uncertainty and characterize said uncertainty in the form of *prior* distributions.
4. Identify relevant data from physical experiments that can be used to refine these prior distributions on the inputs.
5. Generate a set of input samples according to the input distribution.
6. Propagate the set of input samples through the simulation model to get the corresponding output values.
7. Analyze the results to make informed decisions about subsequent analyses.

FOQUS UQ provides tools to perform Steps 5-7. With respect to Step 7, a variety of analysis capabilities are available. They include parameter screening methods, response surface construction/validation/prediction, uncertainty analysis, sensitivity analysis, and visualization.

In this chapter, components of the UQ user interface are first explained, then the use of these components for UQ analyses is illustrated.

UQ User Interface

The UQ module enables the user to perform UQ studies on a flowsheet. From the Uncertainty button on the Home window, the user can configure different simulation ensembles (different sets of samples generated using different sampling schemes), run them, and perform analyses. This screen is illustrated in Figure *Uncertainty Quantification Screen*.

1. **Simulation Ensemble Table** displays all of the simulation ensembles: each ensemble being a row in the table. A simulation ensemble is a collection of sample points where each sample point has a different set of values for the uncertain variables. The values of these variables are generated based on the sampling scheme designated by the user. When launched, the output values of the sample points are calculated based on the generated sample input values. Subsequently, the corresponding simulation outputs can be analyzed. For each ensemble, the table displays the **Ensemble** index, **Run Status** (how many have been completed), **Setup** and **Launch** options (discussed below), and a **Descriptor**. The **Descriptor** contains the name of the corresponding node in the flowsheet

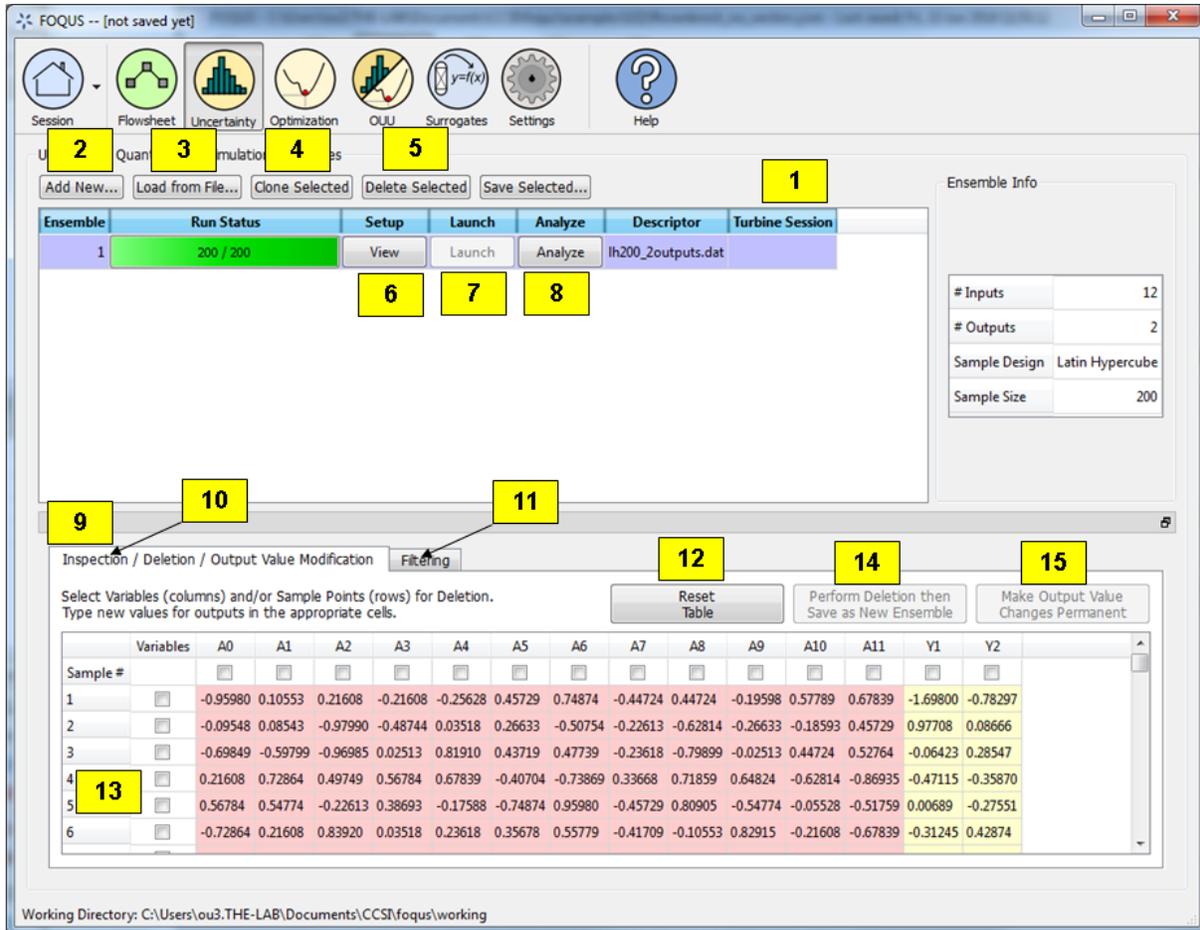


Fig. 1: Uncertainty Quantification Screen

or the name of the file if the ensemble was loaded from a file. Additional sample information such as **# Inputs**, **# Outputs**, **Sample Design**, and **Sample Size** are also displayed on the right.

2. **Add New** creates a simulation ensemble (a set of input samples) as a new row in the **Simulation Ensemble Table**. Once clicked, a dialog is displayed to prompt the user to choose between using (1) a flowsheet (an exact simulation model) or (2) a response surface (an approximate simulation model or an emulator) associated with the ensemble.
If using an emulator, the user must browse a PSUADE-formatted file that contains the training data for the emulator (in this version, the response surface type has been designated inside the sample file and can only be changed by editing the sample file) and select the output(s) to be evaluated by the trained emulator. Subsequently, a simulation setup dialog box is displayed for setting up the distributions of input variables and the sampling scheme to generate samples of the uncertain input variables. This **Simulation Ensemble Setup** dialog is explained in further detail in Section *Simulation Ensemble Setup Dialog*.
3. **Load from File** loads a simulation ensemble from a sample file that conforms to the PSUADE full file format. (See Section *File Formats* for details on the PSUADE full file format.) The user can click **Save Selected** to save an existing ensemble as a PSUADE full file, and load it by clicking **Load from File** to perform further analyses.
4. **Clone Selected** clones the selected simulation ensemble and adds the copy as a new row at the end of the table. This ensemble can then be edited (e.g., depending on whether the ensemble has been run, the user has different options for modifying the ensemble). This allows the user to create a new ensemble similar to the current ensemble without having to start from scratch (i.e., setting up the input parameters). For example: (1) **Clone Selected** can be used in conjunction with **Load from File** to clone an existing ensemble before input/output modification to prepare a new but similar ensemble for UQ analysis. (2) **Clone Selected** can also be used to prepare a fresh ensemble for evaluation via a different simulation model. In this case, the user should save the cloned ensemble, reload it by clicking **Add New**, associate it with a node, and then click **Launch** to start the runs.
5. **Delete Selected** deletes the currently selected simulation ensemble.
6. **Revise** enables a user to change a simulation ensemble before launching the run. If the ensemble was previously run or it is cloned from an already-generated sample, the corresponding button becomes **View** so the user can view the input samples in the simulation ensemble.
7. **Launch** starts the simulation process of the ensemble. (**Launch** is not enabled until the user has setup everything needed for simulations.) A simulation is launched for each sample point to compute the corresponding outputs.
8. **Analyze**, when enabled (after all simulation results are ready), enables the user to perform various UQ analysis to the ensemble. When clicked, a new dialog box displays, allowing the user to configure and run analysis.
9. **Data Manipulation** enables (1) the deletion of inputs, outputs, or samples, (2) the modification of output values for specific sample points, and (3) the range-based filtering of samples.
10. **Inspection/Deletion/Output Value Modification** serves three purposes: it enables the user to (1) view the numerical values of samples in table form, (2) delete variables and/or samples, and (3) edit the output values of specific samples. **Deletion** creates a new simulation ensemble as a new row in the simulation table that contains only those inputs/outputs and samples that were not selected for deletion. **Output Value Modification** changes the values within the ensemble itself.
11. **Filtering** enables the user to filter samples based on the values of an input or output. First, select the ensemble to be filtered from the **Simulation Ensemble Table**. Once filtering is complete, a new simulation ensemble is added as a new row in the simulation table. The new simulation ensemble contains only those samples that satisfy the filtering criterion (with input or output samples within the specified range).
12. **Reset Table** resets the table to default, meaning all variable and sample selections are unselected and output values within the table are reverted back to their original values, thus undoing any edits to the table.
13. The table displays the values of inputs and outputs for each sample. Inputs are highlighted in pink; outputs are highlighted in yellow. The user can select which variables (columns) to delete by selecting the checkboxes on top. Likewise, the user can select which samples (rows) to delete by selecting the checkboxes on the left. Multiple samples can also be selected/deselected by using (1) Shift+Click or Ctrl+Click to select multiple rows, or (2)

right-clicking to bring up a menu to check or uncheck the checkboxes corresponding to the rows of the selected samples. In addition, the user can change any output value by editing the appropriate cell. These modified cells are highlighted green until changes are made permanent by clicking the appropriate button.

14. **Perform Deletion then Save as New Ensemble** creates a new simulation ensemble as a new row in the **Simulation Ensemble Table**. The new ensemble is without the variables and samples that were previously selected for removal.
15. **Make Output Value Changes Permanent** overwrites the output values in the current ensemble with those that are highlighted green in the table.

enumerate

The **Filtering** tab is illustrated in Figure *Filtering Tab* and enables the user to filter samples based on the values of an input or output.

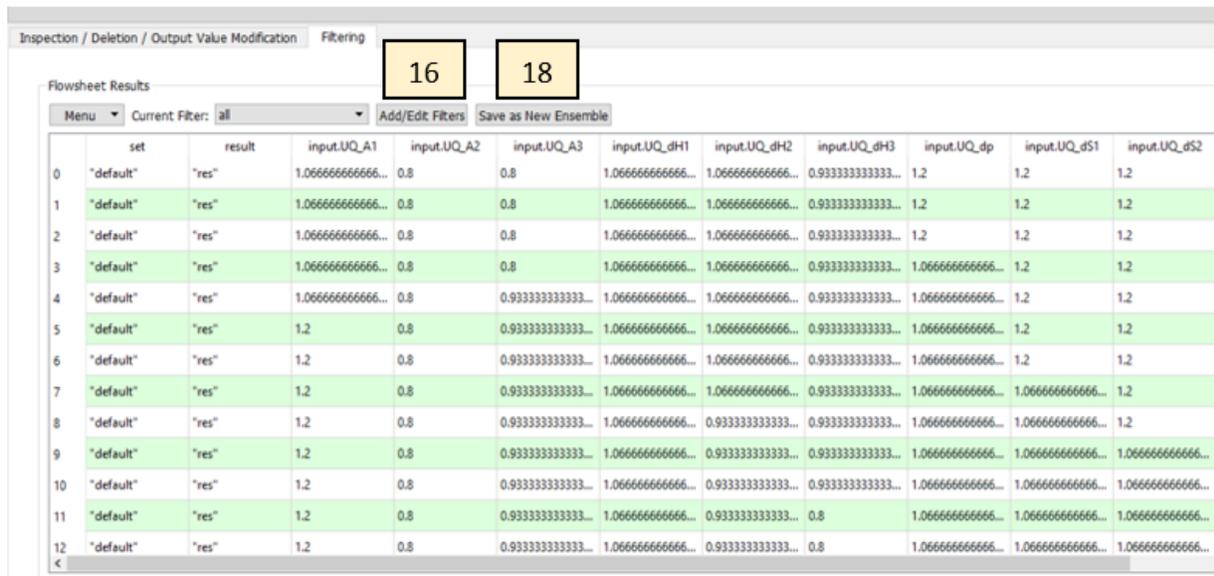


Fig. 2: Filtering Tab

enumerate

enumerate

16. Click on Add/Edit Filters, in the Flowsheet Results window within the “Filtering Tab”
17.
 1. Within the Filter Dialog Box, Click on “New Filter” to add a filter
 2. Enter a filter expression in python format. Variables can be dragged into the expression, from the “Columns”, click Done.
18. Select a “Current Filter” after which the the filtered ensemble can be saved by clicking on “ Save as New Ensemble”

The single-output **Analysis of Ensemble** dialog, which is displayed when **Analyze** is clicked for the selected ensemble, has two modes, as shown in Figure *Analysis Dialog, Ensemble Data Analysis, Wizard Mode* and Figure *Response Surface Based Mixed Epistemic-Aleatory Uncertainty Analysis*.

enumerate

19. Select **Wizard** or **Expert** mode. The **Wizard** mode provides more detailed guidance on how to perform UQ analysis. For users familiar with UQ analysis techniques, the **Expert** mode provides more functionality and

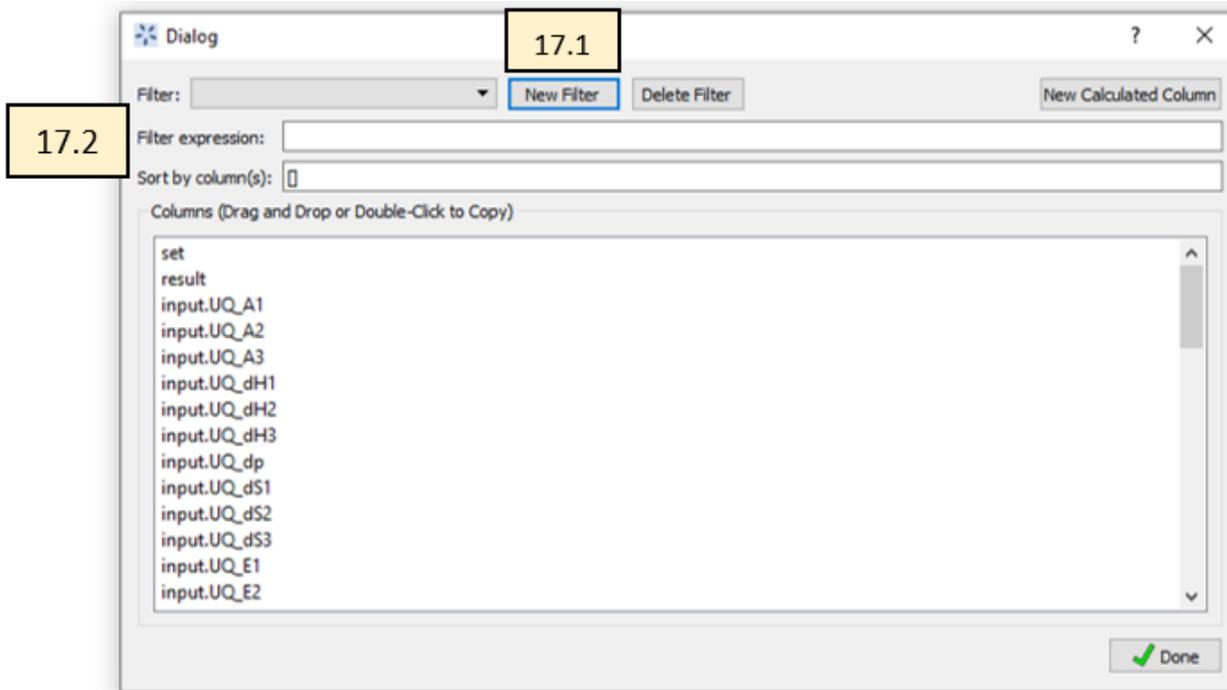


Fig. 3: Filtering Dialog Box

flexibility but with less guidance on its use. For example, users will be able to customize the input distributions, as well as run more advanced uncertainty analysis that handles both epistemic and aleatory uncertainties.

20. The **Analyses Performed** section provides the user a history of previous analyses that were performed. The results of these analyses are cached, so the user can plot the analysis results without having to recompute them.
21. The **Analysis Table** populates as the user performs analyses. It lists previous analyses that the user has performed, along with some of the main analysis settings (analysis type, inputs and outputs analyzed, etc.)
22. Depending on the type of analysis performed, the **Additional Info** button displays any additional settings or parameters set by the user in the selected analysis that were not shown in the **Analysis Table**.
23. The **Results** button will display the results of the selected analysis.
24. The **Delete** button will delete the selected analysis from the history of previous analyses. Once deleted, the user will need to perform the analysis again to see its results.
25. The **Qualitative Parameter Selection** (top part of the **Analysis of Ensemble** dialog) houses the controls for parameter selection analysis. Parameter selection is a qualitative sensitivity analysis method that identifies a group of dominant input parameters that are recommended for inclusion in subsequent UQ analyses, as they are the ones that most impact the output uncertainty. The parameter screening results are shown as bar graphs so that the user can rank the uncertain parameters visually.
26. Before performing parameter selection, the user must select a single output for identifying parameter sensitivities from the **Choose output to analyze** drop-down list.
27. There are several methods of parameter selection. The list of parameter selection methods available depends on the sample scheme of the selected ensemble. Select the appropriate method from the **Choose Parameter Selection Method** drop-down list. Then click **Compute input importance** to start the analysis.
28. The **Ensemble Data** radio button directs FOQUS to perform analyses on the raw ensemble data.

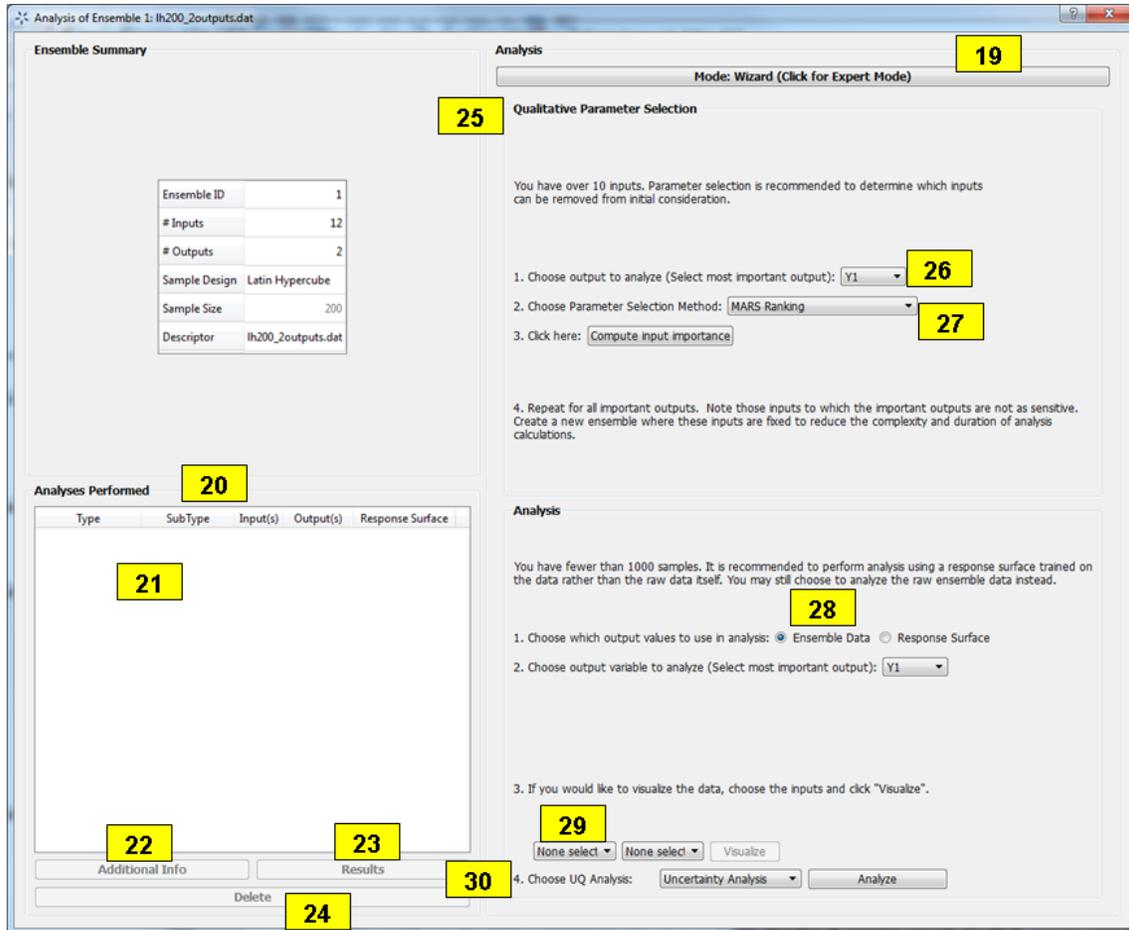


Fig. 4: Analysis Dialog, Ensemble Data Analysis, Wizard Mode

29. To view plots of the raw ensemble data, choose the desired input(s) from the **Select the input(s)** drop-down lists. Then click **Visualize**. If multiple inputs are selected, each must be unique.
30. To perform an analysis, select the desired analysis (“Uncertainty Analysis” or “Sensitivity Analysis”) from the **Choose UQ Analysis** drop-down list. Uncertainty Analysis computes and displays the probability distribution of the single selected output parameter and displays its sufficient statistics, such as mean, standard deviation, skewness, and kurtosis. Sensitivity Analysis computes and displays each uncertain input parameter’s contribution to the total variance of the output. If Sensitivity Analysis is selected, choose the type of sensitivity analysis desired in the next drop-down list. There are three options for Sensitivity Analysis: (1) first-order, (2) second-order, and (3) total-order.
 - First-order analysis examines the effect of varying an input parameter alone.
 - Second-order analysis examines the effect of varying pairs of input parameters.
 - Total-order analysis examines all interactions’ effect of varying an input parameter alone and as a combination with any other input parameters.

Click **Analyze** to run the analysis. (Note: Raw ensemble data analysis may not be suitable if the sample size is small. It may be useful if the data set has tens of thousands of sample points or if an adequate response surface cannot be constructed. Otherwise, response surface-based analyses are recommended.)

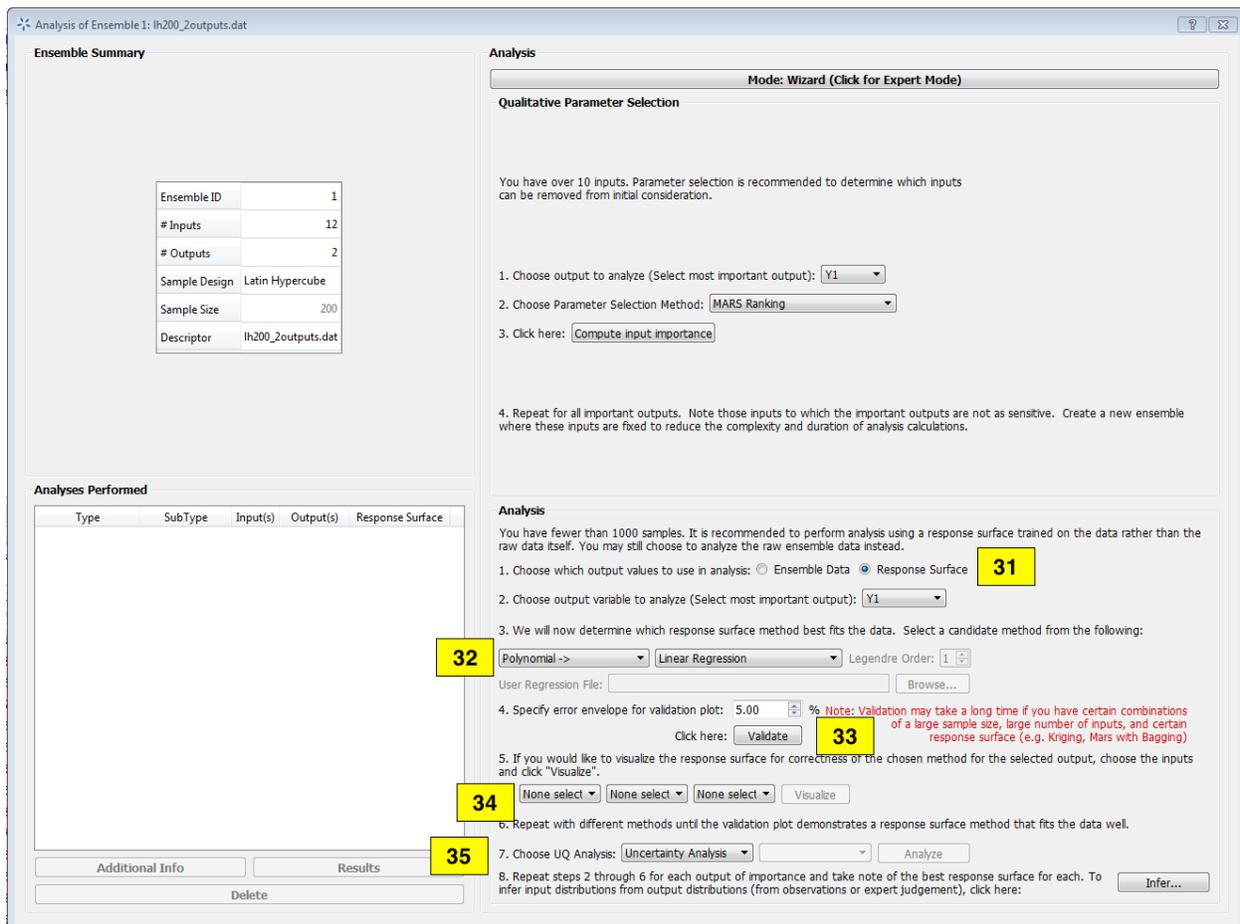


Fig. 5: Analysis Dialog, Response Surface Analysis, Wizard Mode

31. **Response Surface** enables the user to perform all analyses related to response surfaces. A response surface is an approximation of the input-to-output relationship. This is an inexpensive way to approximate the values of

outputs given different input values when the actual simulation of output values is computationally intensive. FOQUS uses the data (i.e., input-output samples) to fit a response surface scheme. The first step in this analysis is to select which output to analyze.

32. Select the **Response Surface Model** to be used to approximate the input-to-output mapping. Selection of “Polynomial” or “MARS” requires one further selection in the second drop-down list. If “Polynomial” is chosen in the first drop-down list and “Legendre” is chosen in the second drop-down list, the user needs to specify a number for the **Legendre polynomial order** before analysis can proceed.
33. The response surface selected must be validated before further analyses can be performed. The user can specify the error envelope for the validation plot. When **Validate** is clicked, the resulting plots display the best fit between the response surface (based on the model selected) and the actual data.
34. **Choose UQ Analysis** enables the user to perform response-surface-based UQ analyses. Select the analysis in the first drop-down list. If the desired analysis is Sensitivity Analysis, select the desired type of sensitivity analysis in the second drop-down list and then click **Analyze**. **Uncertainty Analysis** and **Sensitivity Analysis** compute and display the same quantities as in item 30. However, the results displayed are based on samples drawn from the trained response surface, not the simulation ensemble itself. Moreover, if the selected response surface has uncertainty, the resulting plots also reflect this uncertainty information.
35. FOQUS also provides visualization capabilities, enabling the user to view the response surface as a function of one or multiple inputs. Up to three inputs can be visualized at once. Click **Visualize** to view. A 2-D line plot displays if only one input parameter is selected. A 3-D surface plot and a 2-D contour plot display if two input parameters are selected. A 3-D isosurface plot with a slider bar displays if three input parameters are chosen. For the isosurface plot, the user can use the slider to selectively display the 3-D input parameter space that activates a particular range in the output parameter.

Finally, the **Bayesian Inference of Ensemble** dialog (shown in Figure *Bayesian Inference Dialog*) is used to calculate the posterior distributions (prior distributions integrated with data) of the uncertain input parameters. Inference utilizes Markov Chain Monte Carlo (MCMC) to compute the posterior distributions, using response surfaces that serve as fast approximations to the actual simulation model.

enumerate

36. Inference uses a response surface to approximate the input-to-output mapping. In **Output Settings**, select the observed outputs and select the response surface type that works best with each observed output. As in item 32, further selections may be required based on the response surface chosen. The simulation ensemble is used as the training data for generating the response surfaces.
37. The user can specify which inputs are fixed, design (fixed per experiment, but changes between experiments), or variable using the **Input Settings Table**. In addition, the user can specify which inputs are displayed in the resulting plots of the posterior distributions. By default, once inference completes, all inputs will be displayed in the plots. To omit specific inputs, clear the checkboxes from the **Display** column of the table. Finally, in **Expert** mode, this table can also be used to modify the input prior distributions. The default prior is the input distribution specified in the simulation ensemble. To change the prior distribution type, use the drop-down list in the **PDF** column and enter corresponding values for the PDF parameters. To change the range of a uniform prior, scroll all the way to the right to modify **Min/Max**.
38. The **Observations** section enables the user to add experimental data in the form of observations of certain output variables. At least one observation is required. Currently, the observation noise model is assumed to be a normal distribution. Other distributions may be supported in the future. To specify the observation noise model, enter the mean (and standard deviation, if standard inference is selected) for each output observation. For convenience, the **Mean** and **Standard Deviation** fields have been populated with the statistics from the ensemble uncertainty analysis. If any inputs are selected as design inputs, their values will also be required here.
39. **Save Posterior Input Samples to File** checkbox, when selected, saves the posterior input samples as a PSUADE sample file (format described in Section *File Formats*). This file characterizes the input uncertainty as a set of samples, which can be re-used in the **Simulation Ensemble Setup** dialog, to evaluate the outputs corresponding to these posterior input samples.

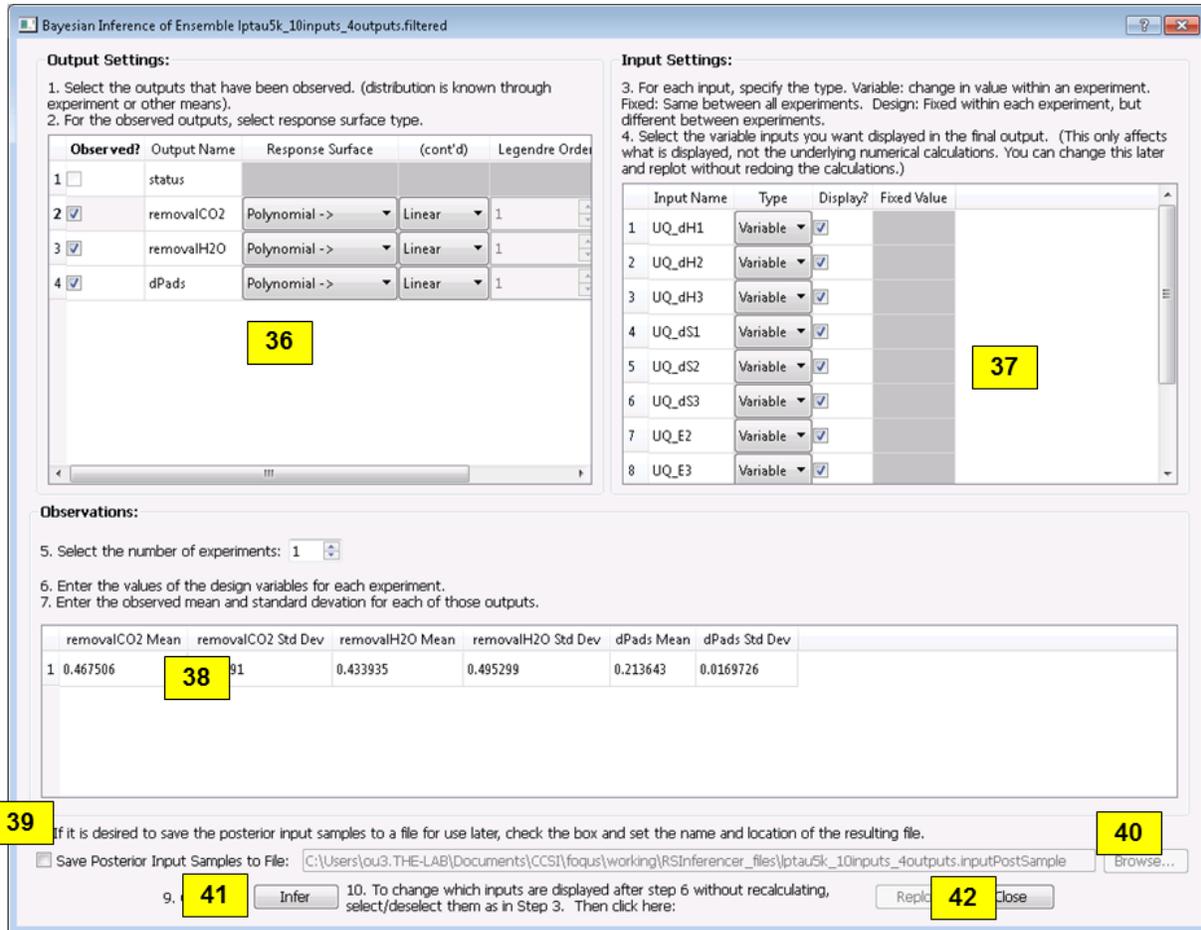


Fig. 6: Bayesian Inference Dialog

40. If saving posterior samples to a file, click **Browse** to set the name and location of where this file is saved.
41. Click **Infer** to start the analysis. (Note: If the inference returns an invalid posterior distribution (i.e., one with no samples), it usually means the prior distributions or that the observation data distributions are not prescribed appropriately. In this case, it is recommended that the user experiment with different priors and/or data distribution means and/or standard deviations.)
42. Inference calculations often take a very long time. If inference has run to completion, use Replot to generate new plots (e.g., to only display a subset of the input posterior graphs) from the cached inference results.

Simulation Ensemble Setup Dialog

The **Simulation Ensemble Setup** dialog (shown in Figure *Simulation Ensemble Setup Dialog, Distributions Tab*) is used to create a new simulation ensemble. This is done by: (1) setting up distribution parameters and generating samples, or (2) loading samples from a file. This dialog is displayed when selecting **Add New** on the UQ window (Figure *Uncertainty Quantification Screen*).

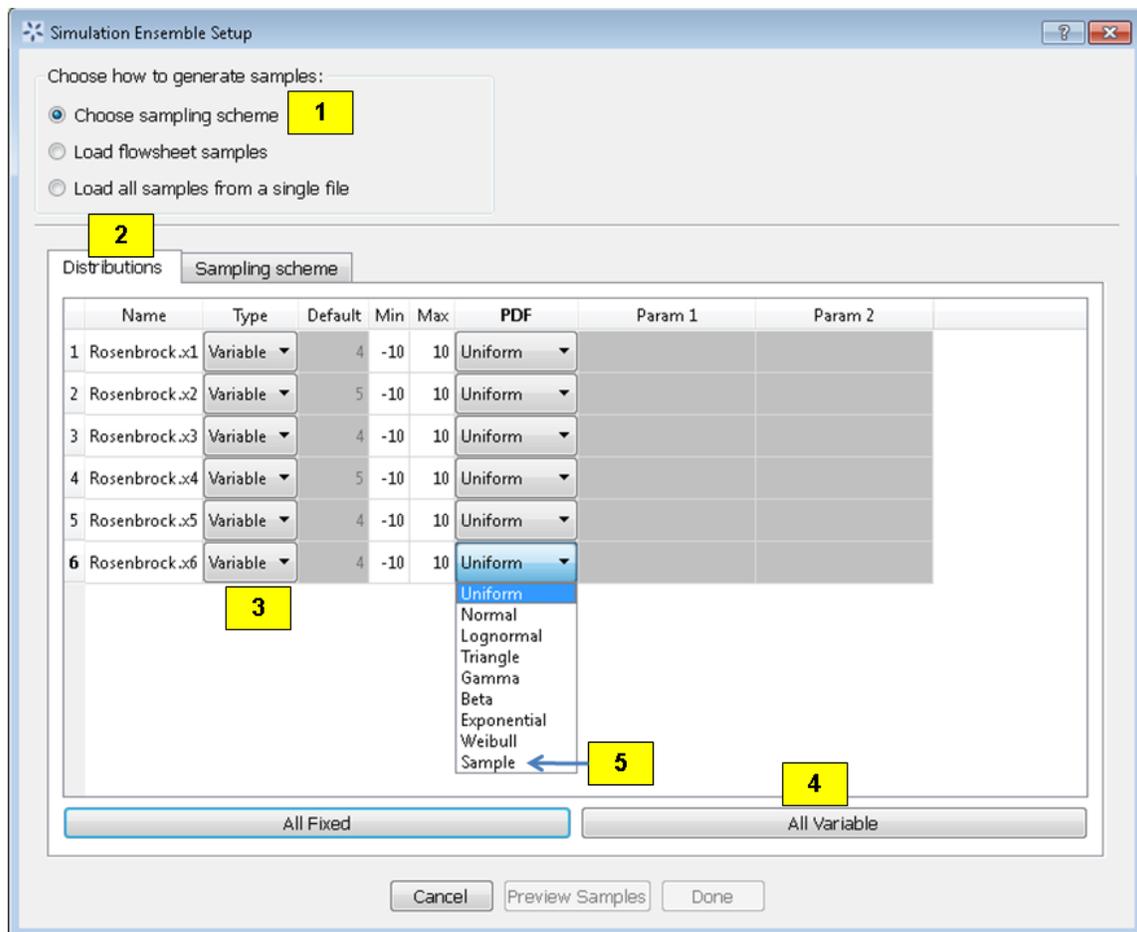


Fig. 7: Simulation Ensemble Setup Dialog, Distributions Tab

1. Choose how to generate samples. There are three options: (1) **Choose sampling scheme** (default), (2) **Load flowsheet samples**, or (3) **Load all samples from a single file**. The option 3 is explained in item 11.
2. If **Choose Sampling Scheme** is selected, the **Distributions** tab is displayed. The user specifies the input uncertainty information.

3. The **Distributions Table** is pre-populated with input variable information gathered from the flowsheet node. Under the **Type** column drop-down list, the user can select “Fixed” or “Variable”. Selecting “Fixed” means that the input is fixed at its default value for all the samples. Changing the type to “Variable” means that the input is uncertain; therefore, its value varies between samples. With any fixed input, the only parameter that can be changed is the **Default** value (i.e., all samples of this input are fixed at this default value). With any variable input, the **Min/Max** values, as well as the probability distribution function (**PDF**), for that input can be changed. Some PDFs have their own parameters (e.g., mean and standard deviation for a normal distribution), which are required in the columns right of the distribution column. See the PSUADE manual for more details on the different PDFs.
4. **All Fixed** and **All Variable** are convenient ways to set all the inputs to variable or fixed.
5. Note: A “Sample” PDF refers to sampling with replacement (i.e., input samples would be randomly drawn, with replacement, from a sample file). If the selected distribution for any input is “Sample”, then the following parameters are required: (1) the path of the sample file (which must conform to the sample format specified in Section *File Formats*); (2) the output index that designates which output is to be used.
6. In the **Sampling scheme** tab (Figure *Simulation Ensemble Setup Dialog, Sampling Scheme Tab*), specify the sampling scheme, the sample size, and perform sample generation.

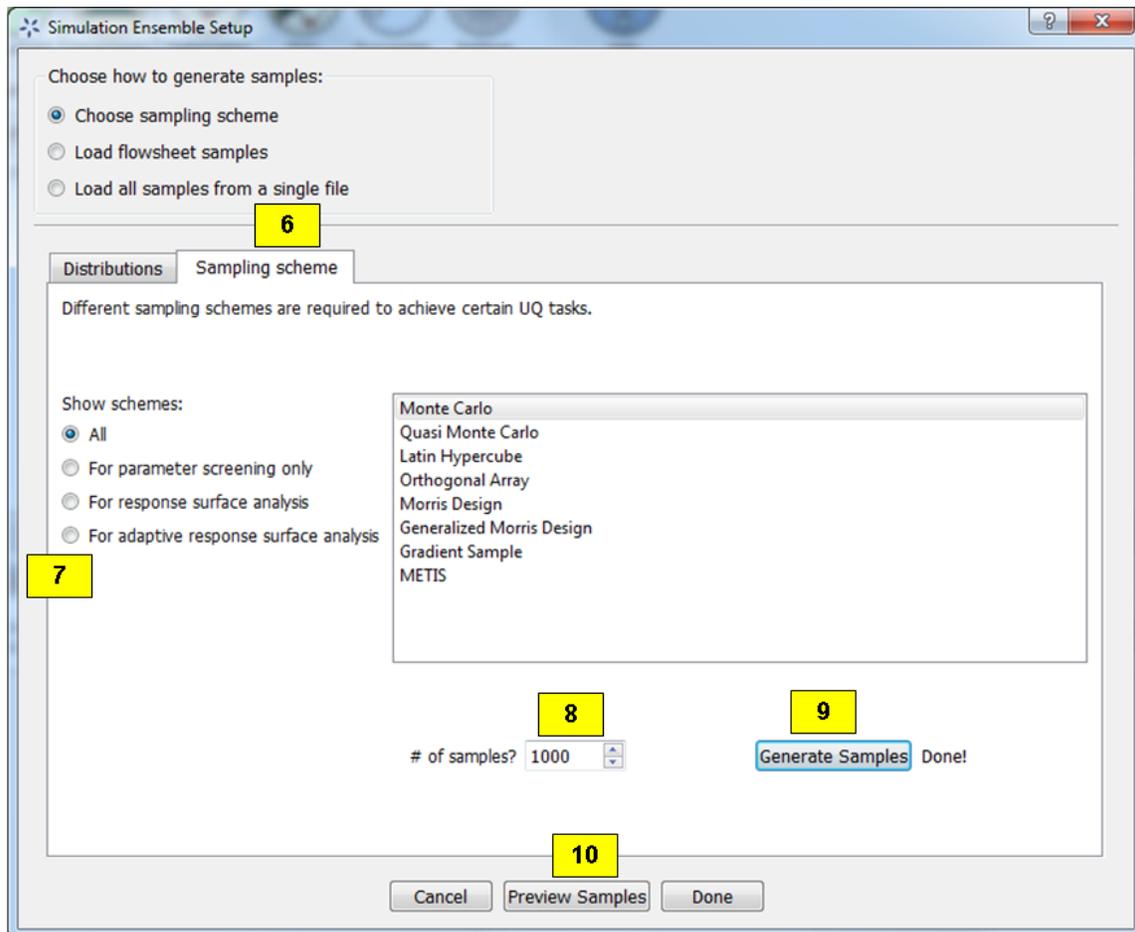


Fig. 8: Simulation Ensemble Setup Dialog, Sampling Scheme Tab

7. Each radio button displays a different list of sampling schemes on the right. The radio buttons serve as a guide to help in the selection of the appropriate sampling schemes for target analyses. A sampling scheme must be selected from the list on the right to proceed.
8. Set the number of samples to be generated from the **# of samples** spinbox.

9. When all parameters are set, click **Generate Samples**. This generates the values for all the input variables, based on the sampling scheme selected.
10. Once samples have been generated, click **Preview Samples** to view the samples that were generated. This displays the sample values in table form, as well as graphically as a scatter plot.
11. From item 1, if the user elects to load all samples from a single file, click **Browse** to select the file containing the samples (Figure *Simulation Ensemble Setup Dialog, Load Samples Option*). This file must conform to the PSUADE full file format, the PSUADE sample format, or CSV file (all formats described in Section *File Formats*). Note: This is the only place where all the formats are supported. Once the file is loaded, the file name displays in the text box. These samples can now be used in the same way as an ensemble that was newly generated (as described above).

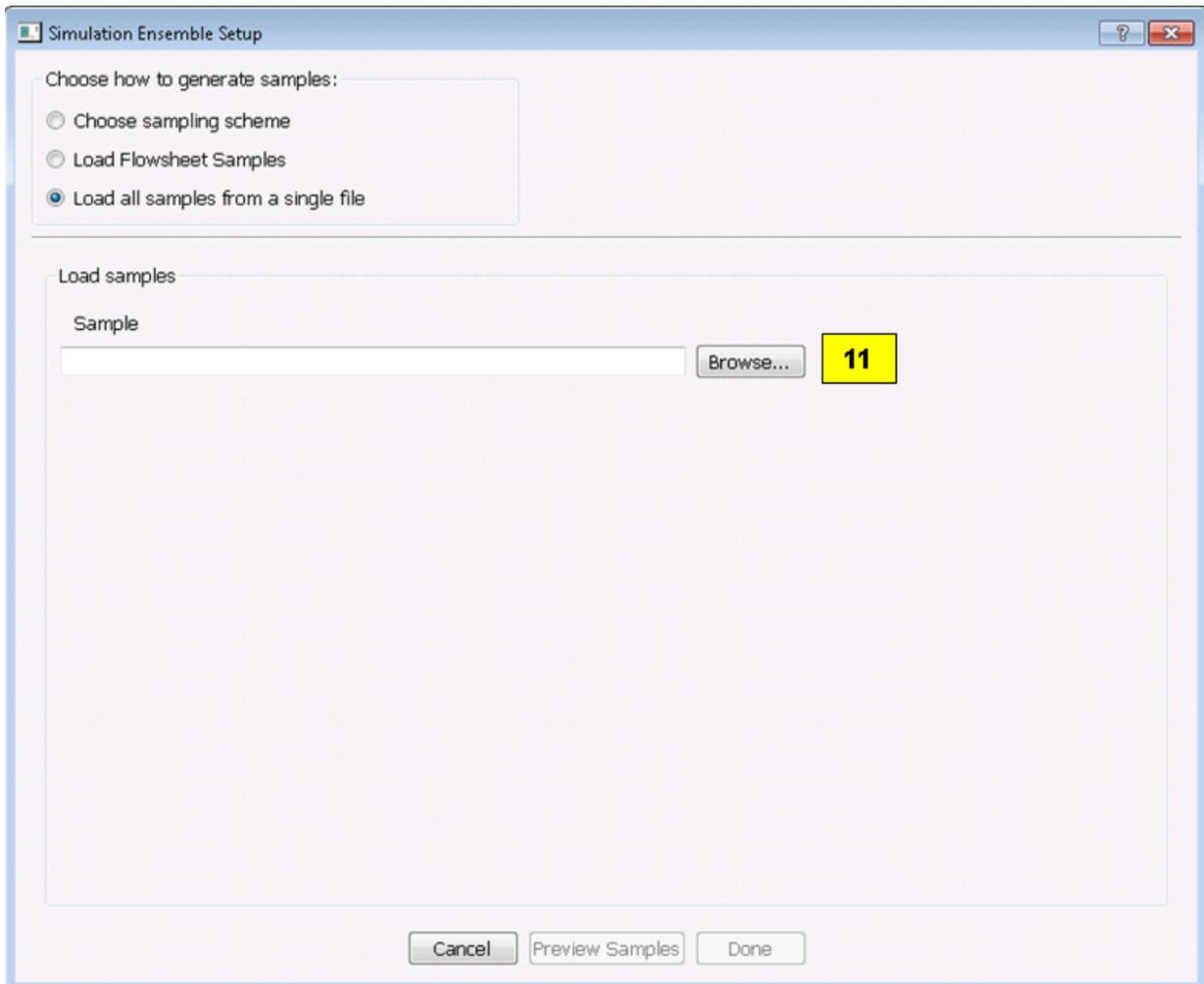


Fig. 9: Simulation Ensemble Setup Dialog, Load Samples Option

5.1.2 Tutorials

This section contains five tutorials that illustrate the use of FOQUS UQ to facilitate the UQ workflow discussed above. Each tutorial will refer to example files located in the examples directory of the FOQUS download.

Tutorial 1: Simulation Ensemble Creation and Execution

Creating a simulation ensemble using the variables' distributions

In this tutorial, a simulation ensemble is created (using FOQUS) and run.

The FOQUS file for this tutorial is **Rosenbrock_no_vectors.foqus**, and this file is located in: `examples/tutorial_files/UQ/Tutorial_1`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

1. From the FOQUS main screen, click the **Session** button and then select **Open Session** to open a session. Browse to the folder shown above, and select the “Rosenbrock_no_vectors.foqus” file (Figure *Home Screen*).

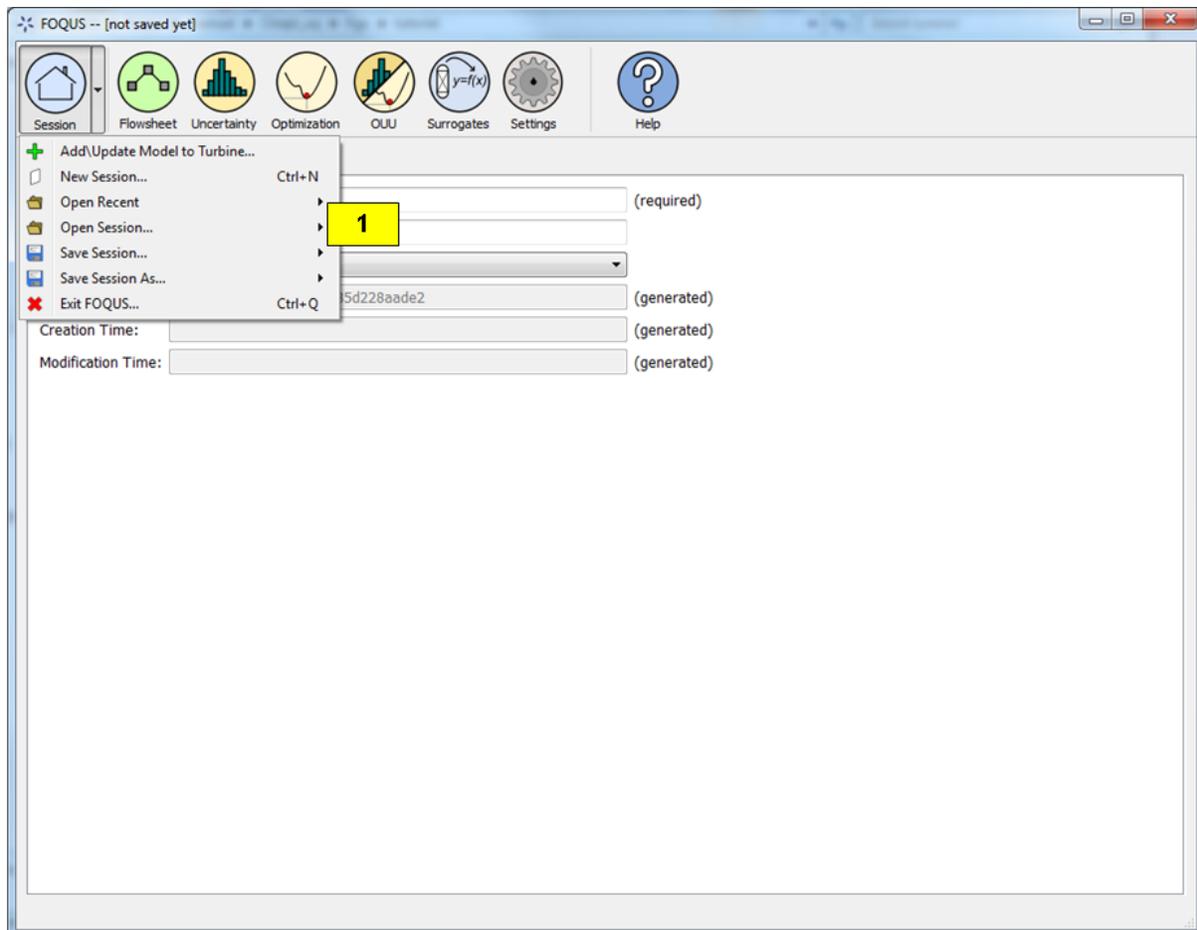


Fig. 10: Home Screen

- Opening this file loads a session that has a flowsheet with one node (Figure *Flowsheet for Rosenbrock Example*). See Section *Tutorial 1: Creating a Flowsheet* for a detailed example of creating a flowsheet.

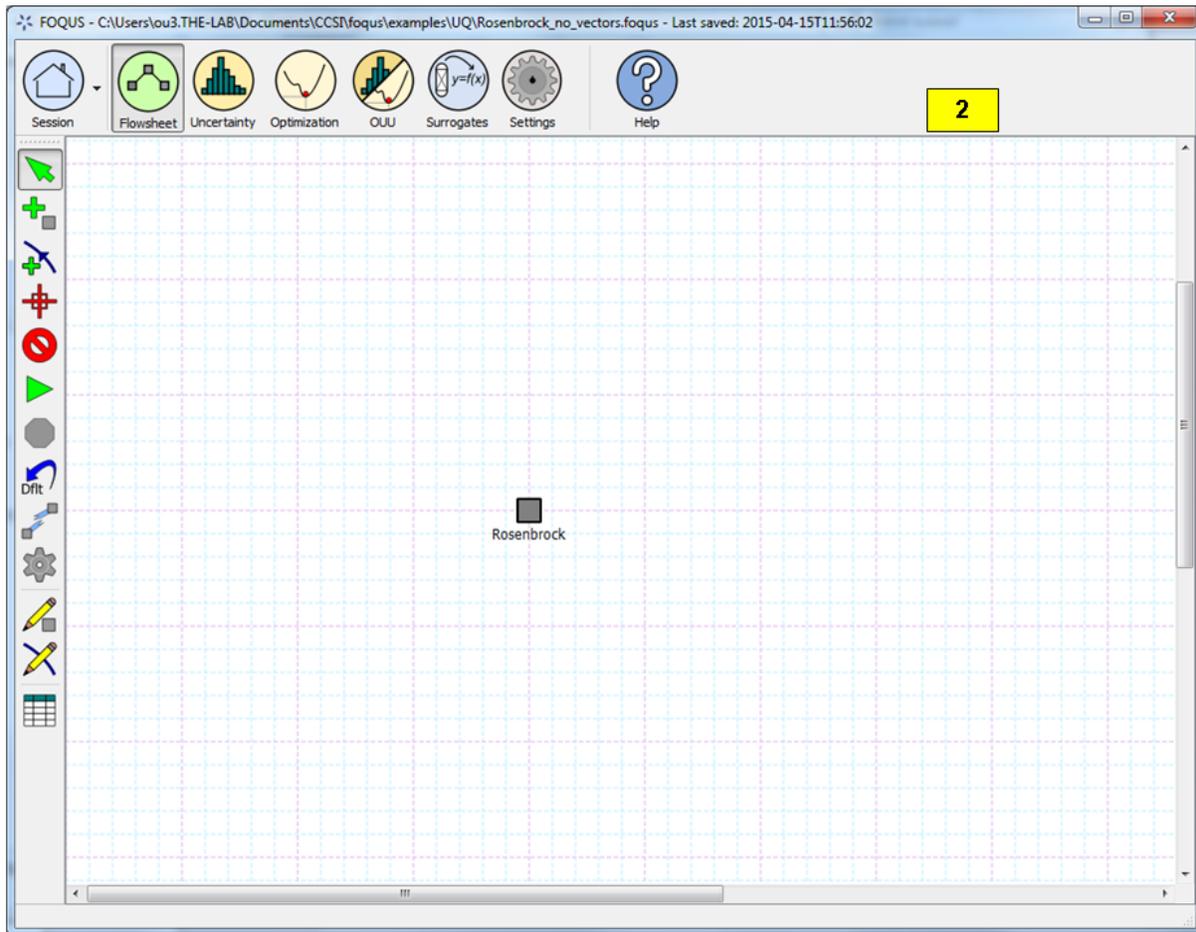


Fig. 11: Flowsheet for Rosenbrock Example

- Click the **Uncertainty** button (Figure *Uncertainty Quantification Screen*).
- Click Add New to create a new simulation ensemble.
- The Add New Ensemble dialog displays (Figure *Add New Ensemble Dialog, Flowsheet Option*). The “Use flowsheet” option should be enabled.
- This item describes additional features and is provided for information only. It is not intended to be followed as part of the step-by-step tutorial.*

An alternative is to use an emulator by selecting “Use emulator.” This alternative is preferred if the actual simulation model is too computationally expensive to be practical for a large number of samples. This option enables the user to trade off accuracy for speed by training a response surface to approximate the actual simulation model. If this option is selected (Figure *Add New Ensemble Dialog, Emulator Option*), the user needs to provide a training data file containing a small simulation ensemble generated from the actual simulation model. This training data file should be in the PSUADE full file format (see section *File Formats*).

- Click Browse and select the training data file with which to train the response surface. The inputs, outputs and response surface type is read from the training data and populated accordingly on this dialog box.
- Select Output(s) of Interest. To select multiple outputs, the user can use Shift + Click to select a range, or use Ctrl + Click to select/deselect individual outputs.

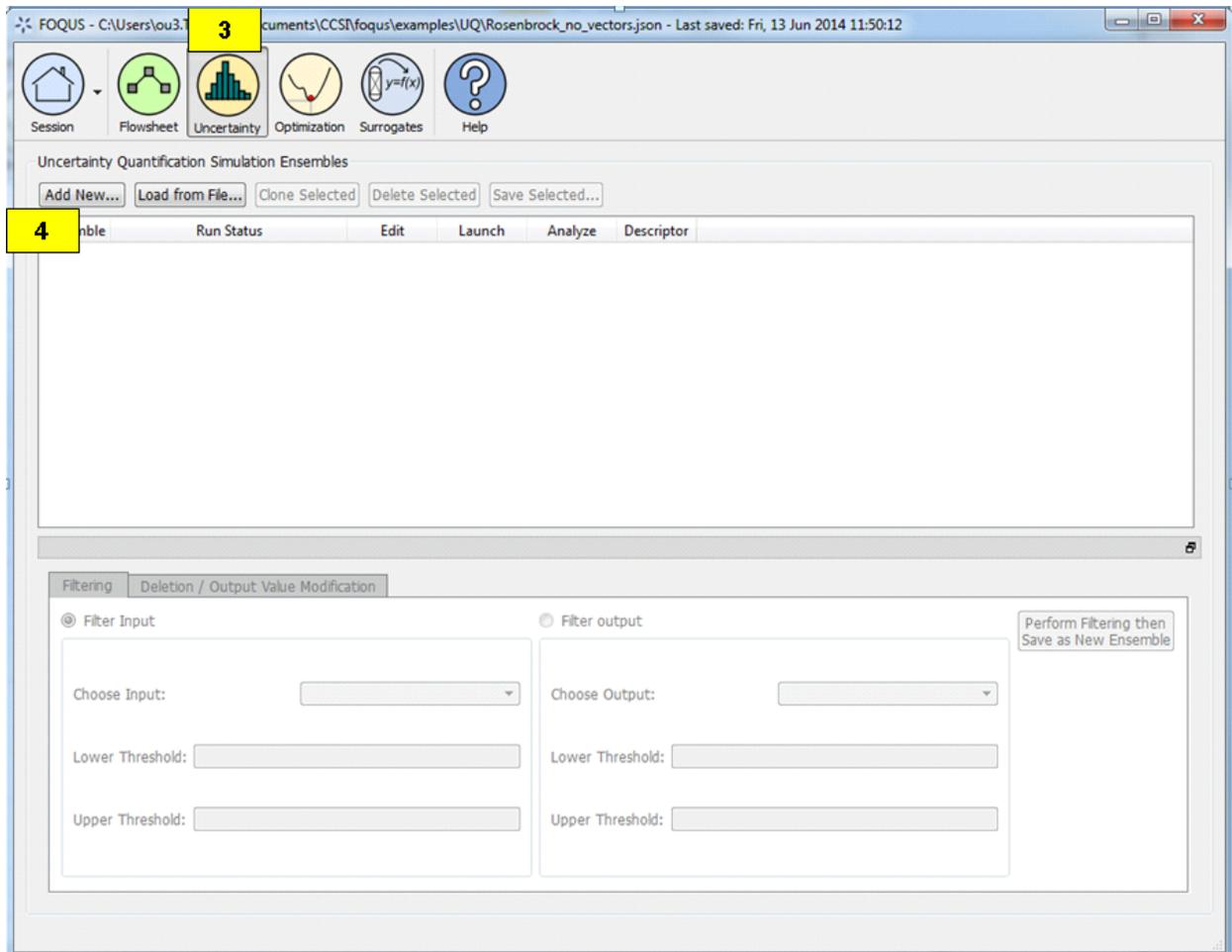


Fig. 12: Uncertainty Quantification Screen

7. Click OK.

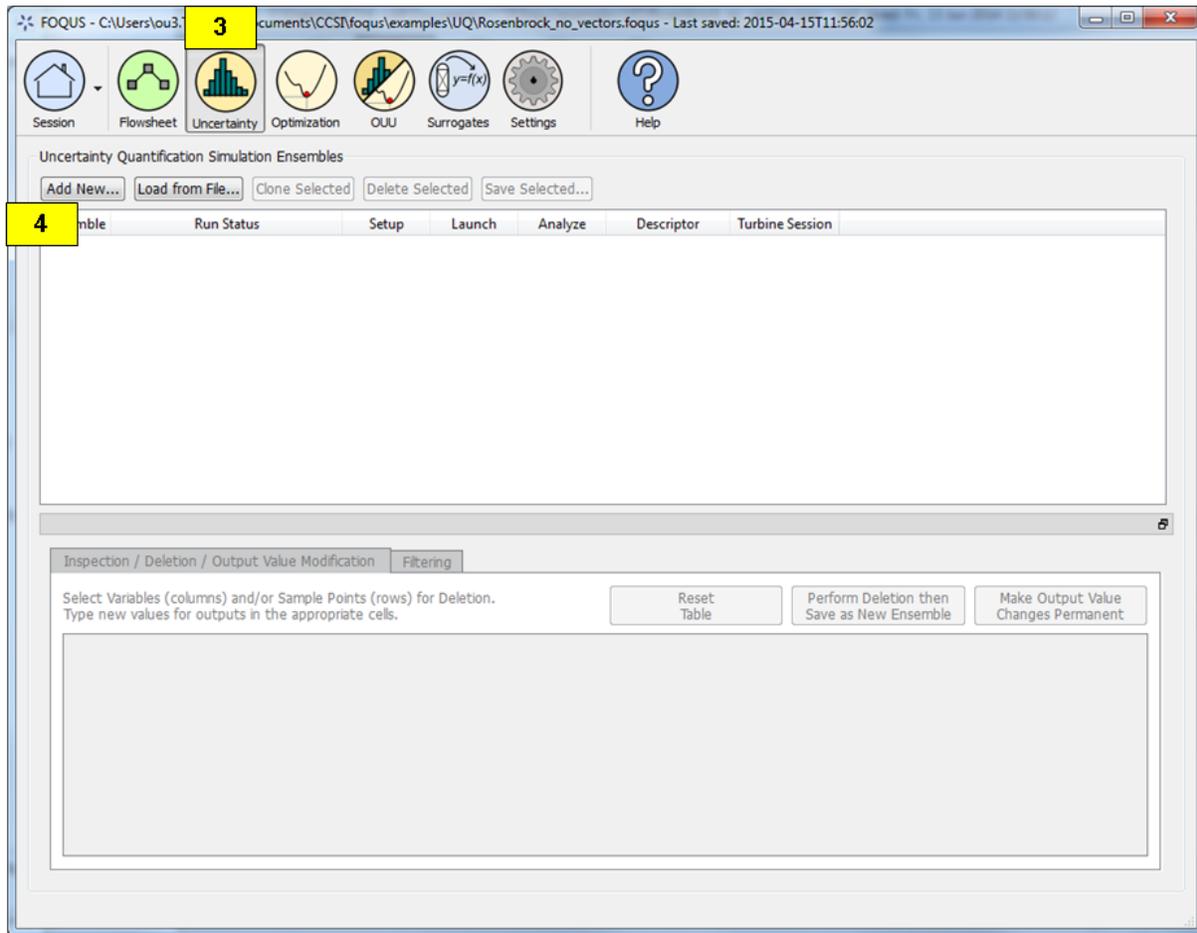


Fig. 13: Add New Ensemble Dialog, Flowsheet Option

enumerate

8. This displays the Simulation Ensemble Setup dialog box (Figure *Simulation Ensemble Setup Dialog, Distributions Tab*) that prompts the user for options specific to the creation of input samples.
9. Within the Distributions tab, the Distributions Table has all the inputs from the flowsheet node, each displayed in its own row.
 1. Click the All Variable button.
 2. Change the Type of “x2” to “fixed.”
 3. Enter 5 into the Default column for “x2.”

Subsequently, other cells in the row are enabled or disabled according to the type selection.

enumerate

In this dialog, extra options that are available related to simulation ensemble setup are discussed.

- Change the PDF of “x6” by exploring the drop-down list in the **PDF** column of the **Distributions Table**. The drop-down list is denoted by box (9c) in Figure *Simulation Ensemble Setup Dialog, Distributions Tab, PDF Selection*. If any of the parametric distributions are selected (e.g., “Normal”, “Lognormal”,

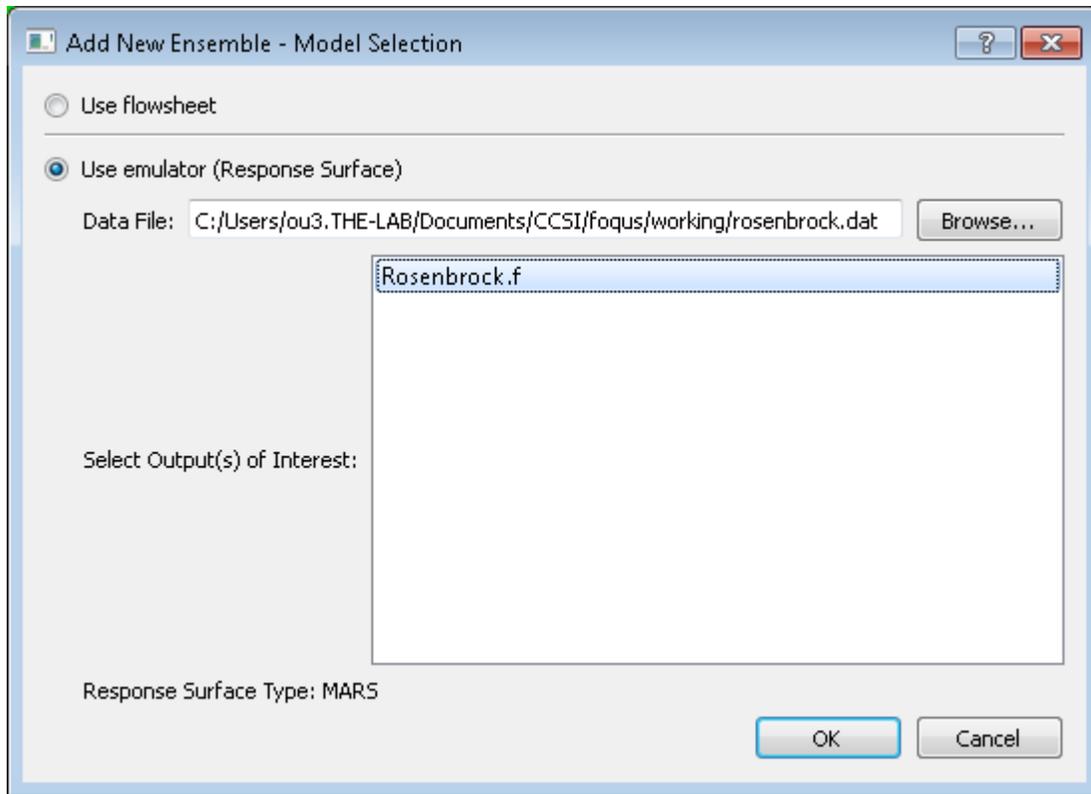


Fig. 14: Add New Ensemble Dialog, Emulator Option

“Weibull”), the user is prompted to enter the appropriate parameters for the selected distribution. If non-parametric distribution “Sample” is selected, the user needs to specify the name of the sample file (a CSV or PSUADE sample format is located in Section *File Formats*) that contains samples for the variable “x6.” The user also needs to specify the output index to indicate which output in the sample file to use. The resulting simulation ensemble would contain “x6” samples that are randomly drawn (with replacement) from the samples in this file.

- Alternatively, select Choose sampling scheme (box (8) of Figure *Simulation Ensemble Setup Dialog, Distributions Tab*), and try selecting “Load all samples from a single file.” With this selection, a new dialog box prompts the user to browse to a PSUADE full file, a PSUADE sample file, or CSV file (all formats are described in Section *File Formats*) that contains all the samples for all the input variables in the model.

Both of these options offer the user additional flexibility with respect to characterizing input uncertainty or generating the input samples directly.

enumerate

10. Once complete, switch to the Sampling Scheme tab (Figure *Simulation Ensemble Setup Dialog, Sampling Scheme Tab*).
11. Select a sampling scheme with the assumption that the user is unsure which sampling scheme to use, but wants to perform some kind of response surface analysis. This example helps the user find a suitable one.
 1. Click For response surface analysis. Note the list on the right changes accordingly.
 2. Select “Latin Hypercube” from the list on the right.
12. To generate 500 samples, change the value in “# of samples.” Some sampling schemes may impose a constraint on the number of samples. If the user has entered an incompatible sample size, a pop-up window displays with

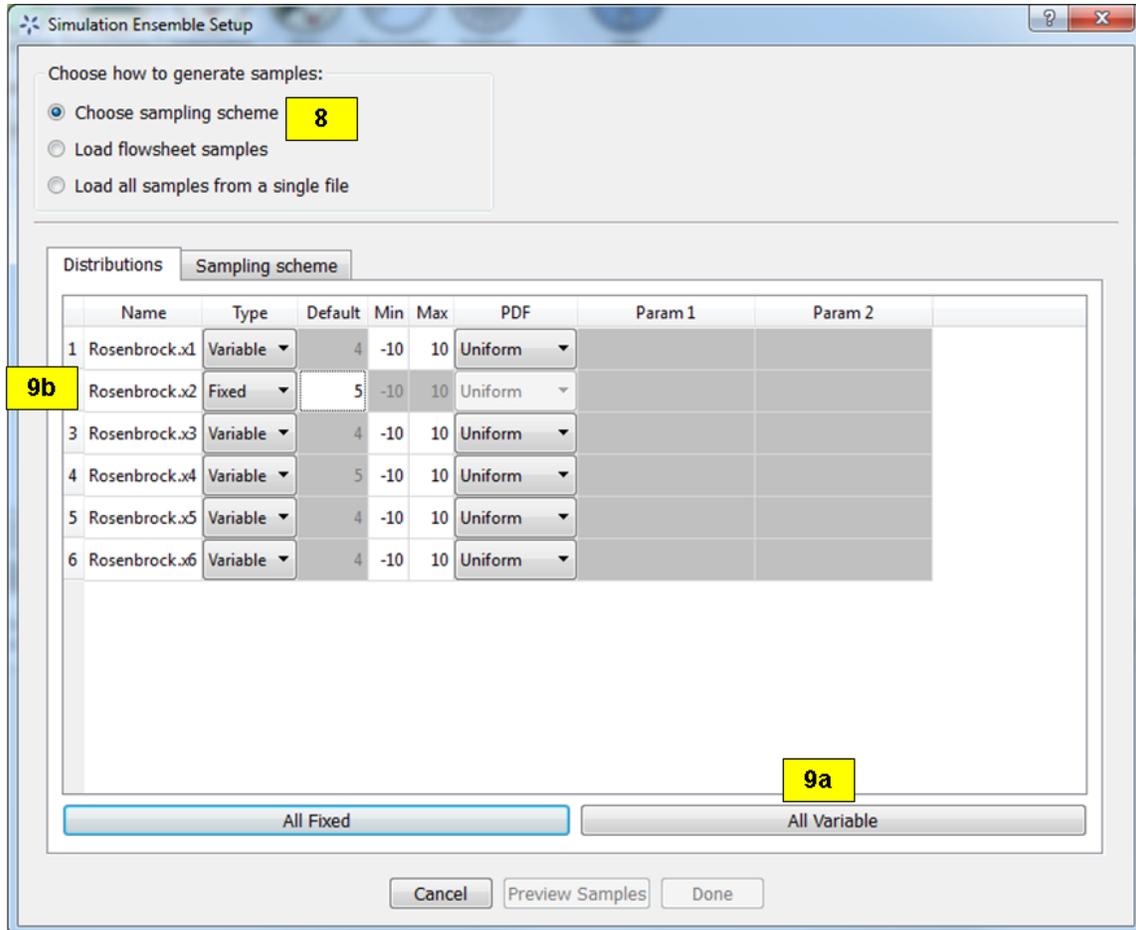


Fig. 15: Simulation Ensemble Setup Dialog, Distributions Tab

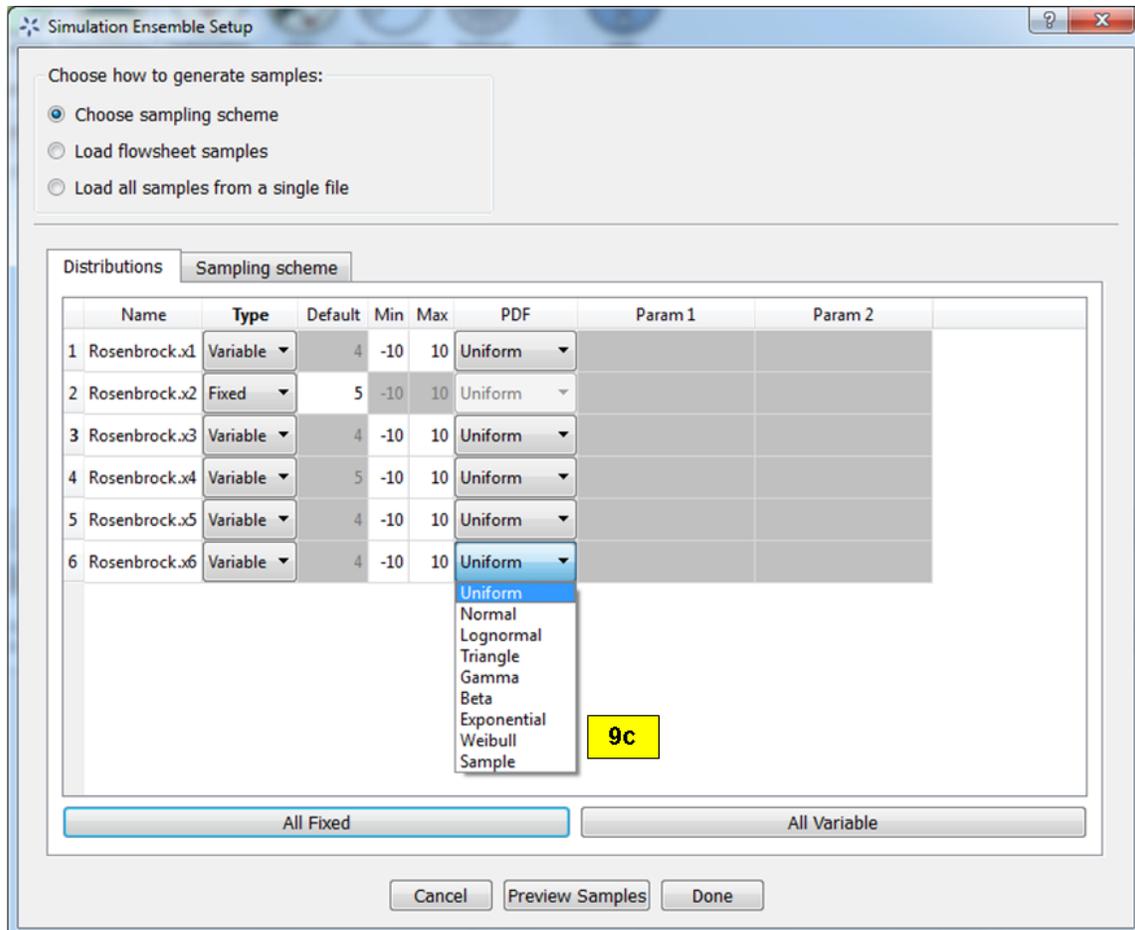


Fig. 16: Simulation Ensemble Setup Dialog, Distributions Tab, PDF Selection

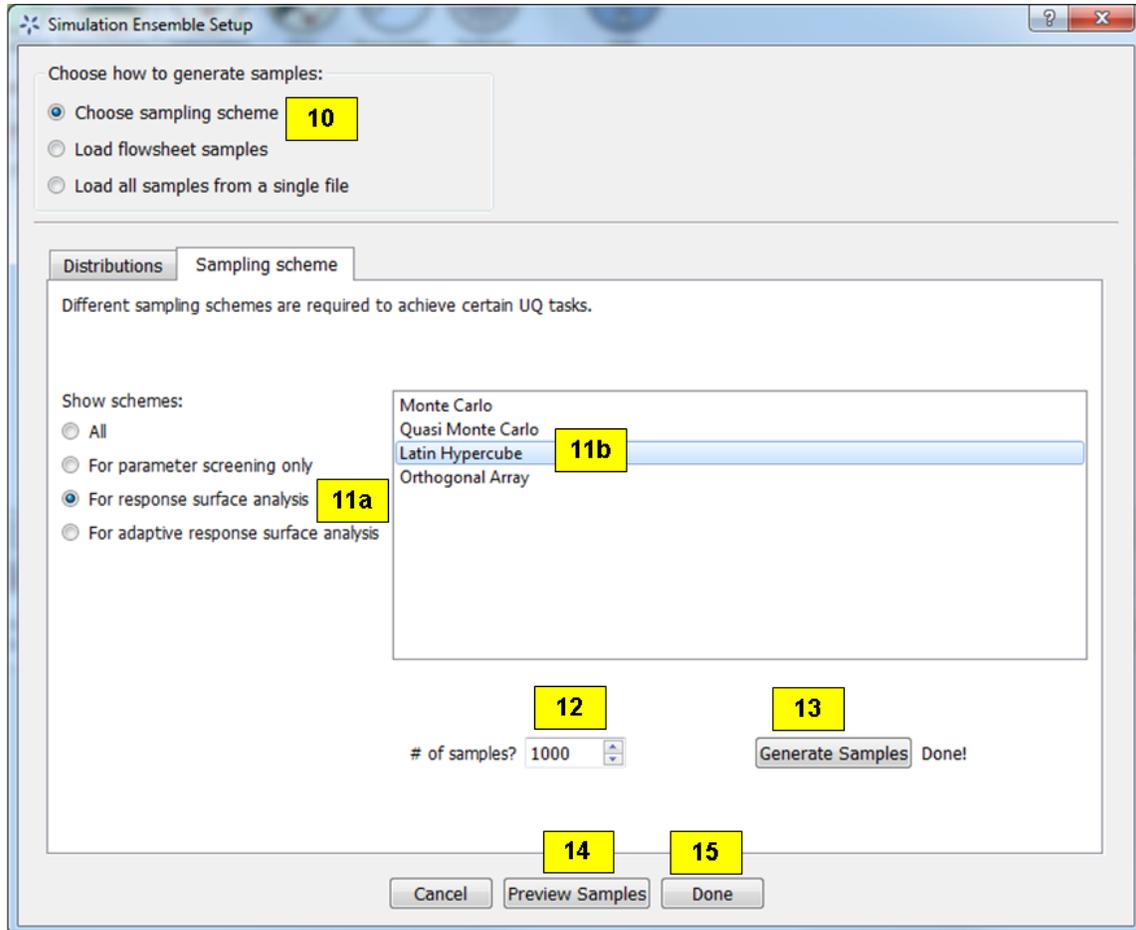


Fig. 17: Simulation Ensemble Setup Dialog, Sampling Scheme Tab

guidance on the recommended samples size.

13. Click Generate Samples to generate the sample values for all the variable input parameters. On Windows, if the user did not install PSUADE in its default location (C:\Program Files (x86)\psuade_project 1.7.1\bin\psuade.exe) and the user did not update the PSUADE path in FOQUS settings (refer to Section *Settings*), then the user is prompted to locate the PSUADE executable in a file dialog.
14. Once the samples are generated, the user can examine them by clicking Preview Samples. This displays a table of the values, as well as the option to view scatter plots of the input values. The user can also select multiple inputs at once to view them as separate scatter plots on the same figure.
15. When finished, click Done.
16. The simulation ensemble should be displayed in the Simulation Ensemble Table. If the user would like to change any of the parameters and regenerate a new set of samples, simply click the Revise button.
17. Next, calculate the output value for each sample. Click Launch. The user should see the progress bar quickly advance, displaying the status of completed runs (Figure *Simulation Ensemble Added*).

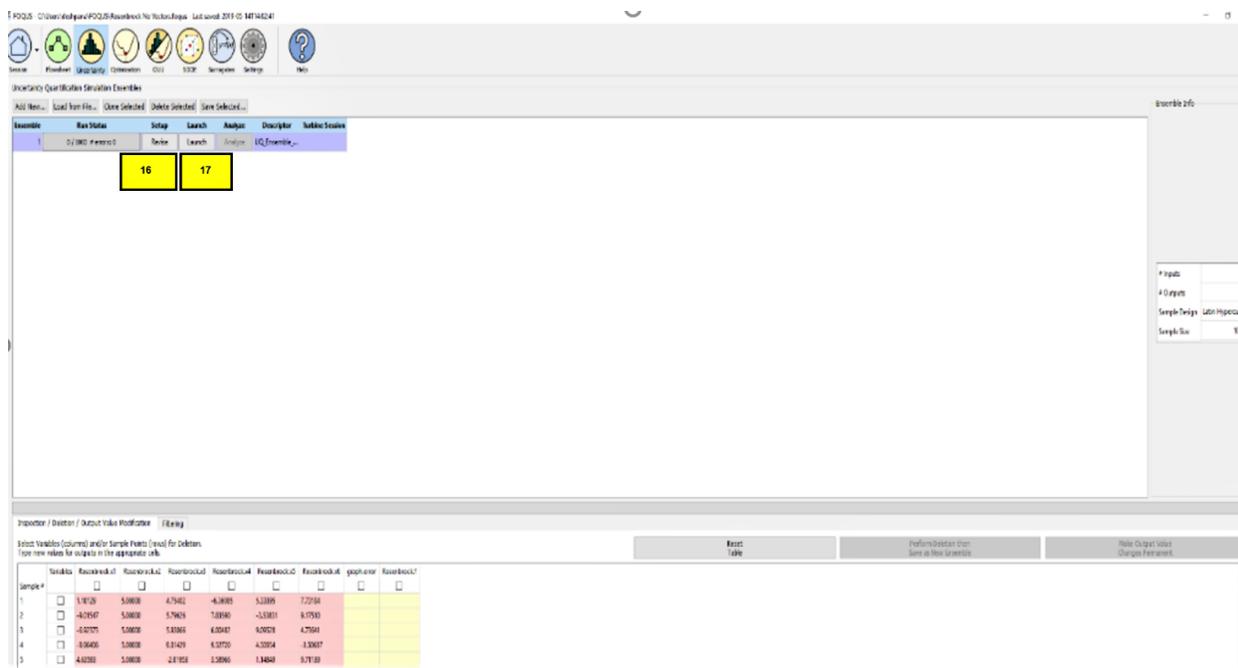


Fig. 18: Simulation Ensemble Added

18. Next, look at the output.

1. Click Analyze for “Ensemble 1” (Figure *Simulation Ensemble Evaluation Complete*).
2. Step 1 of “Analysis” (bottom page), the user selects Ensemble Data (Figure *Simulation Ensemble Analysis*).
3. Step 2 of “Analysis” is to select “Rosenbrock.f” (Figure *Simulation Ensemble Analysis*).
4. Step 3 of “Analysis” is to keep the analysis method as “Uncertainty Analysis” and then click Analyze. The user should see two graphs displaying the probability and cumulative distributions plots (Figure *Uncertainty Analysis Results*). Users should keep in mind these figures are intended to show what type of plots they would get, but they should not expect to reproduce the exact same plots.

Prior to this, the “Rosenbrock” example was selected to illustrate the process of creating and running a simulation ensemble because simulations complete quickly using this simple model. But from this point on, the adsorber subsystem

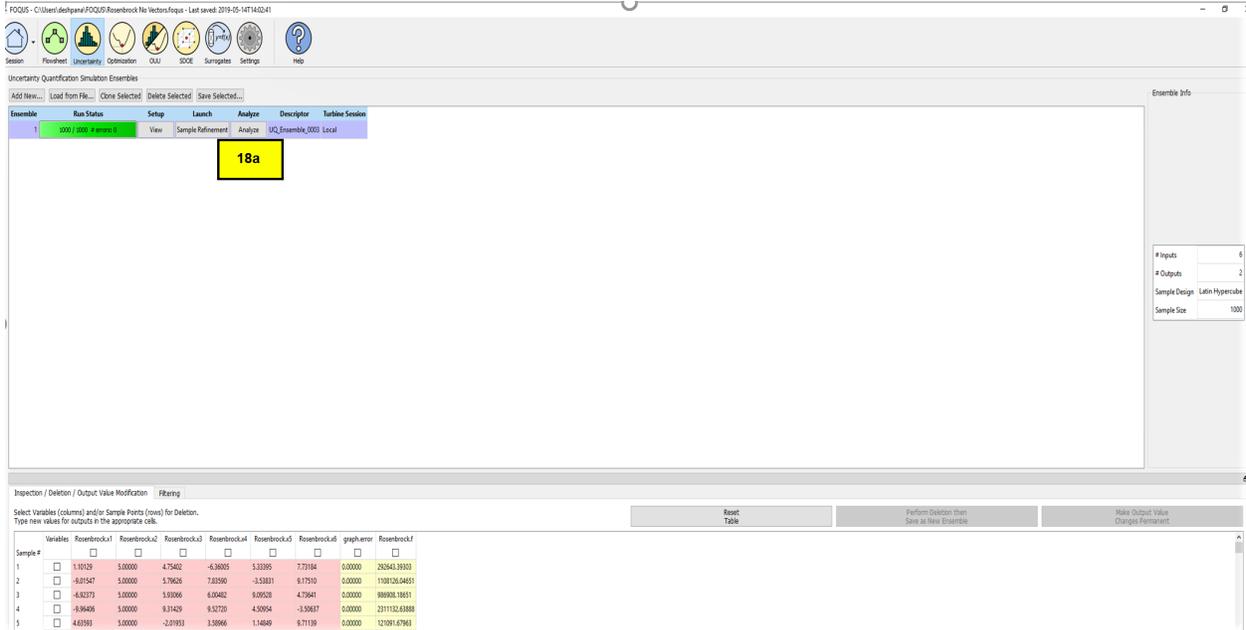


Fig. 19: Simulation Ensemble Evaluation Complete

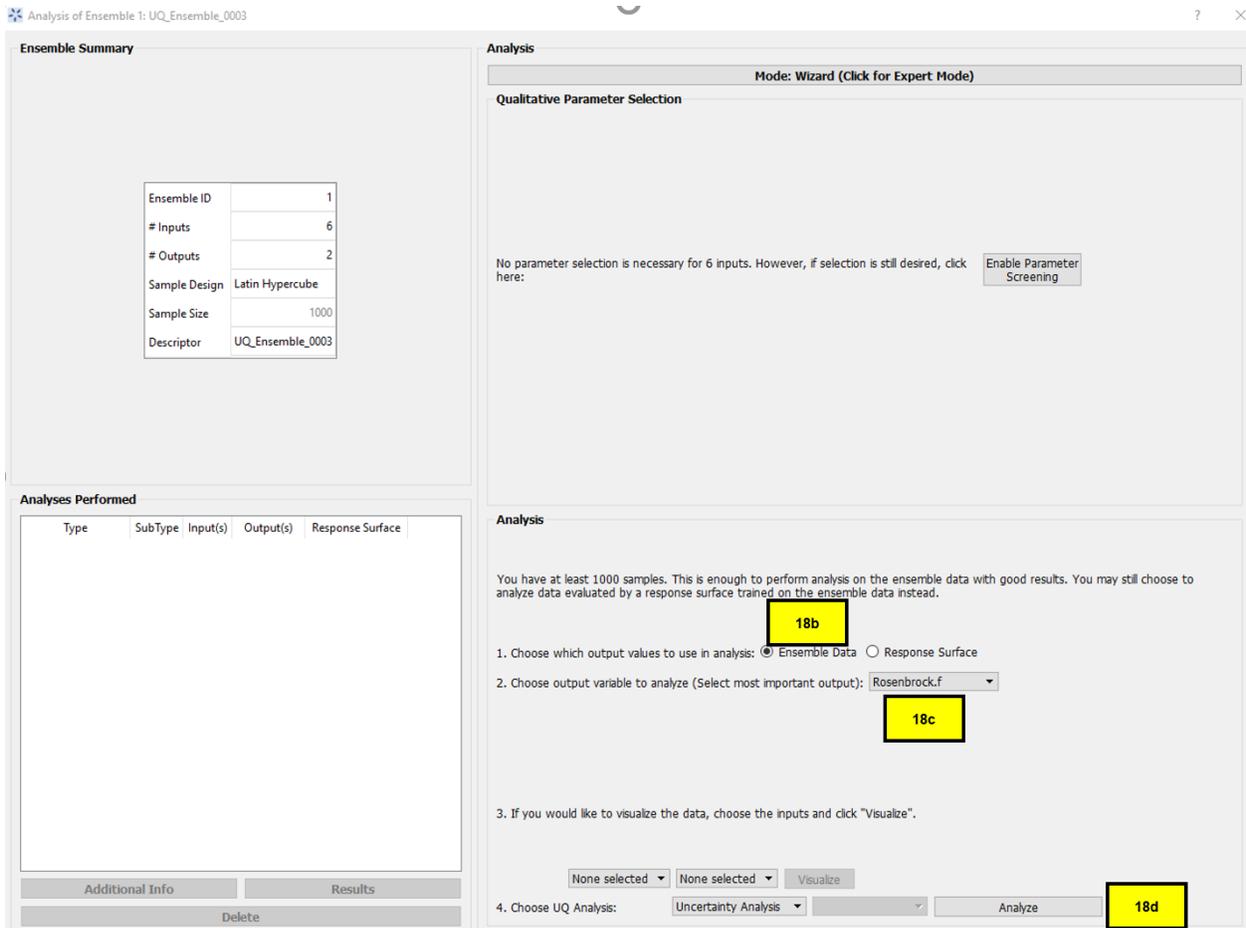


Fig. 20: Simulation Ensemble Analysis

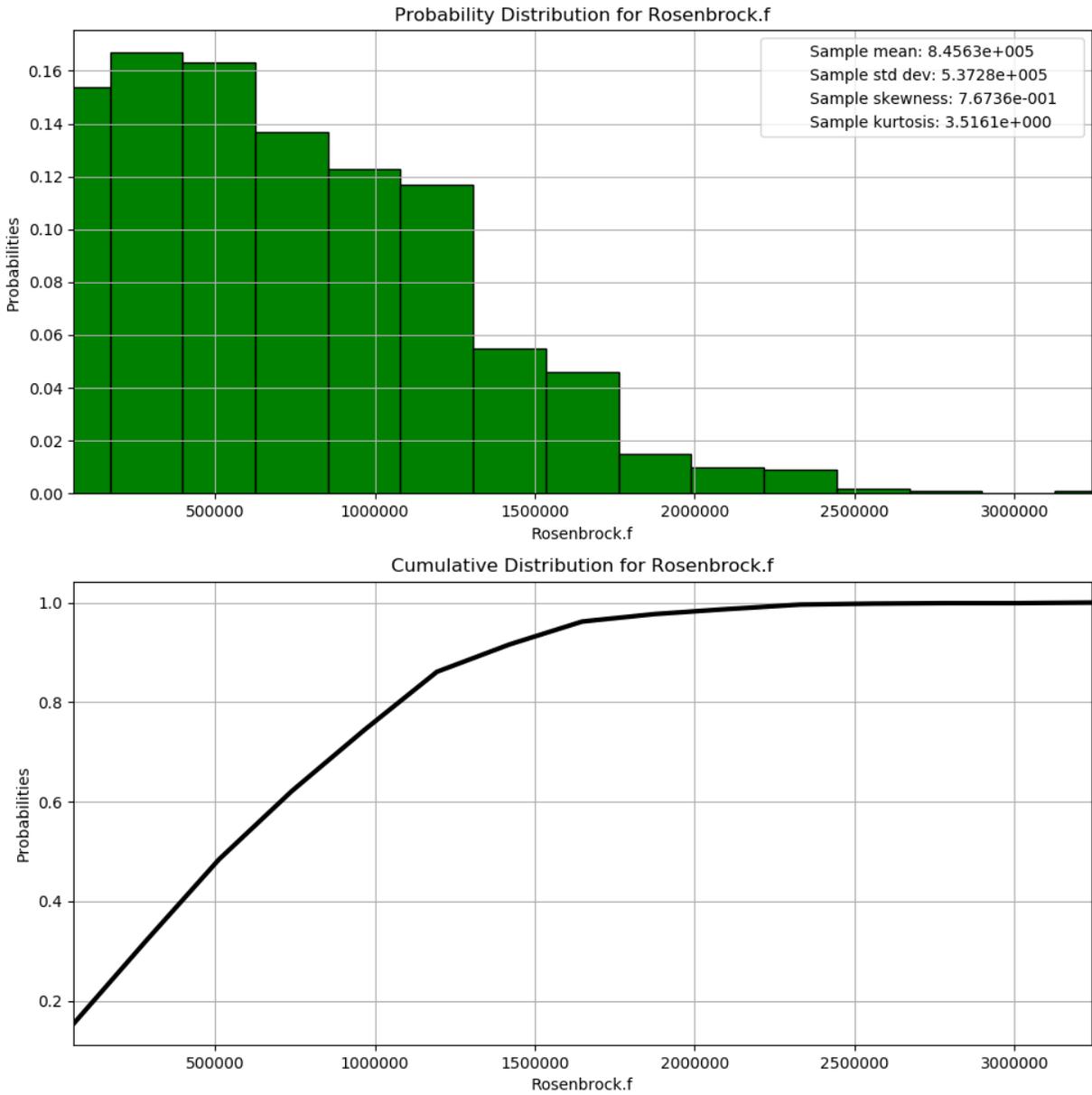


Fig. 21: Uncertainty Analysis Results

of the A650.1 design is used as a motivating example to better illustrate how one would apply UQ within the context of CCSI.

A quick recap on our motivating example: The A650.1 design consists of two coupled reactors: (1) the two-stage bubbling fluidized bed adsorber and (2) moving bed regenerator, in which the output (outlet of sorbent stream) from one reactor is the input (inlet) for the other. The performance of the entire carbon capture system is obtained by solving these two reactors simultaneously, accounting for the interactions between the reactors. However, it is also necessary to study the individual effects of the adsorber and the regenerator without the side effects of their coupling since the two reactors display distinct characteristics under different operating conditions. Thus, the Process Design/Synthesis Team has given us a version of the A650.1 model that can be run in two modes: (1) coupled and (2) decoupled. In this section, analysis results are presented from running the A650.1 model using the decoupled mode and examining the adsorber in isolation from the regenerator.

Automatically running FOQUS for a set of user-defined input conditions

In this tutorial, we will show you how to automatically run a set of user-defined input conditions in FOQUS.

This procedure will require the user to specify the input conditions in a CSV (comma-separated values) Excel file.

We will use a simple example to show the procedure.

1. Open FOQUS.
2. Go to the “Session” tab, and under “Session Name” type: `basic_example` (please see Figure *Specifying the Session Name*).

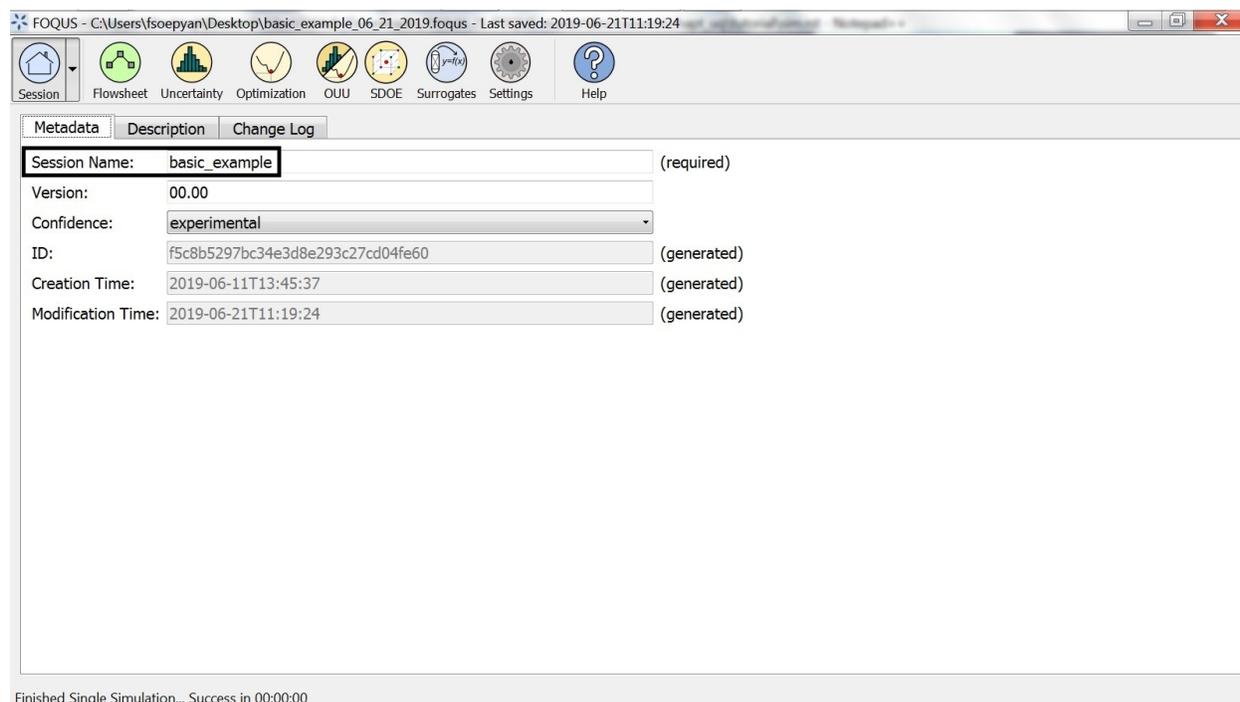


Fig. 22: Specifying the Session Name

3. Go to the “Flowsheet” tab, and click the “Add Node” button (“A” in Figure *Inserting a Node and Specifying the Inputs*).
4. Insert a node called “example” (without the quotes) (“B” in Figure *Inserting a Node and Specifying the Inputs*).

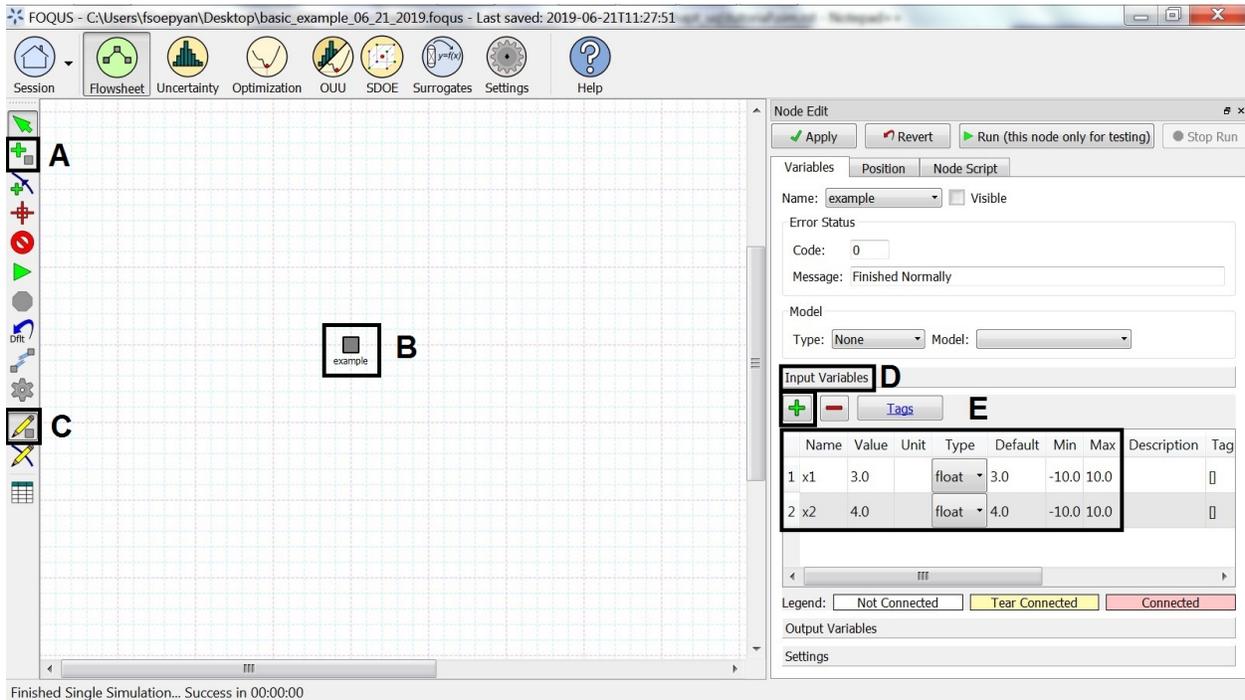


Fig. 23: Inserting a Node and Specifying the Inputs

5. Open the Node Editor by clicking the Toggle Node Editor button (“C” in Figure *Inserting a Node and Specifying the Inputs*).
6. Under the Node Editor, click “Input Variables” and the green “+” button (“D” in Figure *Inserting a Node and Specifying the Inputs*).
7. Insert input variables x1 and x2 (“E” in Figure *Inserting a Node and Specifying the Inputs*).
8. For x1, specify the value, default, minimum, and maximum as 3, 3, -10, and 10, respectively (“E” in Figure *Inserting a Node and Specifying the Inputs*).
9. For x2, specify the value, default, minimum, and maximum as 4, 4, -10, and 10, respectively (“E” in Figure *Inserting a Node and Specifying the Inputs*).
10. Under the Node Editor, click “Output Variables” and the green “+” button (“A” and “B” in Figure *Specifying the Outputs*).
11. Insert output variables y1 and y2 (“C” in Figure *Specifying the Outputs*).
12. Under the Node Editor, click “Node Script” (“A” in Figure *Inserting the Equations*).
13. In the first line under “Node Script (Python Code)”, type: $f[‘y1’] = 2 * x[‘x1’] + 3 * x[‘x2’]$ (“B” in Figure *Inserting the Equations*).
14. In the second line under “Node Script (Python Code)”, type: $f[‘y2’] = 3 * x[‘x1’] + 5 * x[‘x2’]$ (“B” in Figure *Inserting the Equations*).
15. Open Microsoft Excel.
16. Type example.x1 and example.x2 as the headings in Cells A1 and B1 (please see Figure *Specifying the Inputs in Excel*).

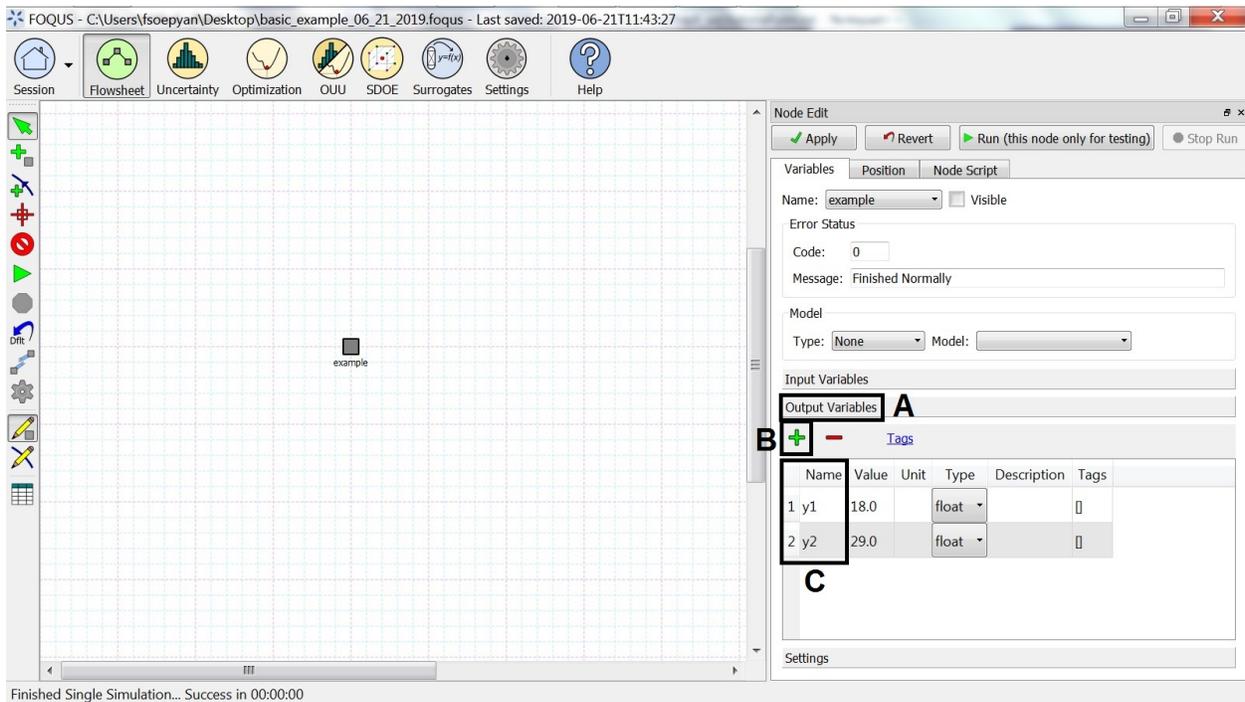


Fig. 24: Specifying the Outputs

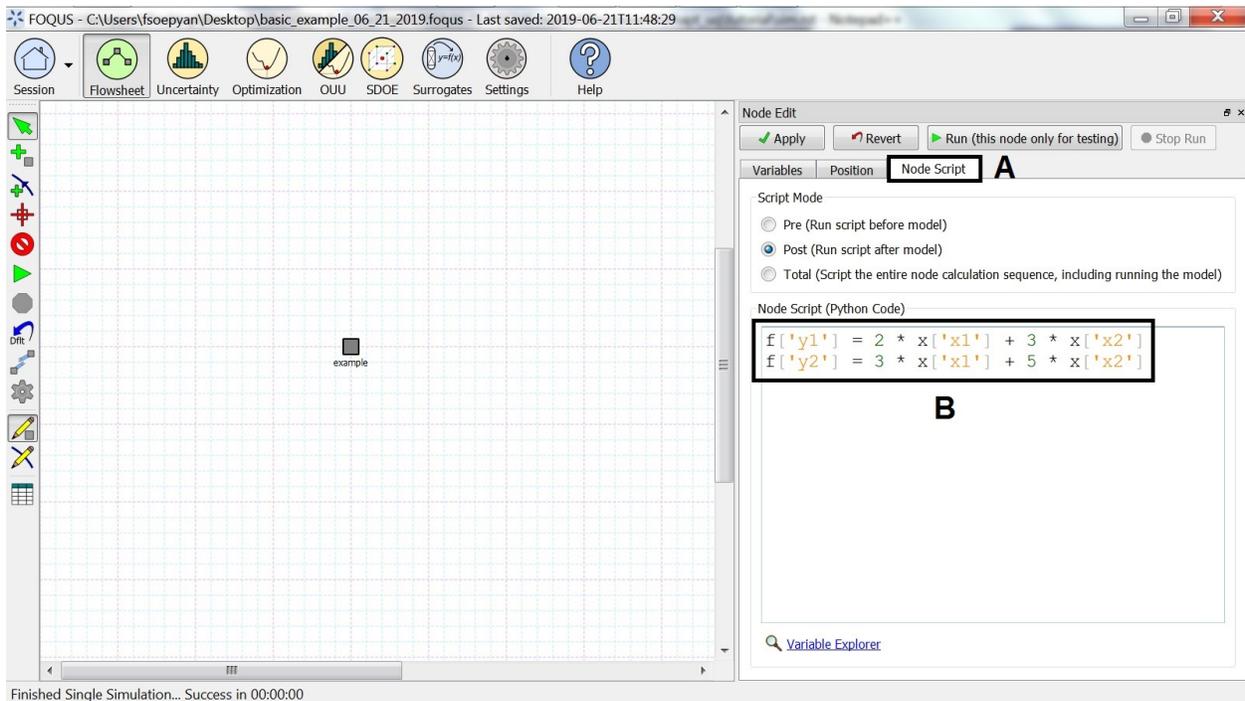


Fig. 25: Inserting the Equations

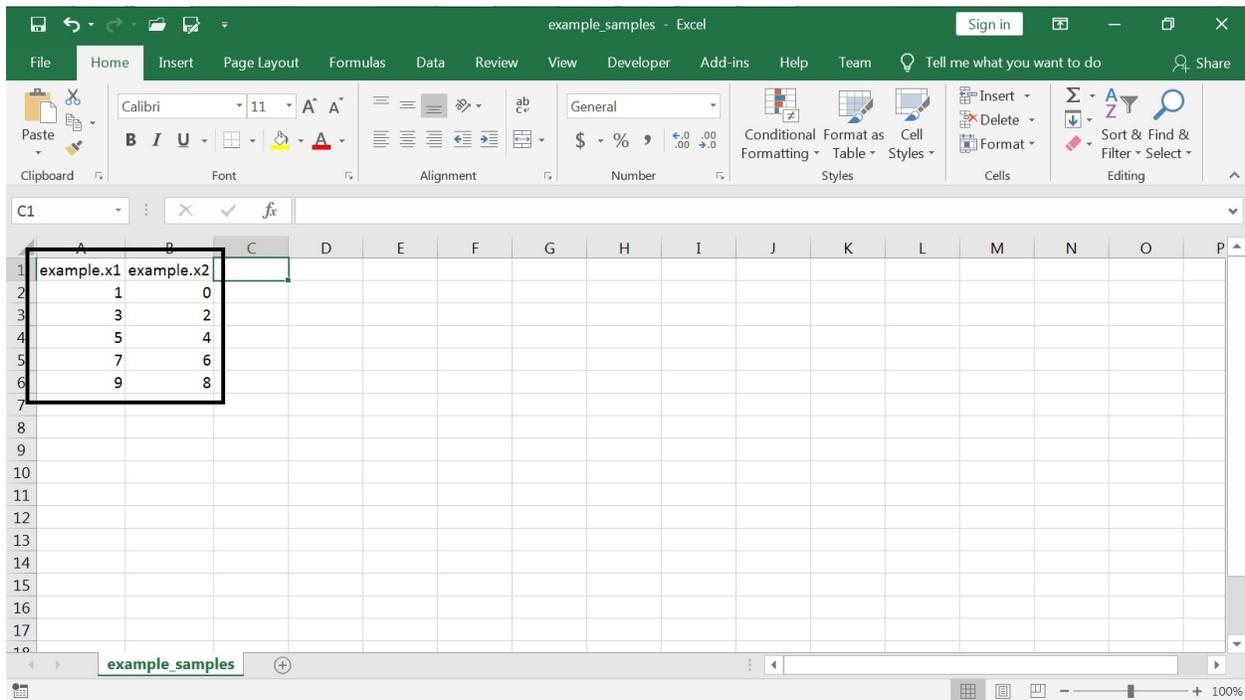


Fig. 26: Specifying the Inputs in Excel

17. Type 1, 3, 5, 7, 9 under example.x1 (please see Figure *Specifying the Inputs in Excel*).
18. Type 0, 2, 4, 6, 8 under example.x2 (please see Figure *Specifying the Inputs in Excel*).
19. Save the Excel file, with file name “example_samples” (without the quotes), and “CSV (MS-DOS)” as the file type .
20. Return to FOQUS, and go to the “Uncertainty” tab (“A” in Figure *The Uncertainty Tab in FOQUS*).
21. Click the “Add New” button (“B” in Figure *The Uncertainty Tab in FOQUS*).
22. Select “Use flowsheet”, and click “OK” (“C” and “D” in Figure *The Uncertainty Tab in FOQUS*).
23. Select “Load all samples from a single file” (“A” in Figure *Uploading the CSV File Containing the Inputs*).
24. Click “Browse”, and select the “example_samples” CSV file (“B” in Figure *Uploading the CSV File Containing the Inputs*).
25. Click “Done” (“C” in Figure *Uploading the CSV File Containing the Inputs*).
26. The user-specified inputs should appear in the “Ensemble” table (please see Figure *The User-Specified Inputs in the Uncertainty Tab*).
27. Run these inputs by clicking the “Launch” button (please see Figure *The User-Specified Inputs in the Uncertainty Tab*).
28. After the runs are finished, the results are shown in the table at the bottom of the “Uncertainty” tab (please see Figure *The Results of the Runs in the Uncertainty Tab*).

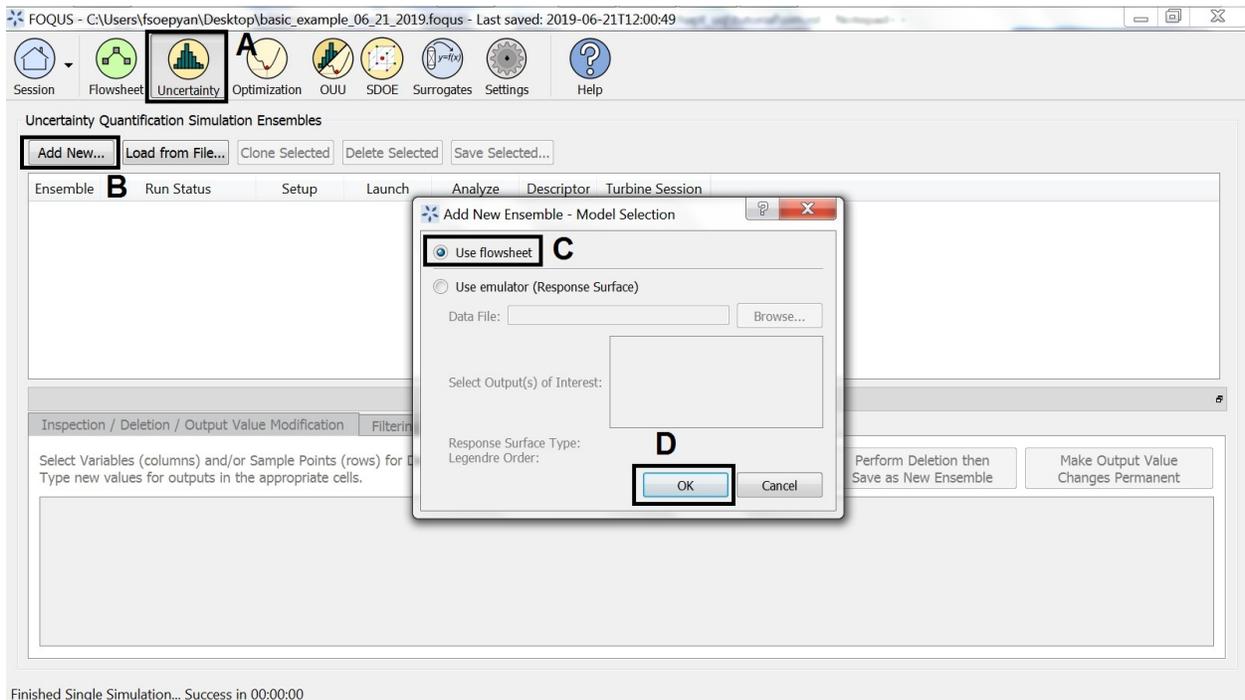


Fig. 27: The Uncertainty Tab in FOQUS

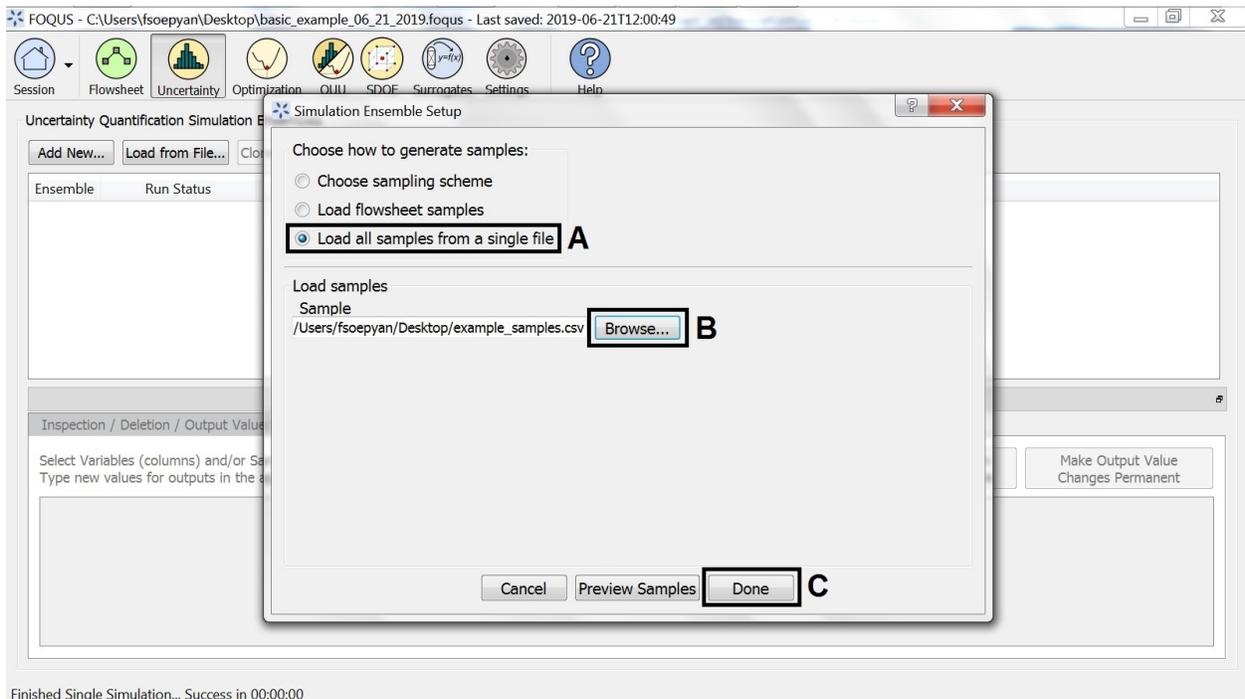


Fig. 28: Uploading the CSV File Containing the Inputs

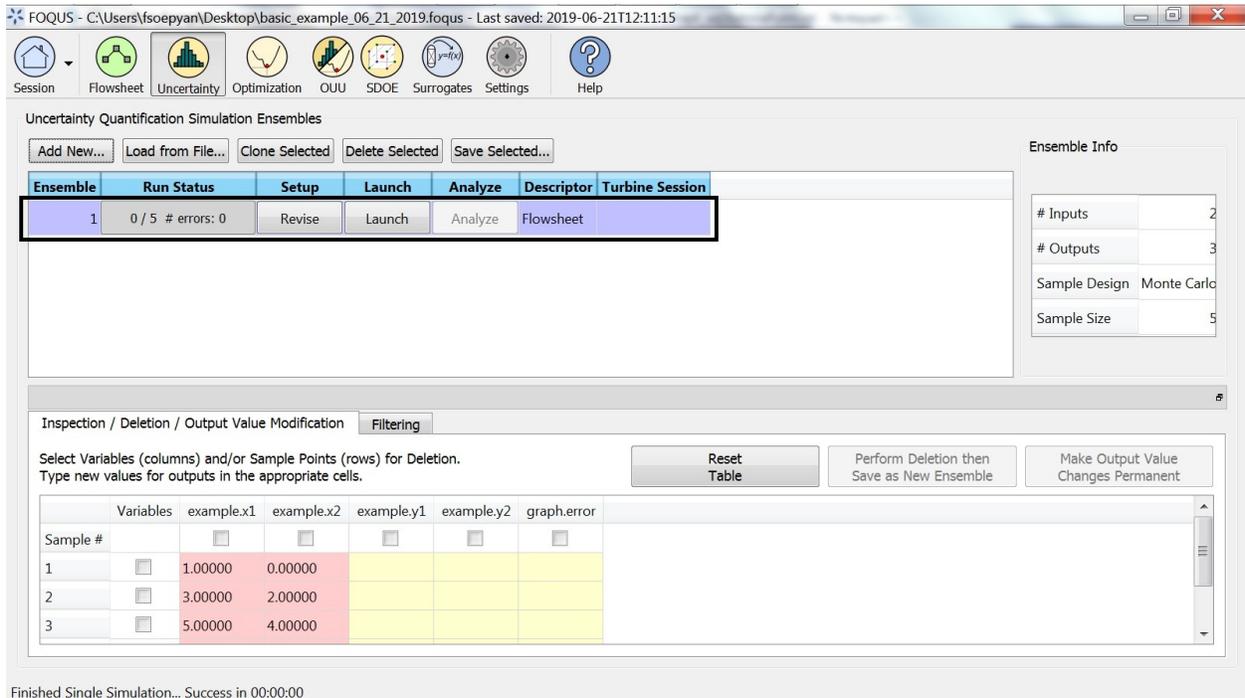


Fig. 29: The User-Specified Inputs in the Uncertainty Tab

29. The user can also view the results in the Flowsheet tab by clicking the “Results and Filtering” button (“A” in Figure *The Results of the Runs in the Flowsheet Table*).
30. The Flowsheet Table contains the results (“B” in Figure *The Results of the Runs in the Flowsheet Table*).

Tutorial 2: Data Manipulation

In this tutorial, instructions to change the data before analysis are described. Current capabilities include sample filtering, input/output variable deletion, and output value modification.

The files for this tutorial are located in: `examples/tutorial_files/UQ/Tutorial_2`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

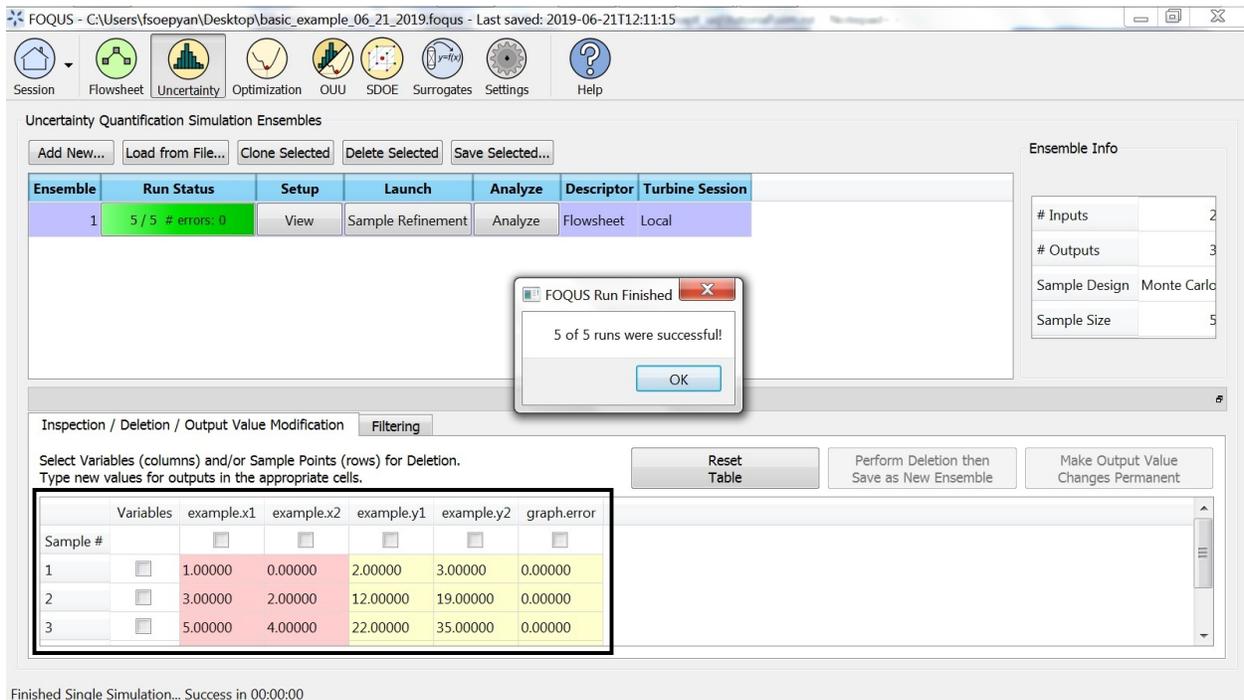


Fig. 30: The Results of the Runs in the Uncertainty Tab

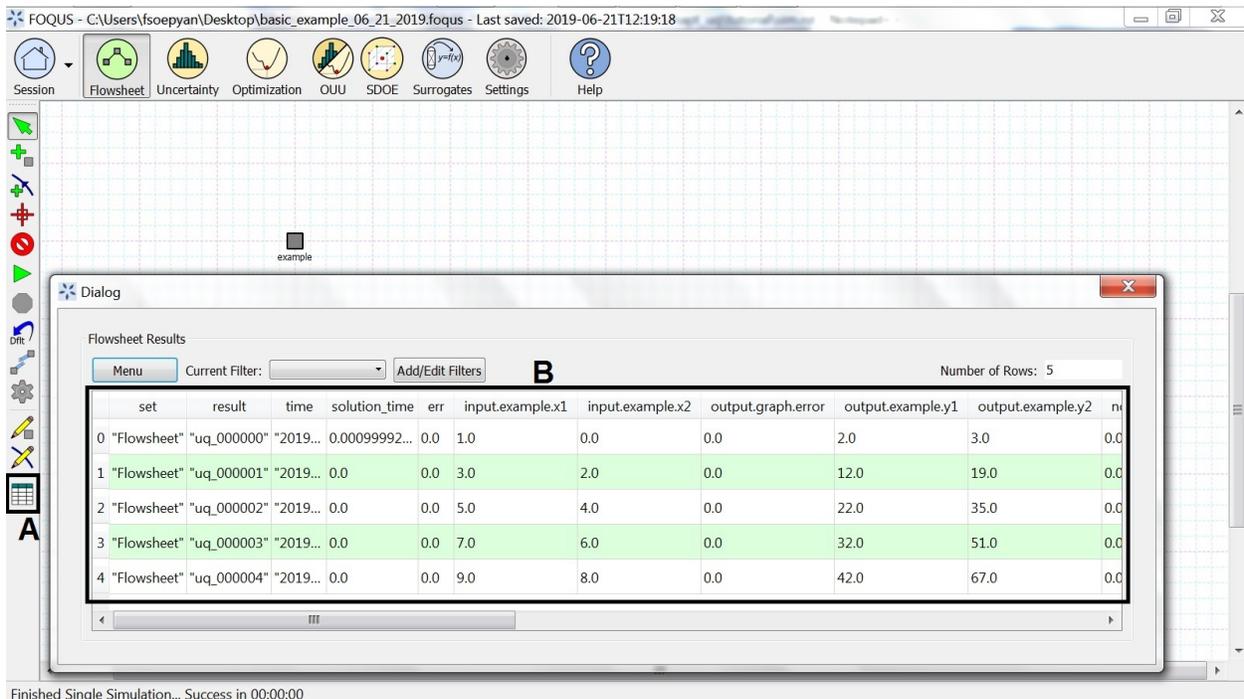


Fig. 31: The Results of the Runs in the Flowsheet Table

Filtering

Filtering involves selecting out samples whose values of a certain input or output fall into a certain range. When runs are returned from the Turbine Web API there often are simulations that failed to converge in Aspen, thus the simulation samples corresponding to these failed runs should be excluded from analysis. Follow the steps below to filter out the samples due to failed runs:

1. Click Load from File on the UQ window (Figure *Data Manipulation, Filtering Tab*).
2. Select the file “gmoat5012_9levels.res” in the examplesUQ folder. This file is an actual simulation ensemble that has already been run. To find this file, the user may need to change the file filter to “All files.”
3. Select the Filtering tab.

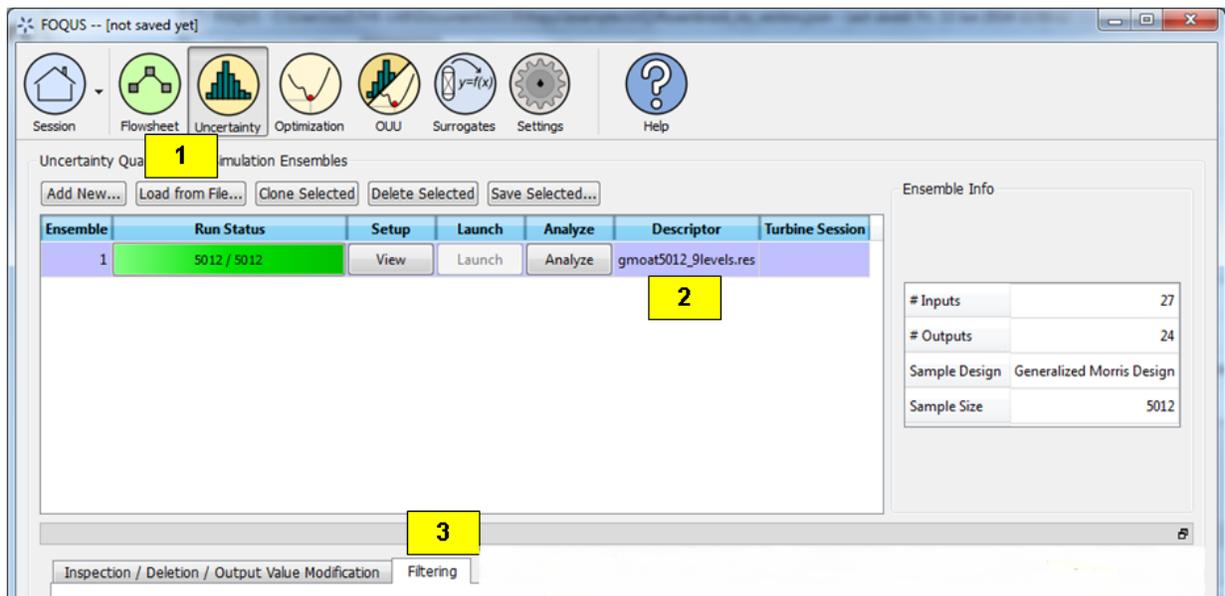


Fig. 32: Data Manipulation, Filtering Tab

4. Filtering the loaded simulation ensemble based on output values is performed.
 1. Click on “New Filter”, and create a filter named “f1”
 2. Add the Filter Expression $c(\text{“output.status”}) == 0$, since the user should keep only the samples in which the output parameter status is “0.”
 3. Click “Done”
 4. Select ‘f1’ as the “Current Filter” in the Flowsheet Result window within “Filtering Tab”
 5. Once the Filtering is complete, click on “Save as New Ensemble” and a new row should be added to the simulation table
5. Once filtering is complete, a new row should be added to the simulation table (Figure *Data Manipulation, Filtering Results*). This ensemble contains only those samples that have a status value of “0.” Analysis can now be performed on this new ensemble because this ensemble contains only the valid simulations (i.e., those with output status value of 0), in which Aspen calculations have properly converged.

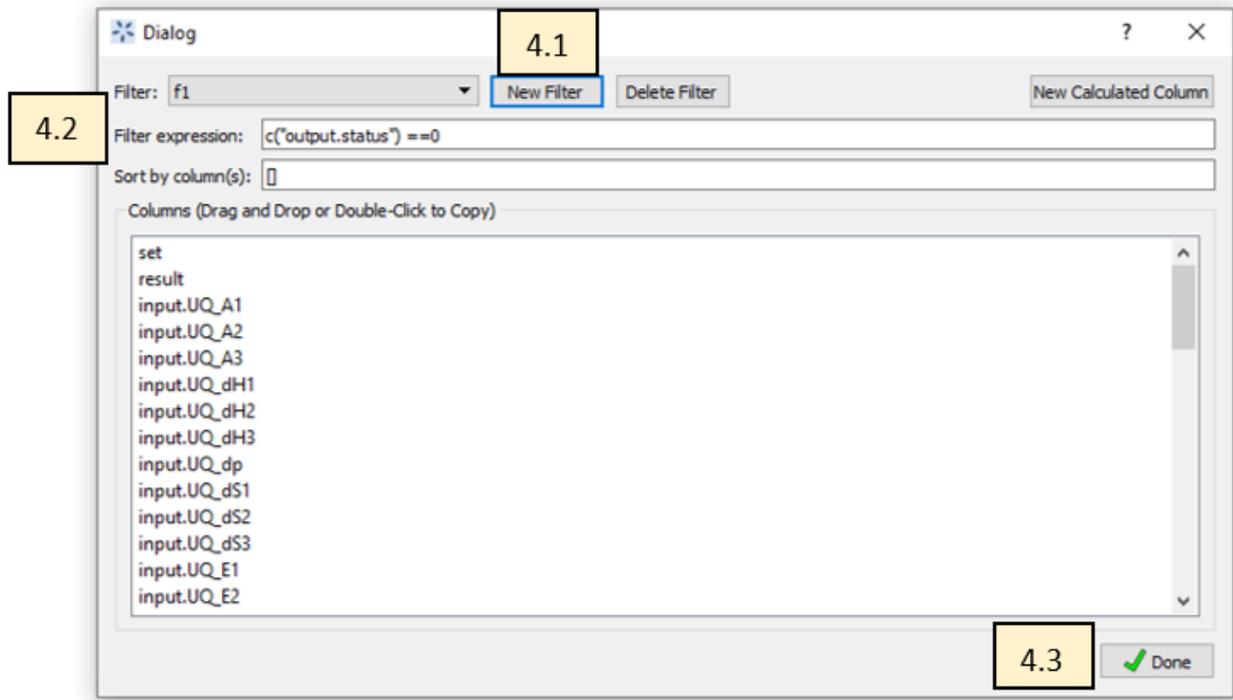


Fig. 33: Data Manipulation, Filtering Dialog Box

Variable Deletion

If an input or output variable is to be removed from consideration for analysis, this can be done in the **Inspection/Deletion/Output Value Modification** tab. Delete the status output from the previous filtering as it is no longer needed for further analysis.

1. Verify that the ensemble that resulted from filtering is selected. If not, select that ensemble.
2. Click the Inspection/Deletion/Output Modification tab.
3. Scroll to the right of the table to the outputs, which are colored yellow.
4. Select the checkbox corresponding to the “status” output (the first output).
5. Click Perform Deletion then Save as New Ensemble.

The results are illustrated in Figure *Data Manipulation, Inspection/Deletion*. Note: The output count has decreased by one for the new ensemble. The user can verify that the “status” output was removed in the new ensemble by viewing this in the **Inspection/Deletion/Output Value Modification** tab again. Deletion of an input can be performed similarly by selecting its checkbox and clicking the **Perform Deletion then Save as New Ensemble** button.

FOQUS -- [not saved yet]

Session Flowsheet **Uncertainty** Optimization OOU SDOE Surrogates Settings Help

Uncertainty Quantification Simulation Ensembles

Add New... Load from File... Clone Selected Delete Selected Save Selected...

Ensemble	Run Status	Setup	Launch	Analyze	Descriptor	Turbine Session
1	5012 / 5012	View	Sample Refinement	Analyze	gmoat5012_9levels.res	
2	4407 / 4407	View	Sample Refinement	Analyze	gmoat5012_9levels.filtered	5

Inspection / Deletion / Output Value Modification Filtering

Flowsheet Results

Menu Current Filter: f1 **4.4** Add/Edit Filters Save as New Ensemble **4.5**

	set	result	input.UQ_A1	input.UQ_A2	input.UQ_A3	input.UQ_dH1	input.UQ_dH2	input.UQ_dH3	input.UQ_dp
0	"default"	"res"	1.06666666666...	0.8	0.8	1.06666666666...	1.06666666666...	0.93333333333...	1.2
1	"default"	"res"	1.06666666666...	0.8	0.8	1.06666666666...	1.06666666666...	0.93333333333...	1.06666666666...
2	"default"	"res"	1.06666666666...	0.8	0.93333333333...	1.06666666666...	1.06666666666...	0.93333333333...	1.06666666666...
3	"default"	"res"	1.2	0.8	0.93333333333...	1.06666666666...	1.06666666666...	0.93333333333...	1.06666666666...
4	"default"	"res"	1.2	0.8	0.93333333333...	1.06666666666...	1.06666666666...	0.93333333333...	1.06666666666...
5	"default"	"res"	1.2	0.8	0.93333333333...	1.06666666666...	1.06666666666...	0.93333333333...	1.06666666666...
6	"default"	"res"	1.2	0.8	0.93333333333...	1.06666666666...	0.93333333333...	0.93333333333...	1.06666666666...
7	"default"	"res"	1.2	0.8	0.93333333333...	1.06666666666...	0.93333333333...	0.93333333333...	1.06666666666...
8	"default"	"res"	1.2	0.8	0.93333333333...	1.06666666666...	0.93333333333...	0.93333333333...	1.06666666666...
9	"default"	"res"	1.2	0.8	0.93333333333...	1.06666666666...	0.93333333333...	0.8	1.06666666666...
10	"default"	"res"	1.2	0.8	0.93333333333...	1.06666666666...	0.93333333333...	0.8	1.06666666666...
11	"default"	"res"	1.2	0.8	0.93333333333...	1.06666666666...	0.93333333333...	0.8	1.06666666666...
12	"default"	"res"	1.2	0.8	0.93333333333...	1.06666666666...	0.93333333333...	0.8	1.06666666666...

Fig. 34: Data Manipulation, Applying the filter

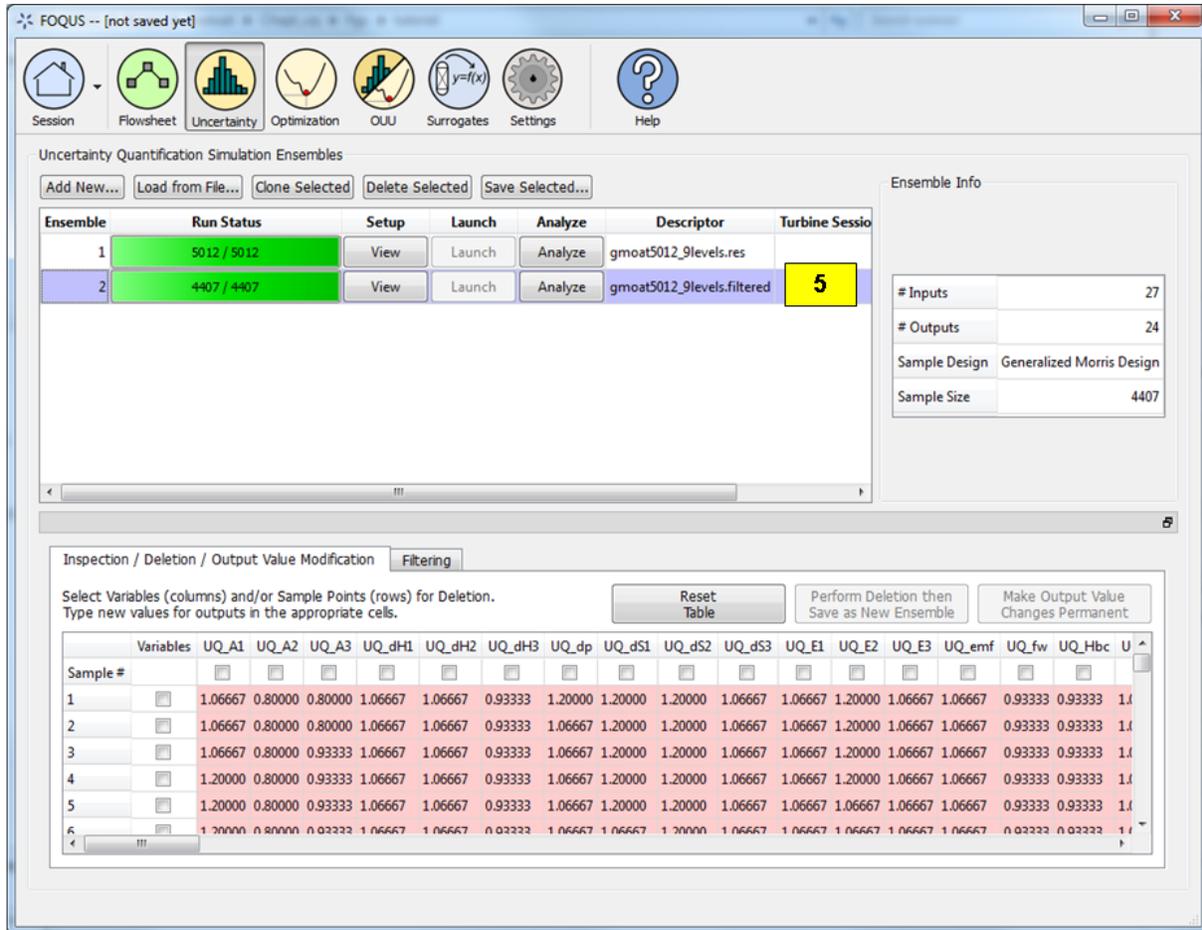


Fig. 35: Data Manipulation, Filtering Results

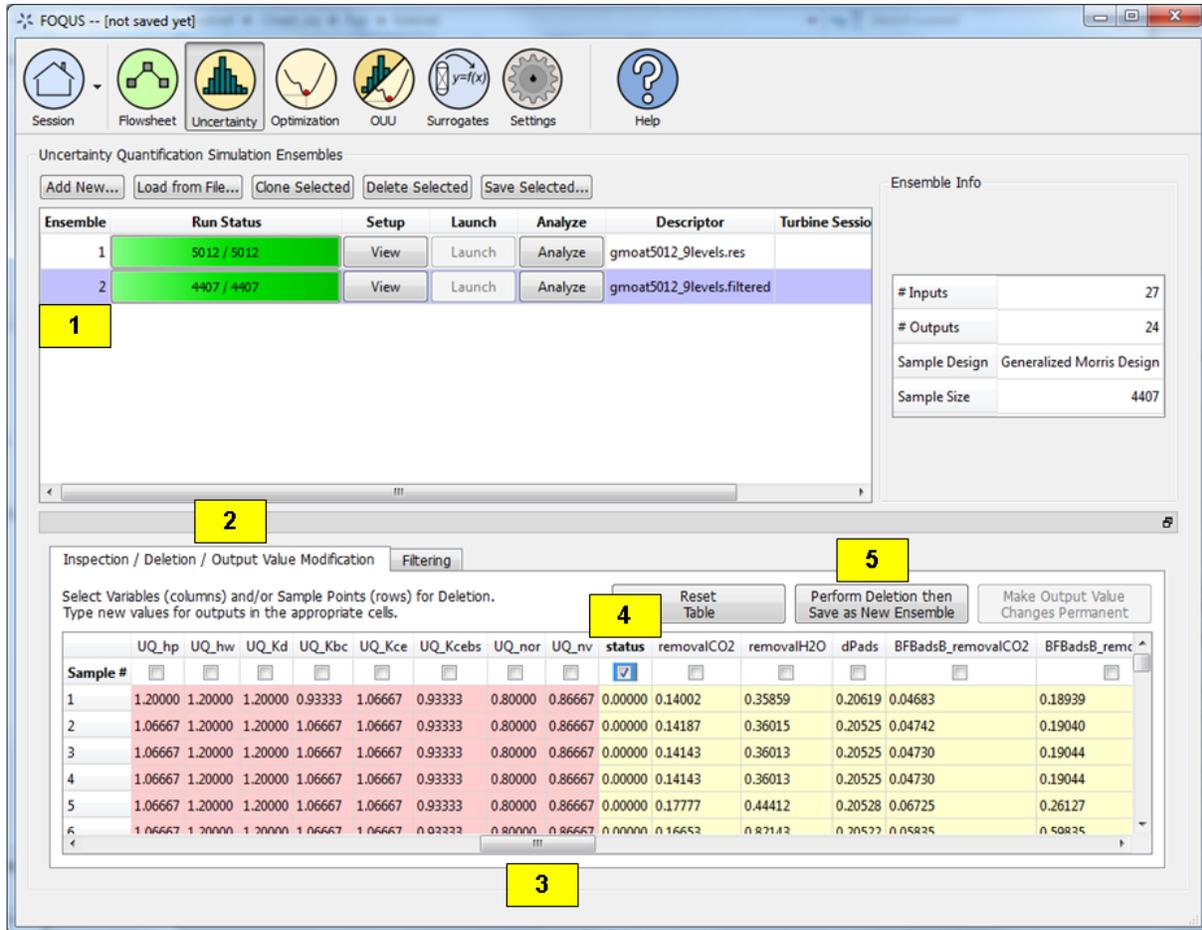


Fig. 36: Data Manipulation, Inspection/Deletion

Output Value Modification

To change the value of an output for a sample or several samples, follow steps below:

1. Select an ensemble.
2. Click the Inspection/Deletion/Output Value Modification tab.
3. Scroll to the right to the outputs.
4. Click on a cell for one of the outputs and enter a new value. Do the same for another cell. Notice that the modified cells turn green. This indicates the cells that have been modified.
5. Click Make Output Value Changes Permanent to permanently change the values. The modified cells will turn yellow, indicating the permanent change. If the user wishes to reset the table and start over before making changes permanent, click the Reset Table.

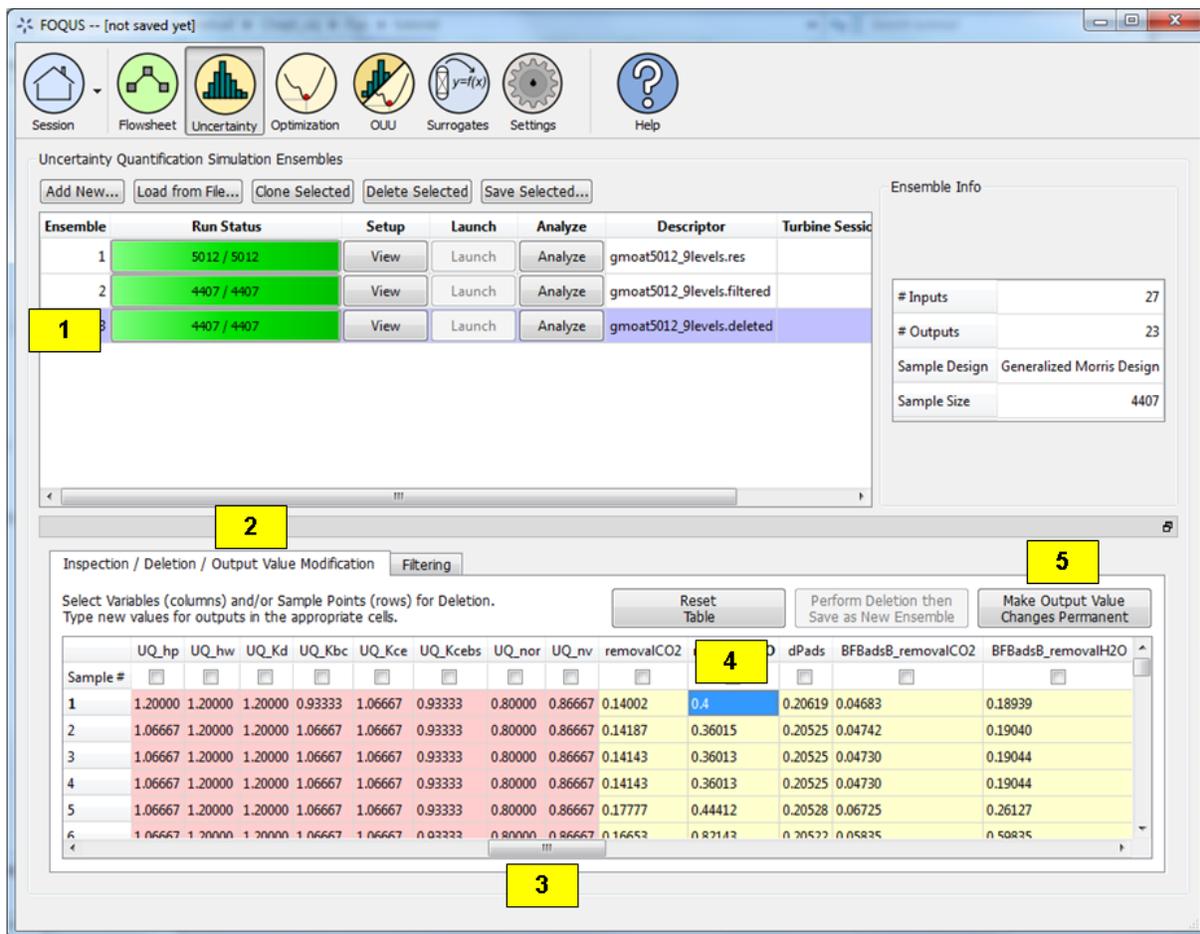


Fig. 37: Data Manipulation, Value Modification

Tutorial 3: Single-Output Analysis

From the Single-Output Analysis Screen, the user can perform analyses that are specific to a particular output of interest. Here, the “removalCO2” output parameter is discussed.

The files for this tutorial are located in: `examples/tutorial_files/UQ/Tutorial_3`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

Parameter Selection

For simulation models that have a large number of input parameters, it is common practice to down-select to a smaller subset of the most important input parameters that are most relevant to the output of interest. This is done so subsequent detailed studies can be performed more efficiently. By using a smaller set of inputs, a smaller set of samples may be needed.

1. From the UQ window, load the file “gmoat5012_9levels.filtered” in `examplestutorial_filesUQTutorial_3`. (This file contains the same set of samples that resulted from data filtering. They are included here to make each demo self-inclusive.)
2. Click **Analysis**. A new page is displayed (Figure [\[fig:uqt_analysis_param\]](#)).
[fig:uqt_analysis_param]
3. Under the Qualitative Parameter Selection section, select “removalCO2” as the output.
4. Select “MOAT” as the method to be used.
5. Click **Compute input importance**. A graph should appear with the results (Figure [\[fig:uqt_param_results\]](#)).
[fig:uqt_param_results]

The bars in the plot represent the importance of a particular input in determining the value of the output. For example, the values of dH3 and dS3 are very important in determining the value of removalCO2, whereas Hce and hp have no affect (the y-axis displays the average changes in the model output as a result of changing the inputs in their respective ranges. For example, from Figure [\[fig:uqt_param_results\]](#), changing dH3 in its range results in an average change in CO₂ removal as much as about 57 percent with a margin of +/- 3 percent). Thus, it would be safe to exclude any inputs that have negligible bar lengths from analysis. Next, down-select the ten most important inputs based on these results. See Section [\[subsubsec:uqt_vardel\]](#) for details. Change the number of samples and scheme as desired and then generate new samples. Click **Launch** to run these samples to obtain another simulation ensemble that can be analyzed.

Ensemble Data Analysis

If the user is interested in the output uncertainty of “removalCO2” based on the uncertainties from the ten most important input parameters, perform uncertainty analysis, which would compute the probability distribution and sample statistics of “removalCO2.”

1. Load “lptau20k_10inputs_4outputs.filtered” from the `examplestutorial_filesUQTutorial_3` folder. Assume this is the file that the user would receive after running the cloned simulation ensemble in which the user has down-selected the ten most important inputs, set the Sampling Scheme to “Quasi-Monte Carlo (LPTAU)”, set the sample size to 20K, and performed data filtering to retain only the samples with the status output equal to “0.”
2. Click Analyze. A new page displays (Figure [\[fig:uqt_analysis_ua\]](#)).
3. Select “Ensemble Data” to indicate that analysis is to be directly performed on the raw sample data.

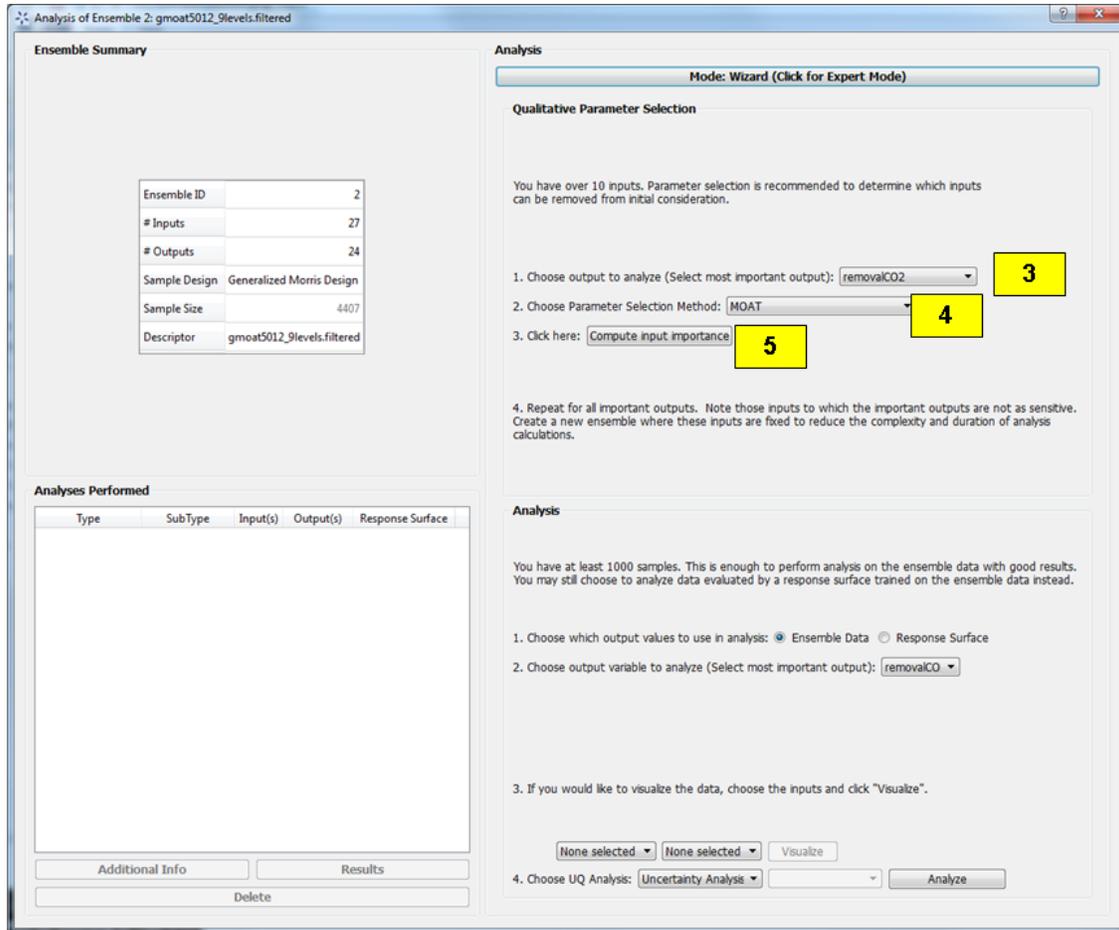


Fig. 38: Analysis Dialog, Parameter Selection

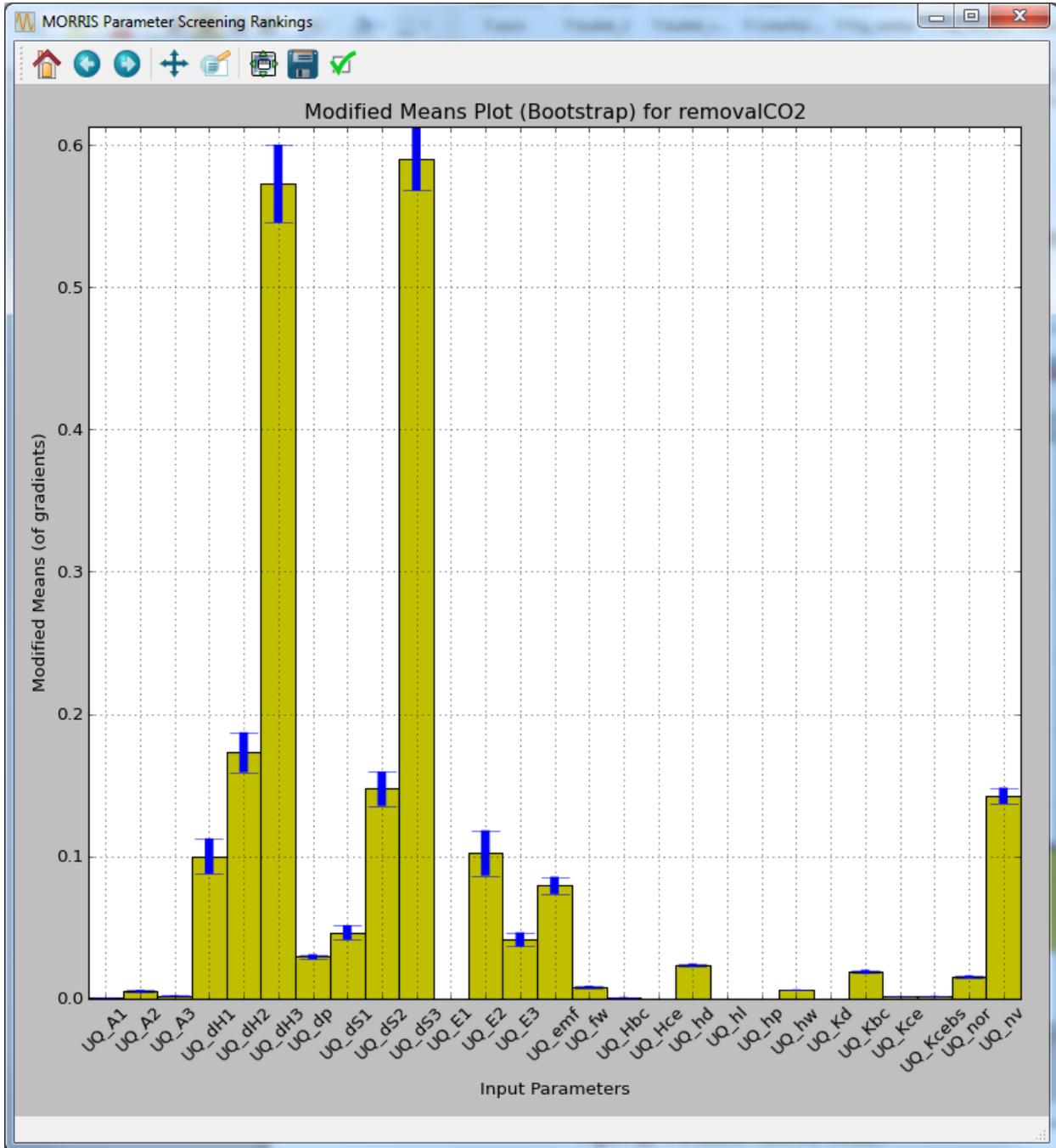


Fig. 39: Parameter Selection Results

4. Select “removalCO2” as the output variable to analyze.
5. Select “Uncertainty Analysis” and then click Analyze.

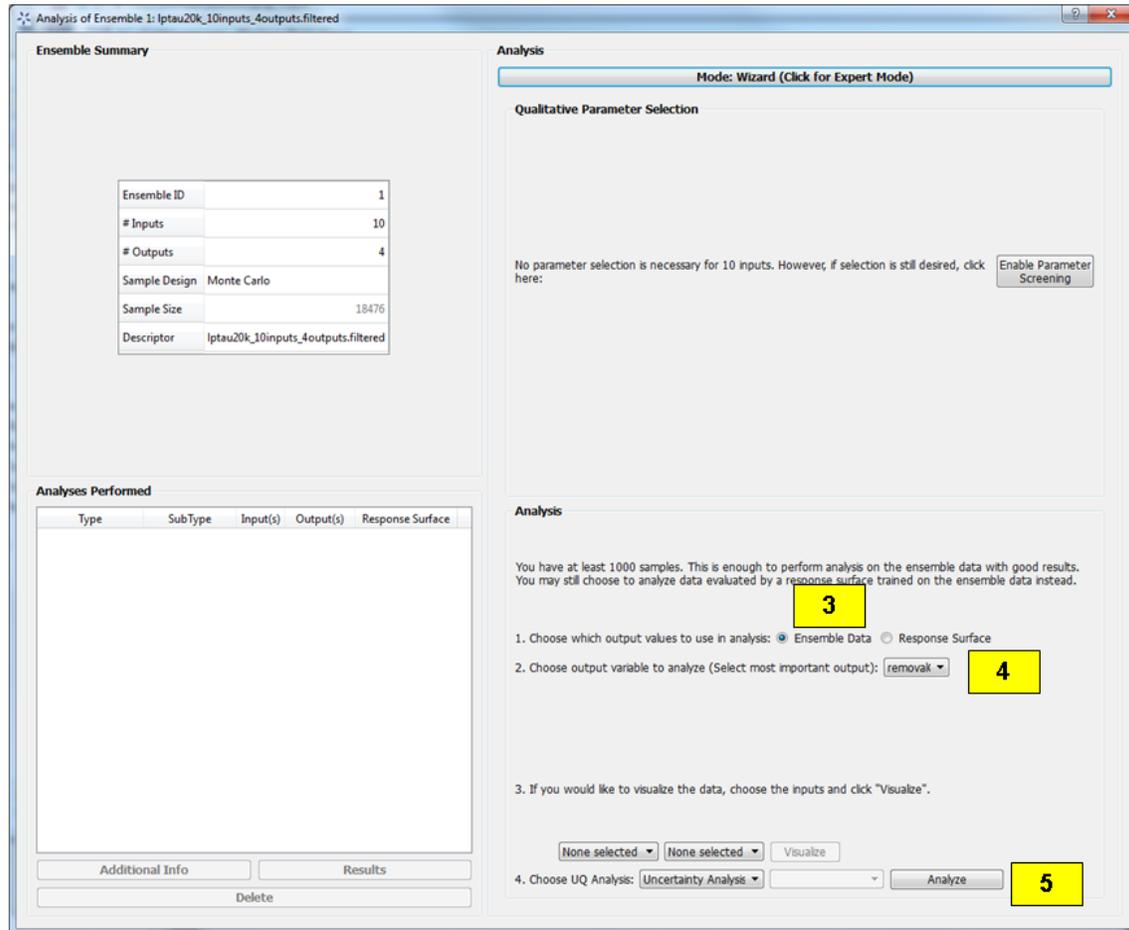


Fig. 40: Analysis Dialog, Ensemble Data Uncertainty Analysis

[fig:uqt_analysis_ua]

Once uncertainty analysis is complete, results display (Figure [fig:uqt_ua_results]) illustrating the probability distribution function (PDF), cumulative distribution function (CDF), and the sufficient statistics (e.g., mean, standard deviation) of “removalCO2” (top left corner of the PDF plot). This is used to evaluate if the output uncertainty is acceptable. If the output uncertainty is too great or the user prefers the system to operate within a higher percentage of capture, pursue further analyses to understand the relationships between the inputs and outputs, and investigate what can be done to reduce the output uncertainties by reducing the input uncertainties.

[fig:uqt_ua_results]

Next, the user may apply variance-based sensitivity analysis to quantify each input’s contribution to the output variance:

enumerate

6. From the bottom of the “Analysis” section, select “Sensitivity Analysis.”
7. There are three options for sensitivity analysis: (1) first-order, (2) second-order, and (3) total-order. First-order analysis examines the effect of varying an input parameter alone. Second-order analysis examines the effect of varying pairs of input parameters. Total-order analysis examines all interactions’ effect of varying an input

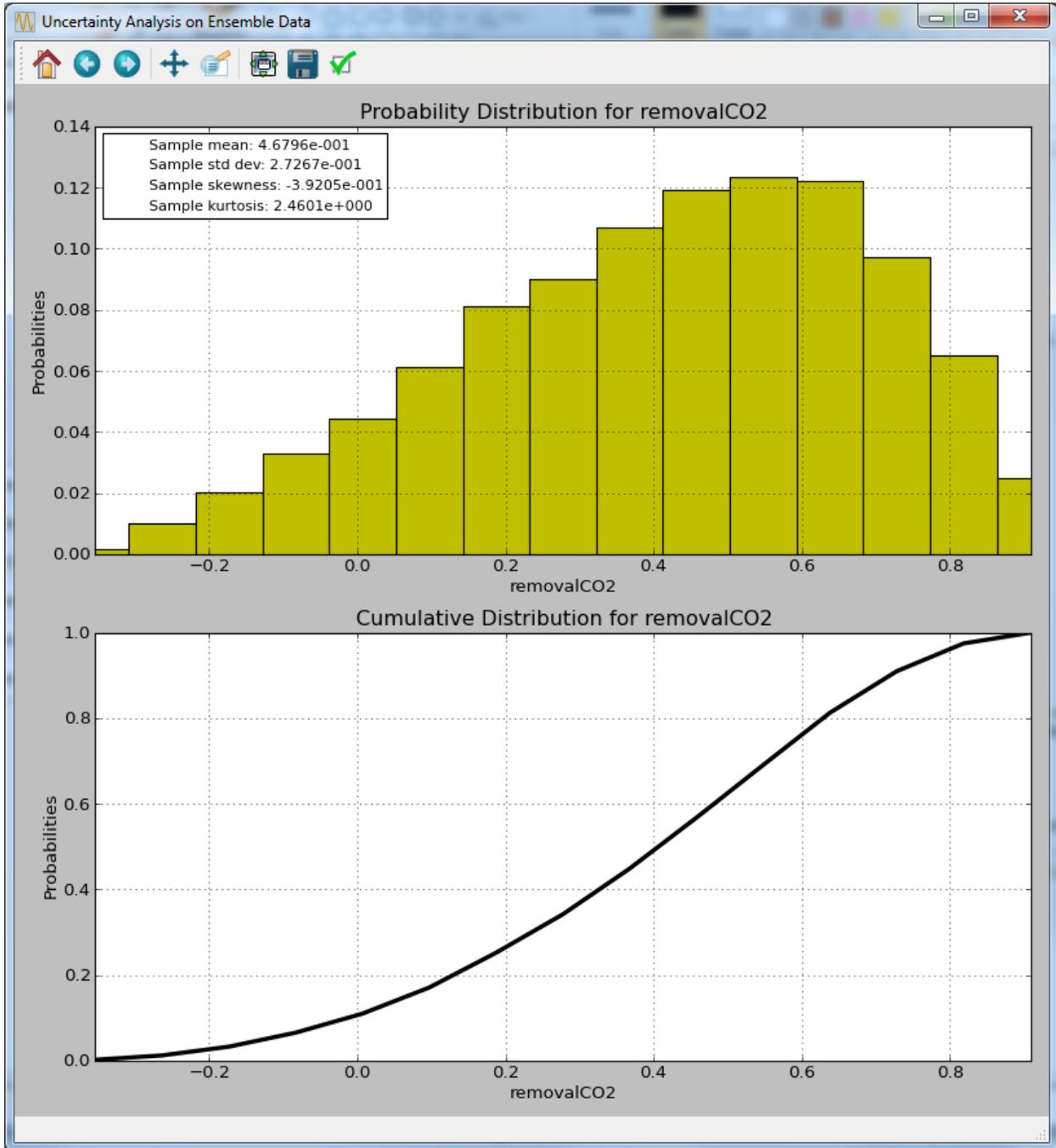


Fig. 41: Ensemble Data Uncertainty Analysis Results

parameter alone and as a combination with any other input parameters. For this demonstration, select “Total-order” and click Analyze. The total sensitivity indices display in a graph. Note: If the simulation ensemble has more than ten inputs, “Total-order” is disabled (since any reasonable sample size is not sufficient). Additionally, since quantitative sensitivity analysis in general requires large ensembles with many samples (thousands or more), ensemble sensitivity analysis (without the use of response surfaces) is often less practical and accurate than response surface based analyses. The result is illustrated in Figure[fig:uqt_sa_results].

[fig:uqt_sa_results]

These results confirm that “removalCO2” is more sensitive to “dH3” and “dS3” than other inputs. (The y-axis displays an approximate percentage of output variance attributed to each individual parameter. Since total sensitivity includes higher order interaction terms with other parameters, the sum of these total sensitivity indices usually exceeds 1.)

Ensemble Data Visualization

1. In this release, ensemble data visualization is only available in “Expert” mode. At the top of the “Analyze” page, toggle the bar to expert mode and select “removalCO2” as the output. Next, to “Visualize Data,” choose an input (e.g., “UQ_dH1”) and click **Visualize** for a 2-D scatter plot of “removalCO2” versus that input (Figure [fig:uqt_splot1_results]).

[fig:uqt_splot1_results]

2. Next, select a second input (e.g., “UQ_dH2”) and click **Visualize** for a 3-D scatter plot of “removalCO2” versus the two inputs. (Note: The input selections must be unique for the **Visualize** button to be enabled.) Figure [fig:uqt_splot2_results] shows the results.

[fig:uqt_splot2_results]

The plot in Figure [fig:uqt_splot2_results] can be rotated by clicking and dragging.

Tutorial 4: Response Surface Based Analysis

For simulation models that are expensive to run, response surface analysis can be a resourceful option. To construct a response surface, a space-filling sampling design is desired. For example, quasi-Monte Carlo (LPTAU) or Latin hypercube sampling schemes are recommended. Additionally, there are several possibilities for curve fitting methods. If the sample size is relatively small, polynomial regression or Gaussian process (if installed as part of PSUADE) is preferred. Alternatively, if the sample size is large enough (one hundred or more), cubic splines (if installed) may also be feasible.

The file for this tutorial is `lptau100_10inputs_4outputs.dat`, and this file is located in: `examples/tutorial_files/UQ/Tutorial_4`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

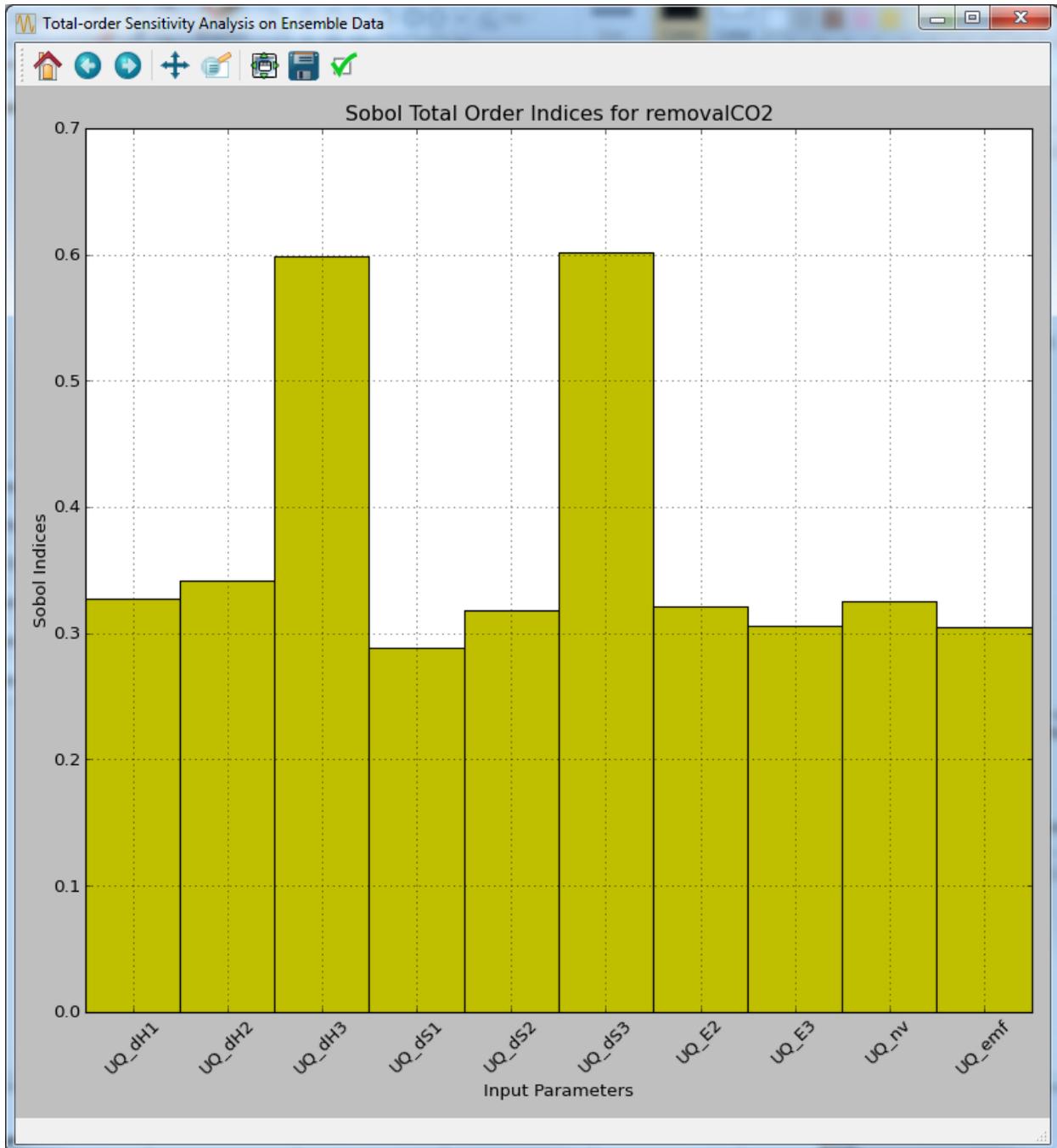


Fig. 42: Ensemble Data Total-order Sensitivity Analysis Results

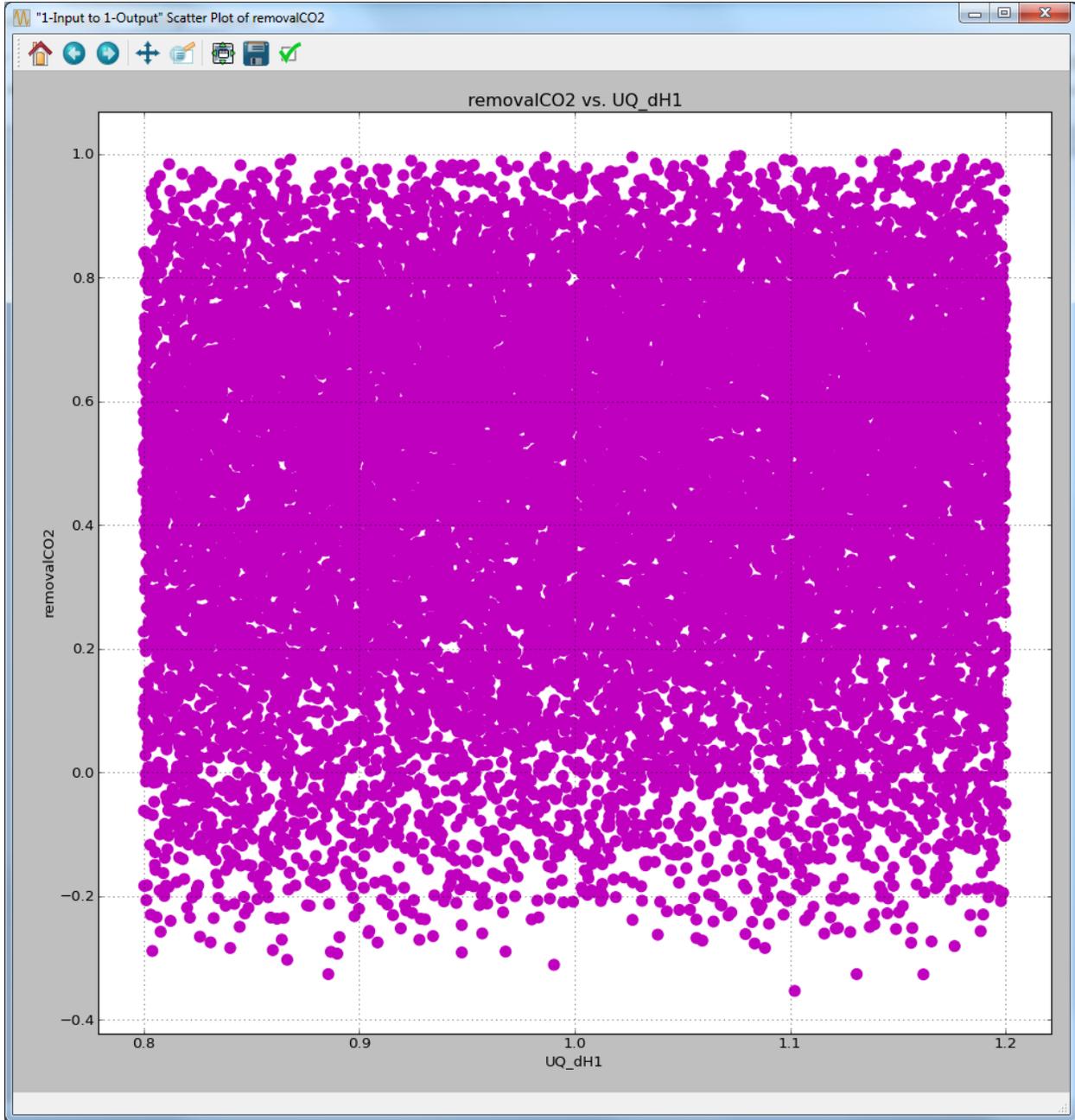


Fig. 43: Ensemble Data Visualization of One Input

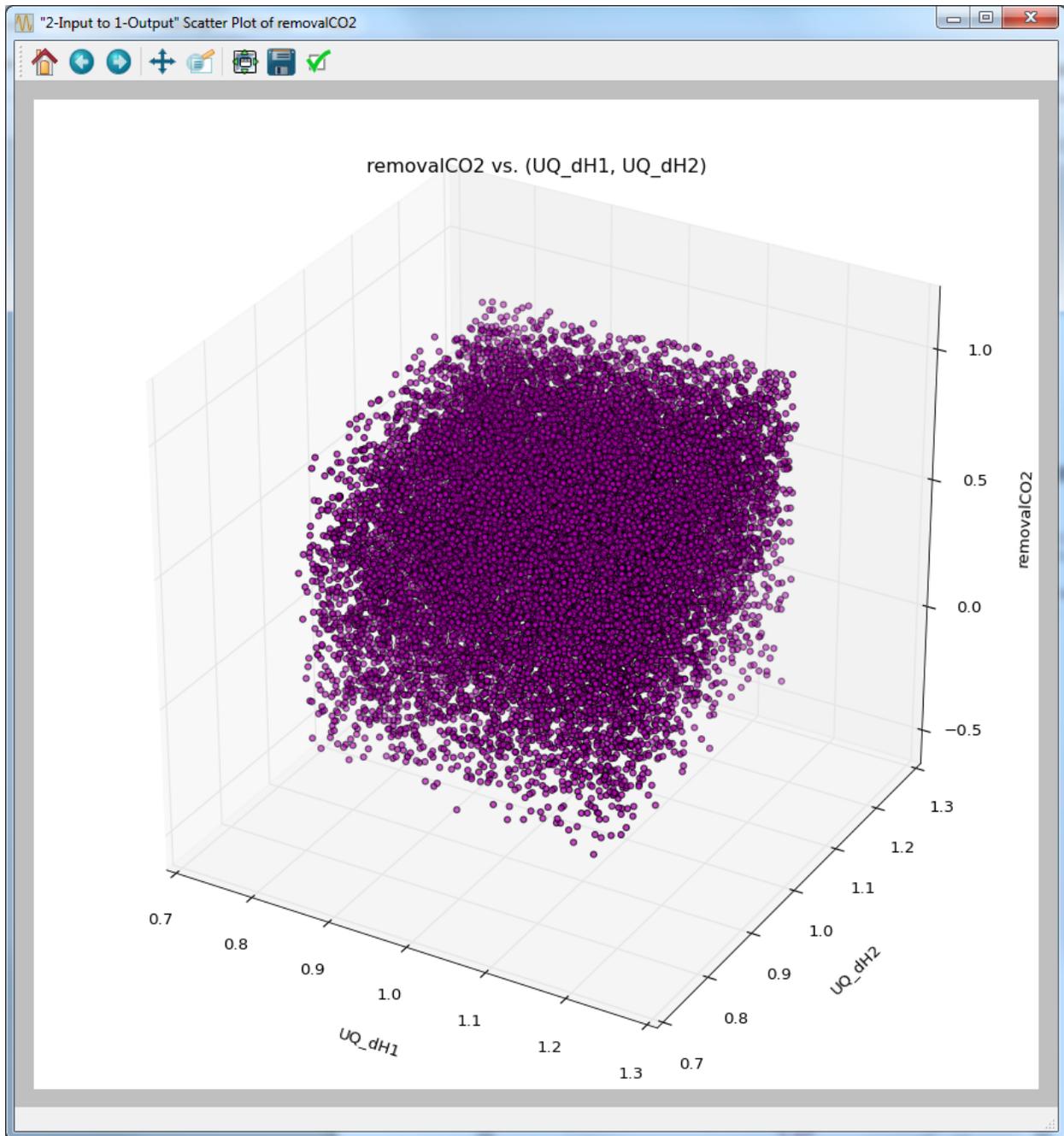


Fig. 44: Ensemble Data Visualization of Two Inputs

Response Surface Model Validation

To proceed with response surface based analysis, the user needs to find a suitable response surface with which to approximate the input-to-output mapping. Validation is performed to see how well a particular response surface can predict a subset of the withheld data.

1. Load the “lptau100_10inputs_4outputs.dat” file. Note: This is an extremely small simulation ensemble, as this is used to highlight the differences (in validation results) between a good response surface and a bad one.
2. Click Analyze for the current ensemble. A new dialog page displays (Figure[fig:uqt_rs_validate]).
3. Under “Analysis” (bottom section), under Step 1, select “Response Surface.”

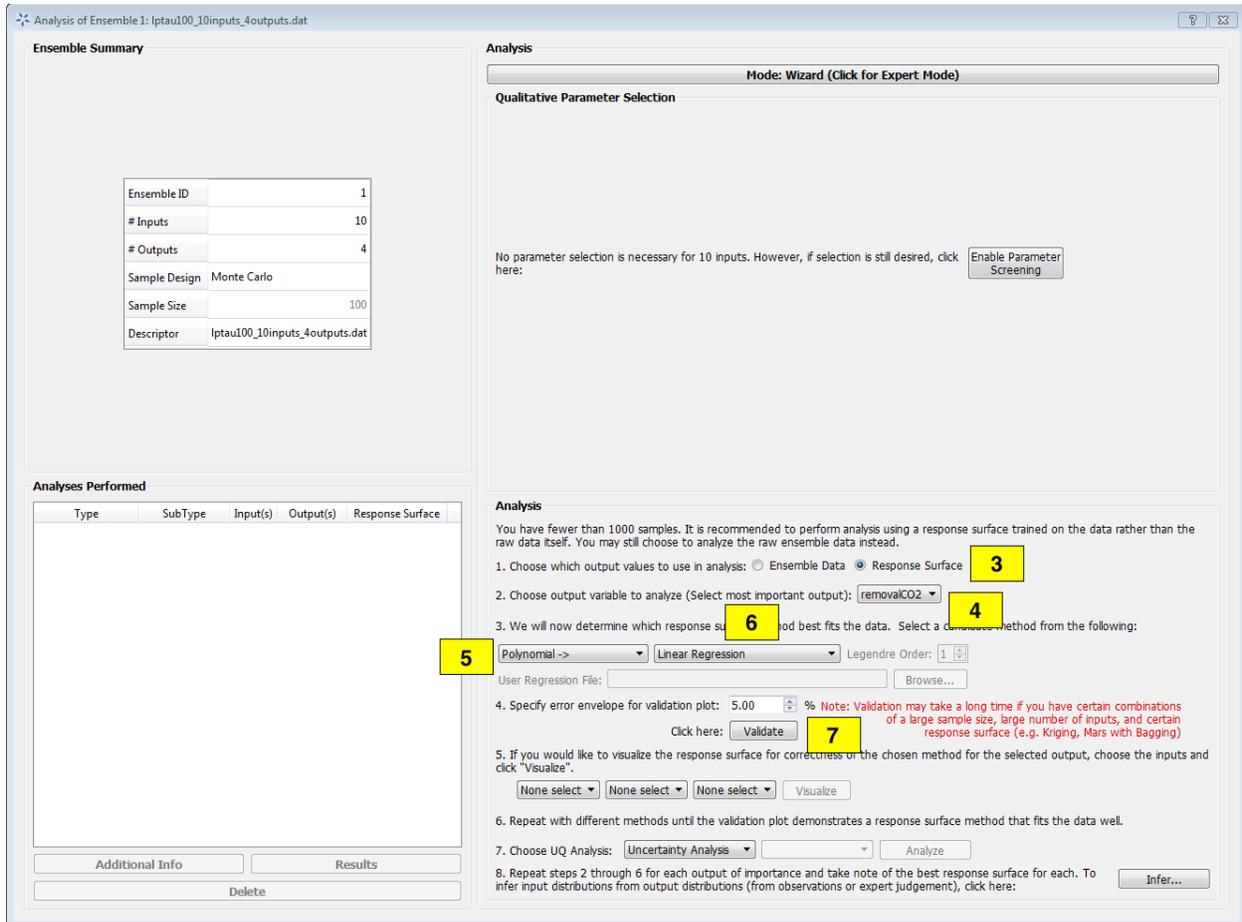


Fig. 45: Analysis Dialog, Response Surface Validation of Linear Model

[fig:uqt_rs_validate]

4. Under Step 2, select “removalCO2” as the output for analysis.
5. Under Step 3, select “Polynomial” for response surface method.
6. There are multiple types of polynomial response surfaces, with increasing complexity as the user navigates down the list. For now, select “Linear” in the next drop-down list.
7. Insert 5.00 as the error envelope for the validation plot. Click Validate. The result is illustrated in Figure[fig:uqt_rs_validate_results].

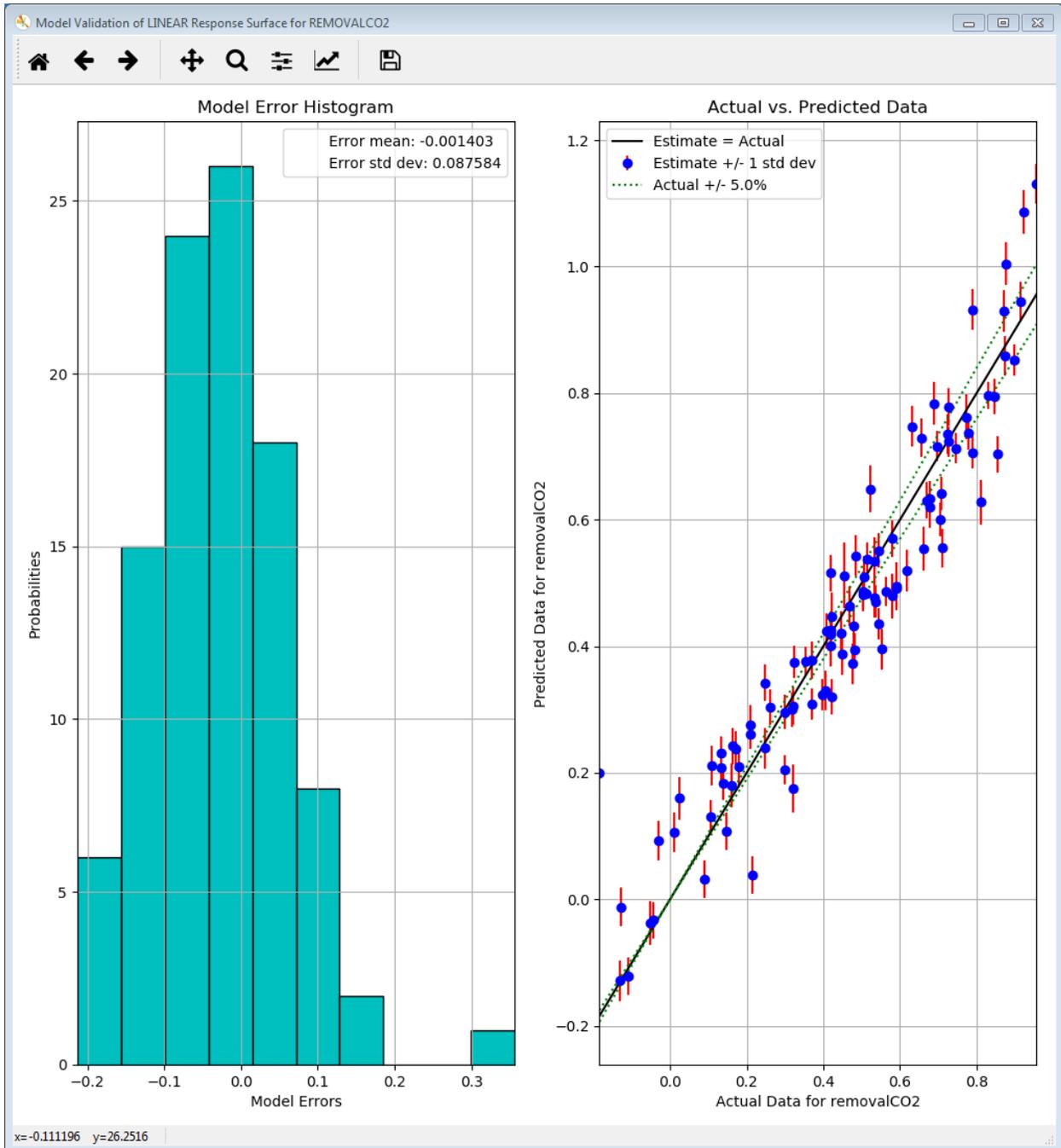


Fig. 46: Linear Response Validation Results

[fig:uqt_rs_validate_results]

The cross-validation results for the linear regression model are displayed as a histogram of errors to the left and a plot of predicted values versus actual values to the right. The histogram displays the cross validation error distribution, which provides the user information on what the errors are like overall. If this distribution is not centered on zero, there may be a systematic bias in the response surface model. If the distribution is too wide, it is not a good fit. As for the plot of predicted values versus actual values, the more closely the points are to the diagonal, the better the fit. Most response surface models, with the exception of MARS, also provide uncertainty information about the response surface. The vertical error bars on the left plot reflect the uncertainty in the linear response's predictions.

In summary, these two figures should provide sufficient information for the user to judge how good the fit is. As is apparent in the figures, the linear model consistently overestimates and thus is an ill-suited response surface to model our data. In general, the user may use a few response surface methods to see which method gives the best fit.

Response Surface Based Uncertainty Analysis

These capabilities are similar to those for ensemble data analysis. The difference is that the results are now derived from a much larger ensemble that is computed from the response surface. With the 100 samples from the ensemble data, a response surface is trained and is used to generate 100K samples internally to compute the results for uncertainty and sensitivity analyses. (Note: Validation must be performed before these analyses are available.)

After the response surface validation step, select “Uncertainty Analysis” to be the UQ analysis in Step 7 of “Analysis” (Figure [fig:uqt_rs_validate]). Click **Analyze** and a distribution representing the output uncertainty will be displayed (Figure [fig:uqt_rsua_results]).

[fig:uqt_rsua_results]

Compare the response surface based uncertainty results (Figure [fig:uqt_rsua_results]) to the results from ensemble data analysis (Figure [fig:uqt_ua_results]). The two main differences are easily seen.

- Two PDFs on top plot: A response surface (in this case, linear regression) is used to predict the output values corresponding to the input samples. From the validation step (left plot of Figure [fig:uqt_rs_validate_results]). Note: There is error associated with the response surface's predictions. This error is propagated in uncertainty analysis, in the form of standard deviations around the predicted output values (i.e., the means). Accordingly, two histograms are presented: The “mean PDF” represents the output probability distribution computed from the response surface's predicted output values only, without consideration for the uncertainties surrounding these predicted values. The “ensemble PDF” represents the output probability distribution that encompasses the uncertainties surrounding these predicted values. In most cases, the ensemble PDF should have a larger spread because it is accounting for more uncertainties (i.e., those that stem from the approximations inherent in the response surface).
- Multiple cumulative distribution functions (CDFs) on bottom plot: The “mean CDF” is constructed from a cumulative sum on the mean PDF in the top plot. Since each predicted output value (i.e., the mean) has an associated standard deviation, this information is used to construct other PDFs that correspond to output values that are +/- 1, 2, and 3 standard deviations from the mean. These PDFs are then converted to CDFs and shown as colored lines. These colored lines provide an uncertainty “envelope” around the mean CDF.

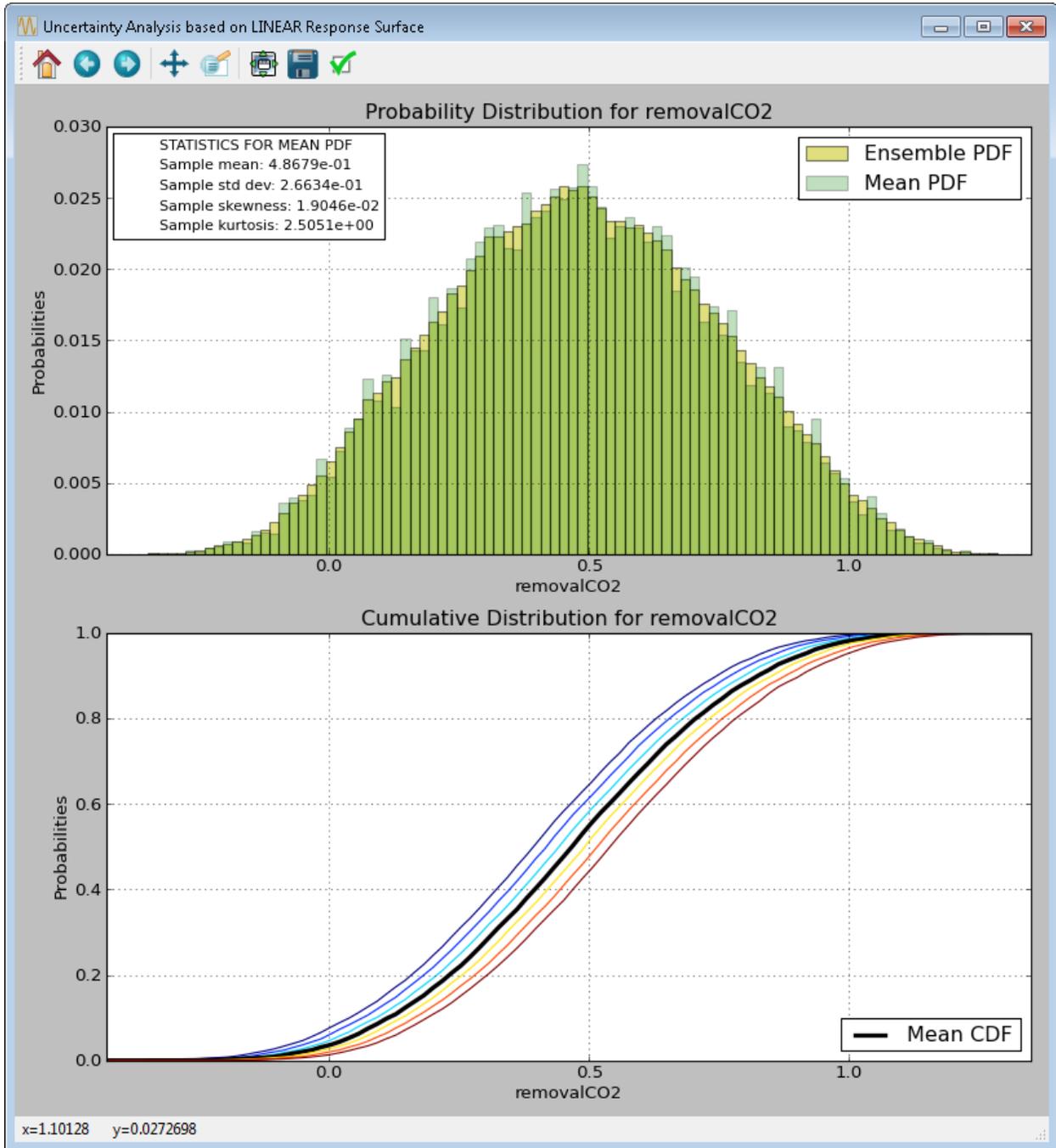


Fig. 47: Response Surface Based Uncertainty Analysis Results

Response Surface Based Mixed Epistemic-Aleatory Uncertainty Analysis

In “Expert Mode”, the user can perform more advanced uncertainty analysis that handles both epistemic and aleatory uncertainties. To do so, the user will need to designate the uncertainty type (epistemic or aleatory) for each uncertain input. In general, epistemic uncertainties are reducible uncertainties that arise due to lack of knowledge, such as simplifying assumptions in a mathematical model. Therefore, epistemic uncertainty is often characterized by upper and lower bounds. On the other hand, aleatory uncertainties are irreducible uncertainties that represent natural, physical variability in the phenomenon under study. As such, aleatory uncertainties are often characterized by distributions. Hence, the user is required to provide a PDF for each aleatory input. (In FOQUS, with the exception of mixed epistemic-aleatory uncertainty analysis, all uncertain inputs are treated as aleatory inputs.)

To perform mixed epistemic-aleatory uncertainty (Figure *Response Surface Based Mixed Epistemic-Aleatory Uncertainty Analysis*), switch to “Expert Mode” by clicking the **Mode** button that toggles between the analysis modes. After response surface validation, select “Uncertainty Analysis” in the first **Choose UQ Analysis** drop-down list, then “Epistemic-Aleatory” in the secondary drop-down list, for the UQ analysis. In the input table, designate the parameter **Type** (“Epistemic”, “Aleatory” or “Fixed”) and the corresponding information for each input. Once complete, click **Analyze**. In this tutorial we consider dH1 & dH2 as epistemic uncertain parameters, and rest of them are aleatory.

The screenshot displays the FOQUS software interface for performing a mixed epistemic-aleatory uncertainty analysis. The main window is titled "Analysis of Ensemble 1: lptau100_10inputs_4outputs.dat".

Ensemble Summary:

Ensemble ID	1
# Inputs	10
# Outputs	4
Sample Design	Monte Carlo
Sample Size	100
Descriptor	lptau100_10inputs_4outputs.dat

Analyses Performed:

Type	SubType	Input(s)	Output(s)	Response Surface
1	RS Validation		removalCO2	Linear Regression

Analysis Configuration (Expert Mode):

- Mode:** Expert (Click for Wizard Mode)
- Select Output under Analysis:** removalCO2
- Qualitative Parameter Selection:** Choose Parameter Selection Method: MARS Ranking; Compute input importance
- Ensemble Data Analysis:** Choose UQ Analysis: Uncertainty Analysis; Visualize Data: None selected
- Response Surface (RS) Based Analysis:**
 - Select RS: Polynomial -> Linear Regression
 - Legendre Polynomial Order: 1
 - MARS Number of basis functions: 100
 - MARS Degree of interaction: 8
 - User Regression File: [Browse...]
 - Validate: Error Envelope: 5.00 %; Use test set for output removalCO2 [Browse...]
 - Number of Cross-Validation Groups: 10
 - Visualize RS: [None selected]
 - Choose UQ Analysis: Uncertainty Analysis -> Epistemic-Aleatory

Input Table:

Input Name	Type	Value	PDF	PDF Param1	PDF Param2	Min	Max
1 UQ_dH1	Epistemic	1	Uniform			0.8	1.2
2 UQ_dH2	Epistemic	1	Uniform			0.8	1.2
3 UQ_dH3	Aleatory	1	Uniform			0.8	1.2
4 UQ_dS1	Aleatory	1	Uniform			0.8	1.2
5 UQ_dS2	Aleatory	1	Uniform			0.8	1.2
6 UQ_dS3	Aleatory	1	Uniform			0.8	1.2
7 UQ_E2	Aleatory	1	Uniform			0.8	1.2
8 UQ_E3	Aleatory	1	Uniform			0.8	1.2

Bayesian Inference: Infer...

Fig. 48: Response Surface Based Mixed Epistemic-Aleatory Uncertainty Analysis

The results of mixed epistemic-aleatory uncertainty analysis is a plot (Figure *Response Surface Based Mixed Epistemic-Aleatory Uncertainty Analysis Results*) containing multiple CDFs. In the mixed analysis, the epistemic inputs are sampled according to their lower and upper bounds. Each sample point spawns a response surface based uncertainty analysis, in which the epistemic inputs are fixed at their sampled value and the aleatory input uncertainties

are propagated to generate a CDF that represents the output uncertainty. A slider is provided for the user to extract the probability range corresponding to a particular value of the output.

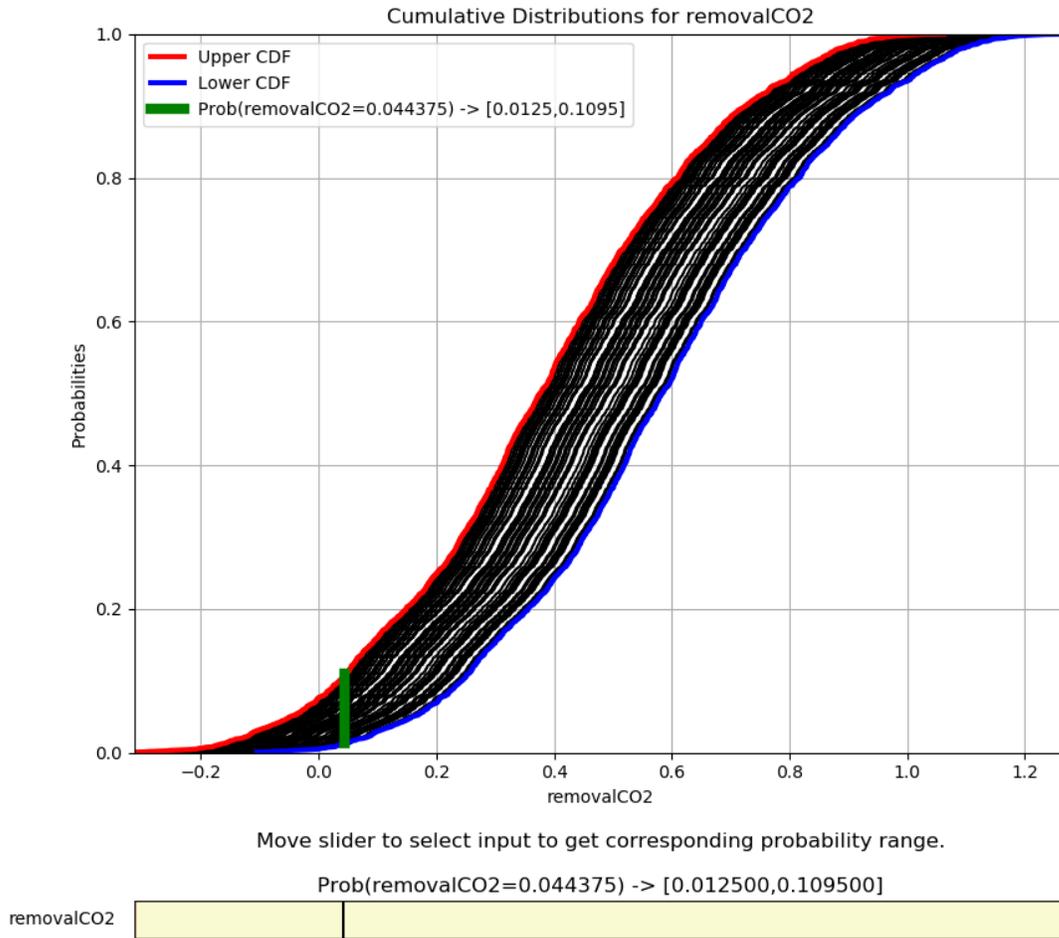


Fig. 49: Response Surface Based Mixed Epistemic-Aleatory Uncertainty Analysis Results

Response **Surface** **Based** **Sensitivity** **Analysis**

For quantitative sensitivity analysis, follows these steps:

1. In the Choose UQ Analysis drop-down list (Step 6 of “Analysis”), select “Sensitivity Analysis.”
2. In the next drop-down list, select “First-order” and click Analyze. (This analysis may take a long time depending on the sample size and the response surface used.)

Prediction errors are associated with the response surface’s predictions of the output values (left plot of Figure [fig:uqt_rs_validate_results]). Earlier, it was observed that the response surface error contributed to the output uncertainty, leading to a larger spread in the output PDF (top plot of Figure [fig:uqt_rsua_results]). In Figure [fig:uqt_rssa_results], the response surface error contributed to uncertainty (shown as blue error bars) surrounding each input’s contribution to the output variance (shown as yellow bars).

[fig:uqt_rssa_results]

Response **Surface** **Based** **Visualization**

The response surface that has been validated can also be visualized.

1. Select one input next to “Visualize Response Surface.”
2. Click **Visualize** to display a 2-D line plot that displays “removalCO2” versus the selected input.
[fig:uqt_rs1_results]
3. Select another input next to the first one for a 2-D response surface visualization.
4. Click **Visualize** to display a figure with a 3-D surface plot and a 2-D contour plot (Figure [fig:uqt_rs2_results]).
[fig:uqt_rs2_results]
5. Select another input next to the second one for a 3-D response surface visualization.
6. Click **Visualize** to display a 3-D isosurface plot. Move the slider to see the points in the 3-D input space that fall within the small range of “removalCO2” (Figure [fig:uqt_rs3_results]).
[fig:uqt_rs3_results]

Tutorial **5:** **Bayesian** **Inference**

For each output variable, the user specifies an observed value (from physical experiments) with the associated uncertainties (in the form of standard deviation), if applicable. Whether standard inference or SolventFit is selected, the tool will launch a Markov Chain Monte Carlo (MCMC) algorithm to compute the posterior distributions of the uncertain input parameters. These input posterior distributions represent a refined hypothesis about the input uncertainties in light of what was previously known (in the form of input prior distributions) and what was observed currently (in the form of noisy outputs).

The file for this tutorial is **lptau5k_10inputs_4outputs.filtered**, and this file is located in:
`examples/tutorial_files/UQ/Tutorial_5`

Note: The **examples/** directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

1. Load the “lptau5k_10inputs_4outputs.filtered” file from the above-mentioned folder.

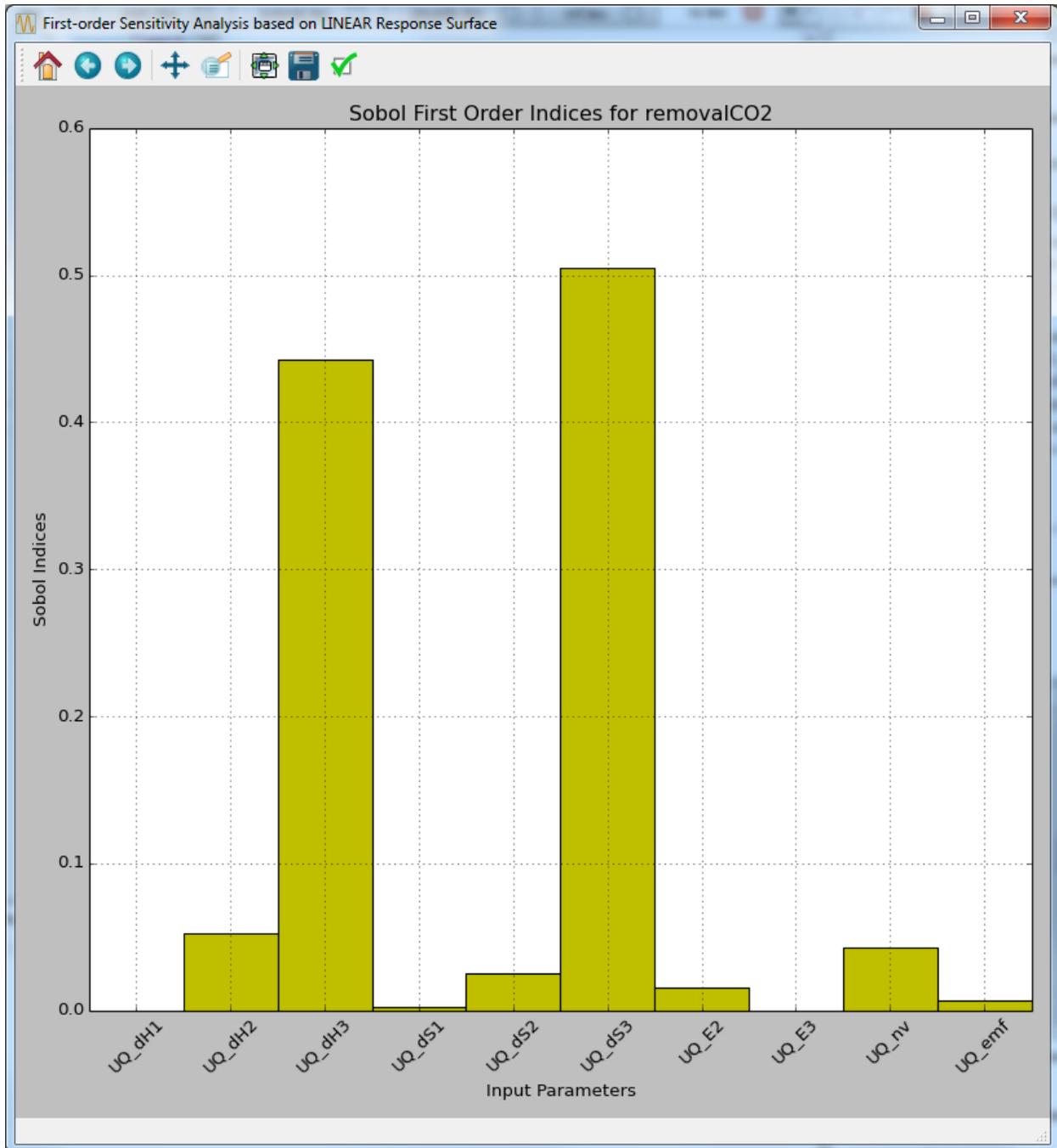


Fig. 50: Response Surface Based First-order Sensitivity Results

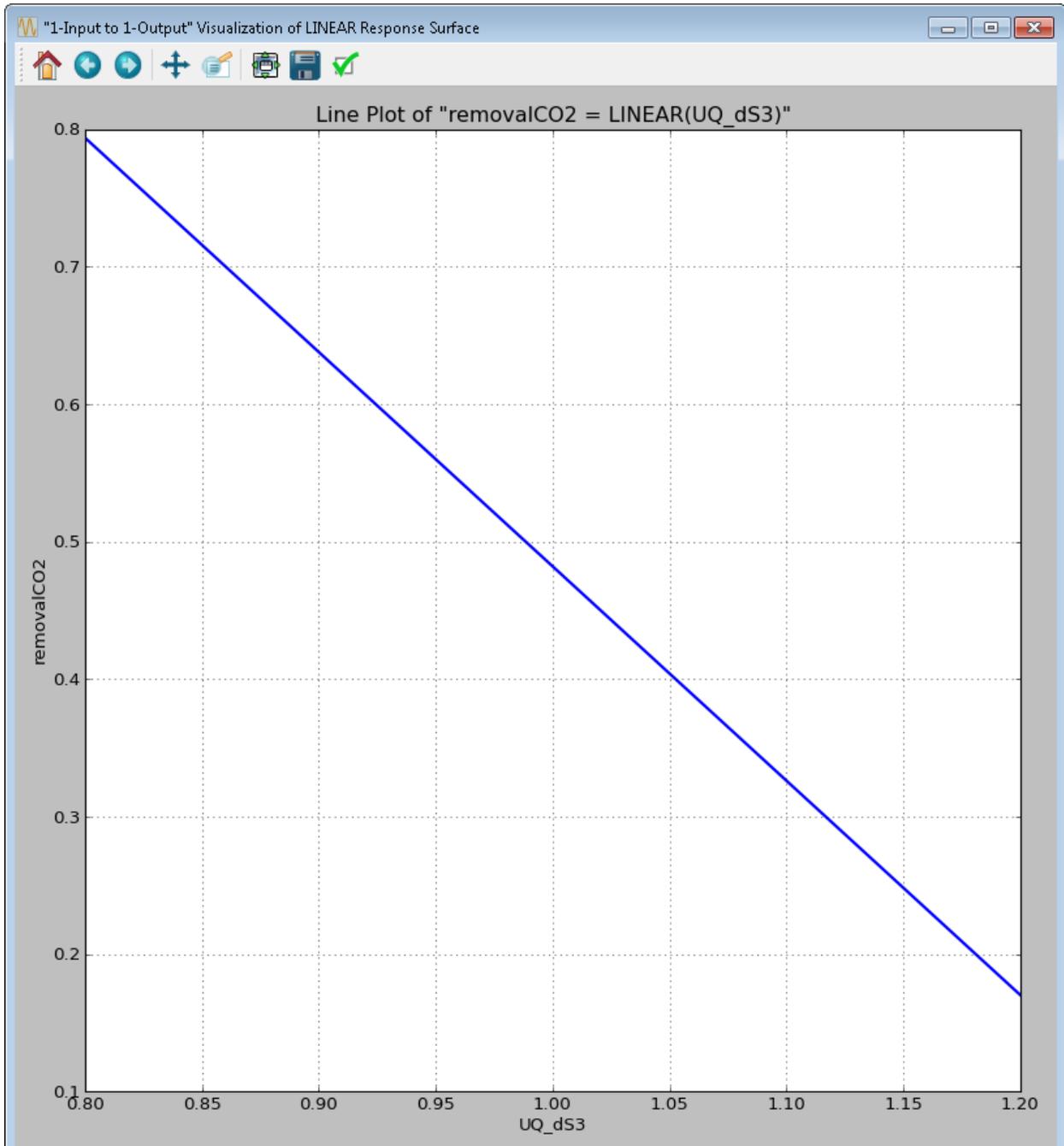


Fig. 51: 1-D Response Surface Visualization

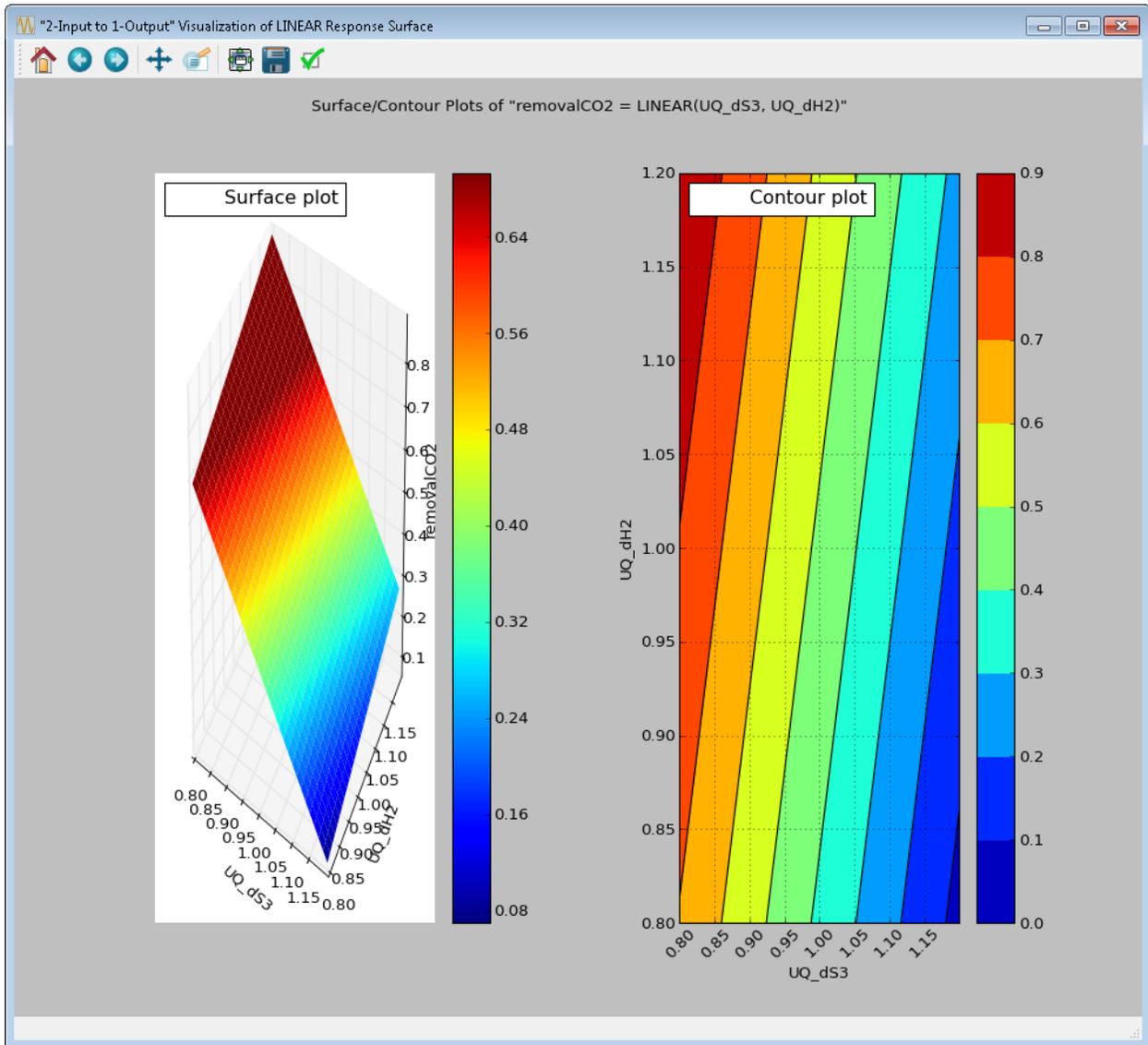


Fig. 52: 2-D Response Surface Visualization

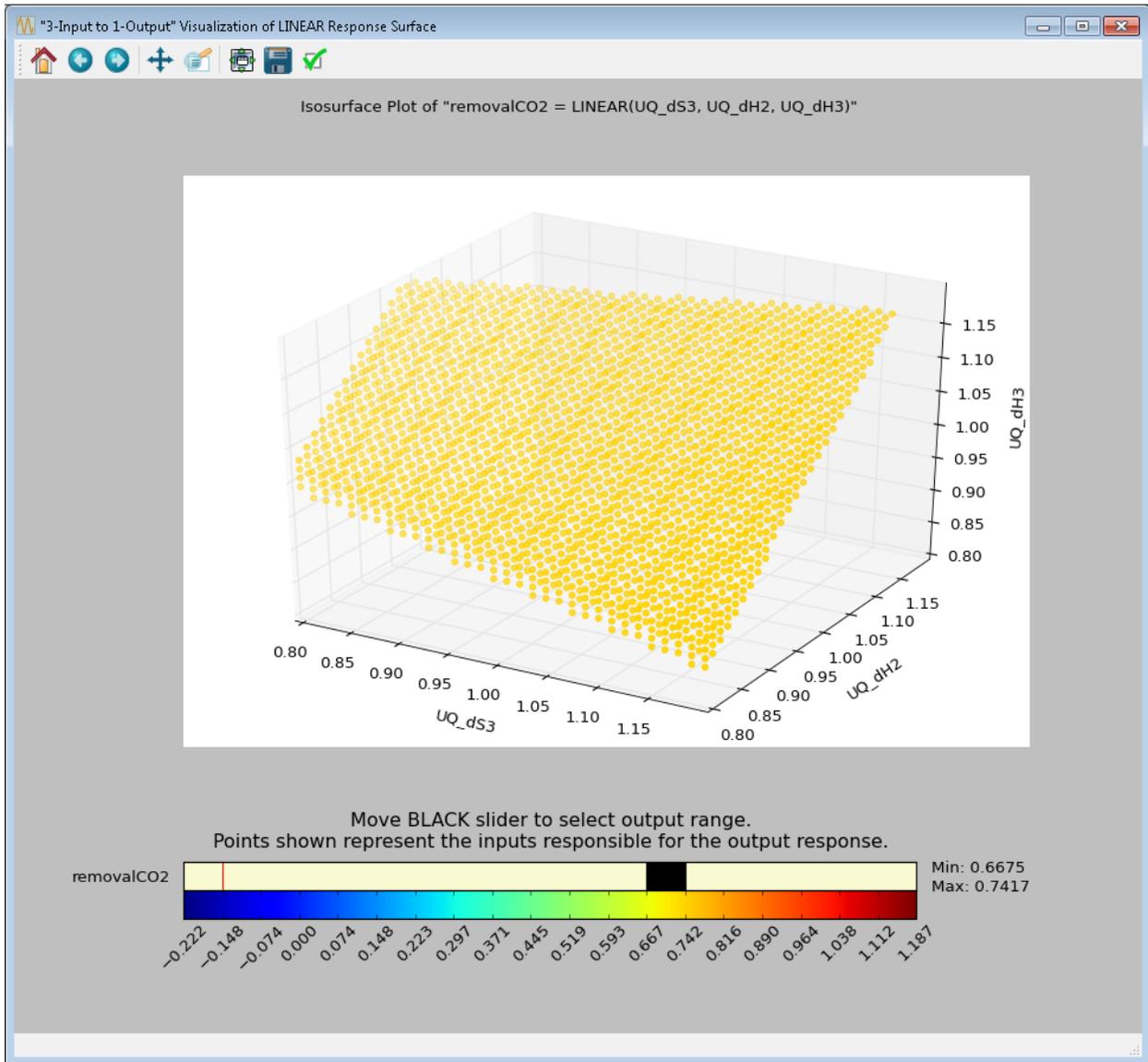


Fig. 53: 3-D Response Surface Visualization

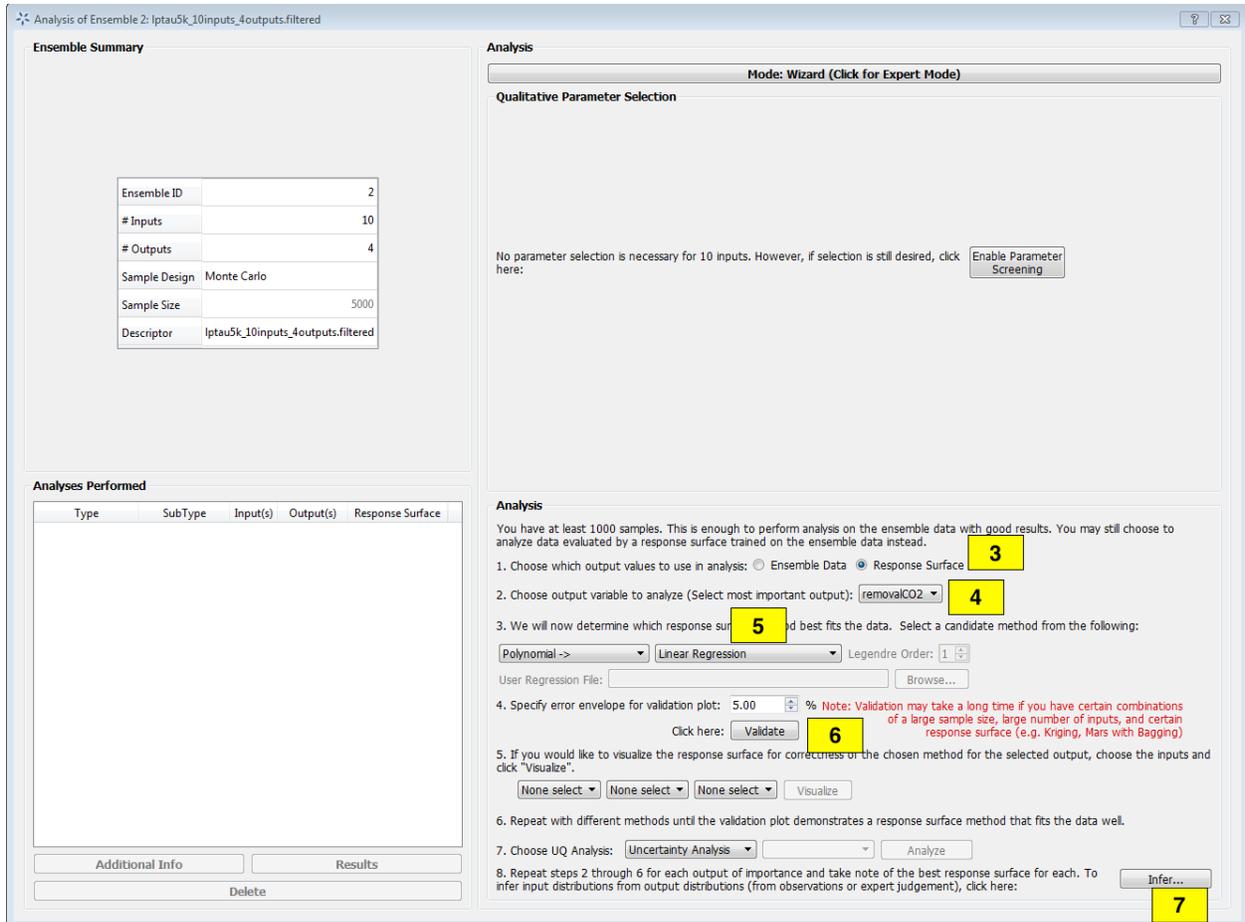


Fig. 54: Analysis Dialog, Bayesian Inference

2. Click **Analyze** for the current ensemble and a new dialog box displays (Figure [\[fig:uqt_analysis_infer\]](#)).
[fig:uqt_analysis_infer]
3. Select “Response Surface” in the “Analysis” section.
4. Select “Output variable to analyze” to be “removalCO2.”
5. Select “Linear Regression” as the response surface.
6. Insert 5.00 as the error envelope for the validation plot. Click Validate. The GUI allows the user to proceed with Bayesian inference after one input has been validated; however, the user may want to validate all outputs since they are all used in the inference.
7. Once validation is completed, click Infer at the lower right corner, which displays a new dialog box (Figure [\[fig:uqt_infer\]](#)).
8. In the Output Settings table (on the left), select the second, third, and fourth outputs as the observed outputs. The user can experiment with using different response surface models (for example, linear polynomials) to approximate the mapping from inputs to each of the outputs.
9. In the Input Settings table (on the right), designate input types (variable, design, or fixed) and if necessary, switch to Expert Mode to revise the prior distribution on the input parameters. The prior distribution represents knowledge that the user possesses about the inputs before observational data (from experiments) has been incorporated into this knowledge. If the user does not have any updated knowledge about the simulation ensemble, it is OK to leave the table as is.
10. In the **Observations** table (in the middle), select the number of experiments from which the user can get observational data. In essence, if the user has N observations, then N should be set as the number of experiments. The table will then populate columns for design inputs (if any) and observed outputs. Currently, only normal distribution is supported as the noise model for observations. Enter the mean and standard deviation for each of these observations. For convenience, the mean and standard deviation values are prepopulated with the results from uncertainty analysis. These values have been provided as a sanity check for the user, in case the observation for a particular output is way out of range from these distributions.
[fig:uqt_infer]
11. To save an input sample drawn from the posterior distribution, select the Save Posterior Input Samples to File checkbox and select a location and file name to store the sample.
12. Click Infer to start the analysis. Inference can take a long time; thus, a stop feature has been implemented. Once inference starts, the Infer button changes to Stop. To stop inference calculations, click Stop which changes the button back to Infer, allowing the user to restart the calculations from scratch. If inference is allowed to run its course, its results are interpolated to produce heat maps (off-diagonal subplots in Figure [\[fig:uqt_infer_results\]](#)) for visualization. This interpolation step can take a few minutes and while it is running, Infer is disabled.

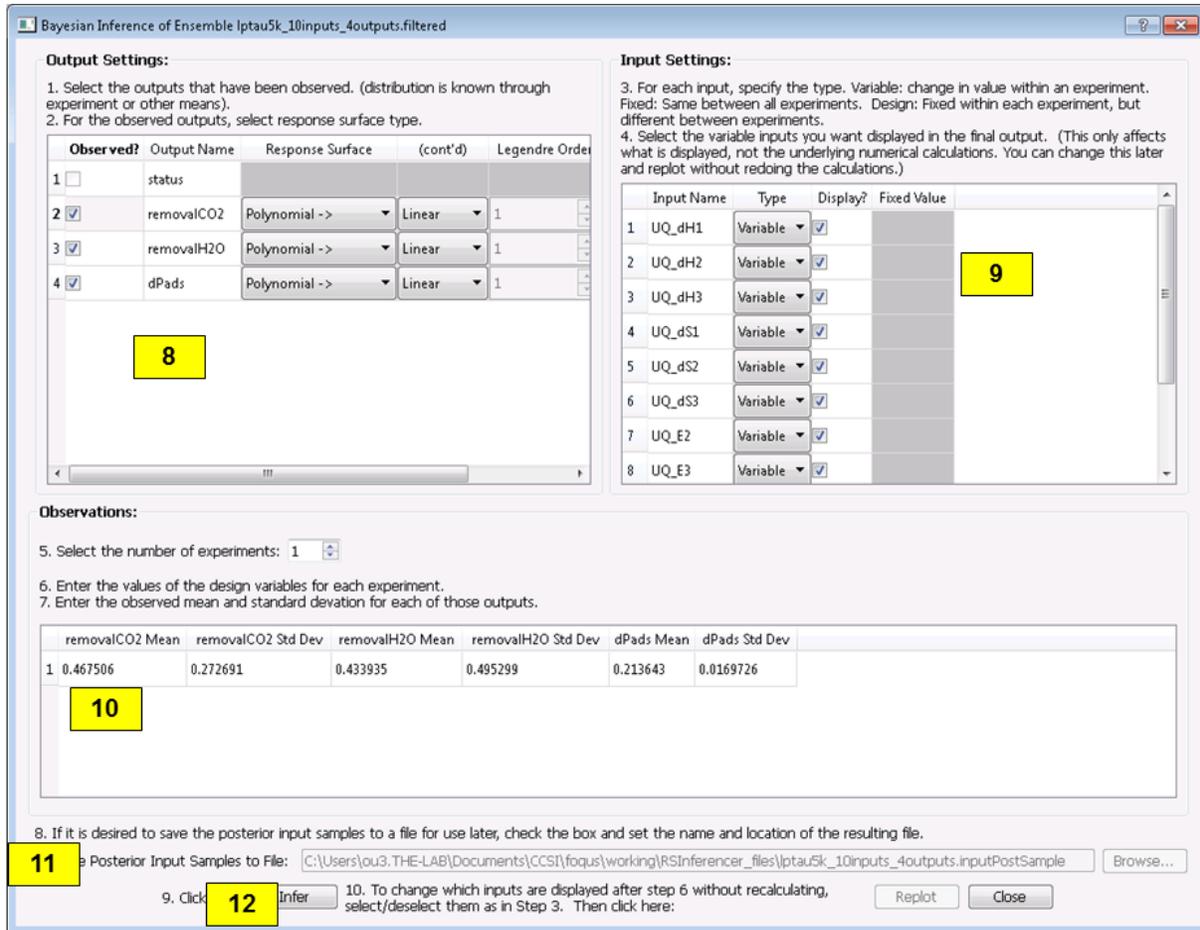
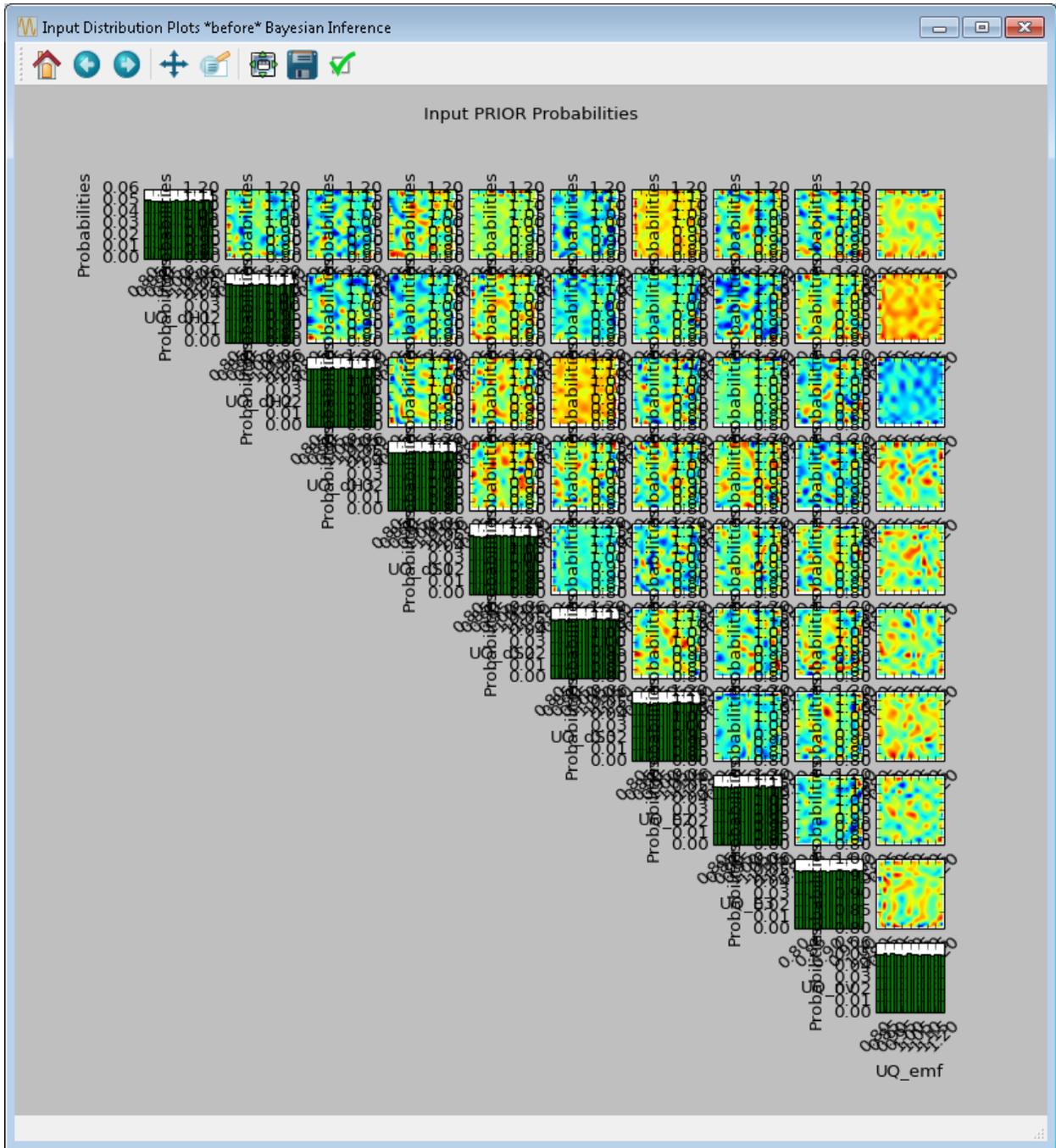
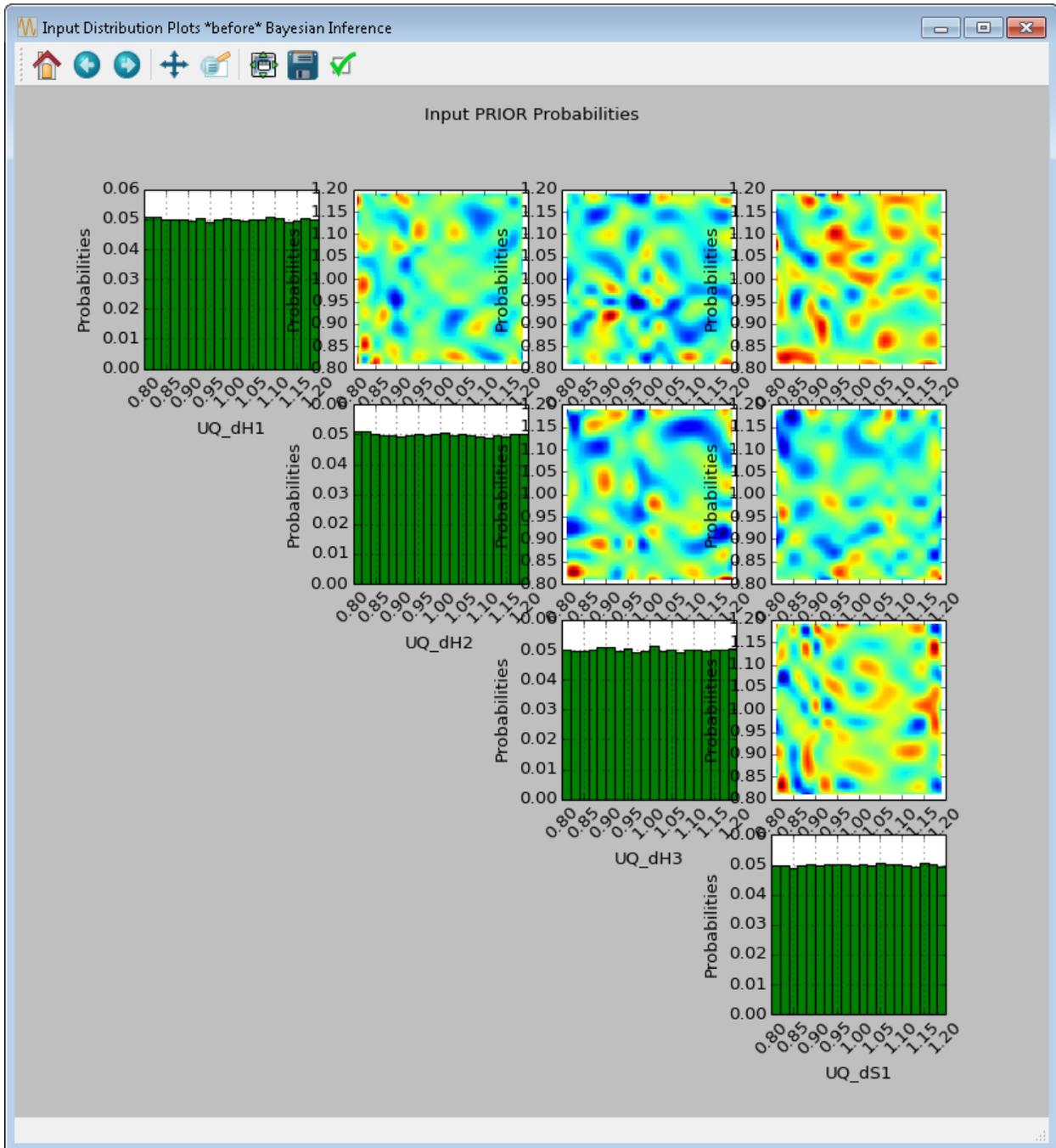
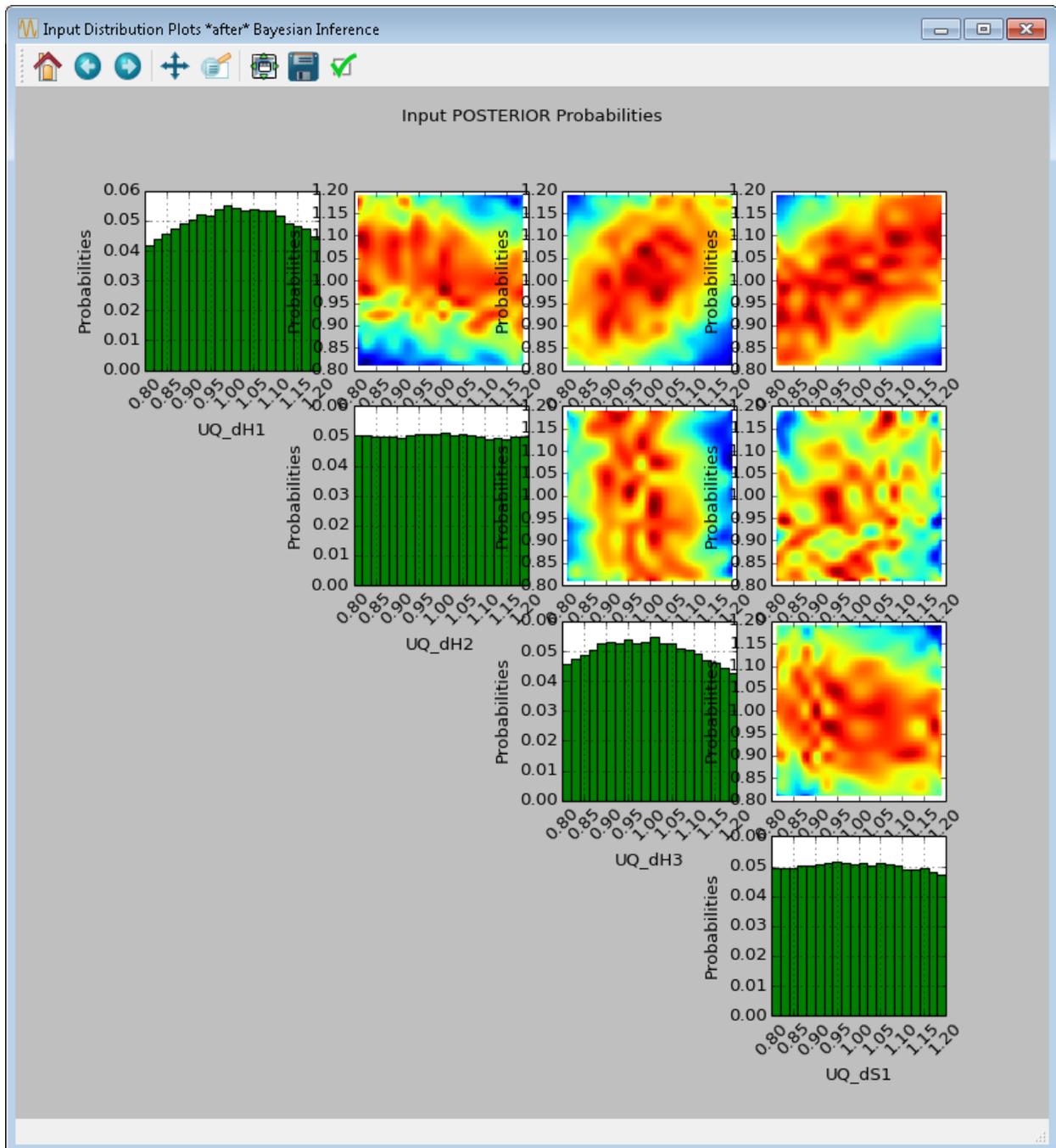


Fig. 55: Bayesian Inference Dialog for Standard Inference



To zoom in on any one of the subplots, left-click; to zoom out, right-click. To display a subset of these subplots, clear the checkbox for the inputs to be omitted (from the first column of the Input Prior Table) and click **Replot** (Figure [fig:uqt_infer_replot_results]).





[fig:uqt_infer_replot_results]

5.1.3 File

Formats

Most UQ capabilities within FOQUS rely on PSUADE. As such, different UQ components require input files in PSUADE formats. CSV (comma-separated values) files are also compatible. The specific requirements are explained in the UQ section *Tutorials* and section *Optimization Under Uncertainty (OUU)*.

PSUADE	Full	File	Format
--------	------	------	--------

The following is an example of the full PSUADE file format. Comments in red do not appear in the file and are only for instructional purposes.

full-format

This file format is accepted when:

- The user load an existing ensemble by clicking the **Load From File** button from the *Uncertainty Quantification Screen*.
- The user creates a new ensemble by clicking the **Add New** button from the *Uncertainty Quantification Screen* and selecting the **Load all samples from a single file** radio button in the user's selection of sample generation (*Simulation Ensemble Setup Dialog, Load Samples Option*).
- The user performs optimization under uncertainty from the main *Optimization Under Uncertainty Screen* and selects the **Load Model From File** radio button for the user's model; for this file, the user does not need to specify the first block (i.e., the PSUADE_IO block).

This file format is written when:

- The user saves an existing ensemble by clicking the **Save Selected** button from the *Uncertainty Quantification Screen*.

PSUADE	Sample	File	Format
--------	--------	------	--------

The following is an example of the sample file format. Comments in red do NOT appear in the file and are only for instructional purposes.

sample-format

This file format is accepted when:

- The user creates a new ensemble by clicking the **Add New** button from the *Uncertainty Quantification Screen* and selecting the **Load all samples from a single file** radio button in the user's selection of sample generation (*Simulation Ensemble Setup Dialog, Load Samples Option*).
- The user creates a new ensemble by clicking the **Add New** button from the *Uncertainty Quantification Screen* and selecting the **Choose sampling scheme** radio button in the user's selection of sample generation (*Simulation Ensemble Setup Dialog, Distributions Tab*); in the **Distributions** tab, if the user designates an input variable's PDF to be of type "Sample", the "Param 1" field will generate a **Select File** button that prompts for the sample file representing the input's PDF.
- Similar to above, when the user enters Expert Mode within the Analysis dialog; within Expert Mode (*Response Surface Based Mixed Epistemic-Aleatory Uncertainty Analysis*), the user can change the input distribution before performing response surface based analysis.
- The user performs optimization under uncertainty from the main *Optimization Under Uncertainty Screen*; if any of the variables are designated as random variables, the **UQ Setup** tab will be displayed and any prompt for loading existing sample (e.g., "Load existing sample for Z3" or "Load existing sample for Z4") will require this file format. (Currently, the **UQ Setup** tab is missing from the Figure because no variables have been designated as random).

This file format is written when:

- The user wants to save the results of inference by clicking **Save Posterior Input Samples to File** within Bayesian Inference (*Bayesian Inference Dialog*), which is accessible from the Analysis screen of UQ (*Analysis Dialog, Ensemble Data Analysis, Wizard Mode*).

Comma Separated Values (CSV) File Format

The following is an example of the CSV file format. Comments in red do not appear in the file and are only for instructional purposes. CSV files can be easily generated using Excel and exporting in the .csv format.

csv-format

Variable names are specified in the first line, with input names and then output names. Output names can be specified, even if there is no data available for them yet. Data is only required for inputs. In addition, the variable names line is not required in those places where a PSUADE sample file is acceptable.

This file format is accepted when:

- The user loads an existing ensemble by clicking the **Load from File** button from the *Uncertainty Quantification Screen*. Variable names are required.
- The user creates a new ensemble by clicking the **Add New** button from the *Uncertainty Quantification Screen* and selecting the **Load all samples from a single file** radio button in the user's selection of sample generation (*Simulation Ensemble Setup Dialog, Load Samples Option*).
- The user creates a new ensemble by clicking the **Add New** button from the *Uncertainty Quantification Screen* and selecting the **Choose sampling scheme** radio button in the user's selection of sample generation (*Simulation Ensemble Setup Dialog, Distributions Tab*); in the **Distributions** tab, if the user designates an input variable's PDF to be of type "Sample", the "Param 1" field will generate a **Select File** button that prompts for the sample file representing the input's PDF.
- Similar to above, when the user enters Expert Mode within the Analysis dialog; within Expert Mode (*Response Surface Based Mixed Epistemic-Aleatory Uncertainty Analysis*), the user can change the input distribution before performing response surface based analysis.
- The user performs optimization under uncertainty from the main *Optimization Under Uncertainty Screen*; if any of the variables are designated as random variables, the **UQ Setup** tab will be displayed and any prompt for loading existing sample (e.g., "Load existing sample for Z3" or "Load existing sample for Z4") will require this file format. (Currently, the **UQ Setup** tab is missing from the Figure because no variables have been designated as random).

OPTIMIZATION UNDER UNCERTAINTY (OUU)

6.1 Contents

6.1.1 Reference

The FOQUS OUU module supports several variants of optimization under uncertainty. This chapter first presents the mathematical formulations of these variants. Subsequently, details of the OUU graphical user interface will be discussed.

OUU

Variables

Suppose a simulation model is available for an OUU study. Let this simulation model be represented by the following function:

$$Y = F(Z_1, Z_2, Z_3, Z_4),$$

which is characterized by four types of variables:

1. Design/Decision/Optimization variables

- Notation: Z_1 with dimension n_1
- Definition: Design variables are continuous variables that may be bounded or unbounded. They are generally the set of optimization variables in a single-stage optimization or the set of outer optimization variables in the two-stage optimization.

2. Recourse/Operating variables

- Notation: Z_2 with dimension n_2
- Definition: Operating variables are optimization variables in the inner optimization for a given scenario (or realization) of the uncertain variables in a two-stage optimization.

3. Discrete uncertain variables

- Notation: Z_3 with dimension n_3
- Definition: Discrete uncertain variables are random variables that have an enumerable set of states (called scenarios) such that each state is associated with a finite probability and the sum of probabilities for all the scenarios is equal to 1.

4. Continuous uncertain variables

- Notation: Z_4 with dimension n_4
- Definition: Continuous uncertain variables are associated with a joint probability distribution function from which a sample can be drawn to compute the basic statistics.

OUU	Objective	Functions
------------	------------------	------------------

In the presence of uncertainties, OUU seeks to find the optimal solution in some statistical sense. In other words, an optimization goal may be to find the design settings that minimize some metric. Currently, OUU supports the following three metrics:

1. statistical mean of some selected output;
2. a linear combination of statistical mean and standard deviation of some selected output; and
3. the probability of exceeding the best value is smaller than some percentage at any point in the design space (this is analogous to conditional value at risk).

Note that these metrics are defined in the design variable space - that is, at each iteration of an OUU algorithm, the selected metric will be computed for the decision point under consideration. Since the calculation of these statistical metrics requires a (possibly large) sample, OUU can benefit from parallel computing capabilities of the AWS FOQUS Cloud.

Mathematical	Formulations
---------------------	---------------------

FOQUS supports two types of OUU methods: single-stage OUU and two-stage OUU. The main difference between single-stage and two-stage OUU is the presence of the recourse (or operating) variables. Strictly speaking, since recourse variables are generally hidden (they are only needed in the inner stage and their values are not used in the outer stage of two-stage OUU), the distinction between single-stage and two-stage OUU is not readily obvious. Nevertheless, for the sake of clarity, we will describe details of each formulation separately. The current OUU does not support linearly or nonlinearly-constrained optimization.

Single-Stage	Formulation
---------------------	--------------------

In this formulation, there is no recourse variable:

$$Y = F(Z_1, Z_3, Z_4)$$

and the optimization problem becomes:

$$\min_{Z_1} \Phi_{Z_3, Z_4} [F(Z_1, Z_3, Z_4)]$$

where $\Phi_{Z_3, Z_4} [F(Z_1, Z_3, Z_4)]$ is the statistical metric (one of the three options given above).

For example, if the objective function is the statistical mean, then the formulation becomes:

$$\min_{Z_1} \mathbf{E}_{Z_3, Z_4} [F(Z_1, Z_3, Z_4)] \approx \min_{Z_1} \sum_{j=1}^{n_3} \pi_j \left(\int F(Z_1, Z_3, Z_4) P(Z_4) dZ_4 \right)$$

where, again, n_3 is the number of scenarios for the discrete uncertain variables, π_j is the probability of the j -th scenario, and $P(Z_4)$ is the joint probability of the continuous uncertain variables.

Two-Stage

Formulation

In this formulation, all four types of variables are present. The objective function is given by:

$$\min_{Z_3, Z_4} \Phi_{Z_3, Z_4} \left[\min_{Z_2} F(Z_1, Z_2, Z_3, Z_4) \right].$$

If the objective function is the statistical mean, the formulation becomes:

$$\begin{aligned} & \min_{Z_1} \mathbf{E}_{Z_3, Z_4} \left[\min_{Z_2} F(Z_1, Z_2, Z_3, Z_4) \right] \\ \approx & \min_{Z_1} \sum_{j=1}^{n_3} \pi_j \left(\int \left[\min_{Z_2} F(Z_1, Z_2, Z_3, Z_4) \right] P(Z_4) dZ_4 \right) \end{aligned}$$

Let

$$G(Z_1, Z_3, Z_4) = \min_{Z_2} F(Z_1, Z_2, Z_3, Z_4).$$

Then the two-stage equation can be rewritten as:

$$\begin{aligned} & \min_{Z_1} \mathbf{E}_{Z_3, Z_4} [G(Z_1, Z_3, Z_4)] \\ \approx & \min_{Z_1} \sum_{j=1}^{n_3} \pi_j \left(\int G(Z_1, Z_3, Z_4) P(Z_4) dZ_4 \right) \end{aligned}$$

which is a single-stage OUU with respect to the G function. Therefore, G can be optimized separately before it is used to minimize Z_1 , thus leading to the two-stage formulation.

OUU

User

Interface

The OUU module enables the user to perform optimization under uncertainty studies on a flowsheet. From the OUU tab, the user can set up the different types of optimization parameters, select from the different OUU options, and run the optimization. This screen is shown in Figure [fig:ouu_screen].

1. **Model** provides two options for setting up the model: (1) select a node from the flowsheet that has already been instantiated; or (2) load the model from a file in the PSUADE full file format (with the `opt_driver` variable set to the simulation executable.)
2. **Variables** displays all variables defined in the model that can be used in this context. Each available variable can be set to either one of the 6 types:
 - “Fixed”: The parameter’s value is fixed throughout the optimization process.
 - “Opt: Primary Continuous (Z1)”: Continuous parameter for the outer optimization.
 - “Opt: Primary Discrete (Z1d)”: Discrete parameter for the outer optimization.
 - “Opt: Recourse (Z2)”: Recourse parameter for the inner optimization.
 - “UQ: Discrete (Z3)”: Discrete or categorical uncertain parameter that contributes to scenarios.
 - “UQ: Continuous (Z4)”: Continuous uncertain parameter with a given probability distribution.
3. **Optimization Setup** allows users to select the objective function for OUU. It also allows users to select the inner optimization solver. There are two options for the inner solver: (1) the simulation model provided by users is an optimizer itself, and (2) the simulation provided by users needs to be wrapped around by another optimizer in FOQUS.

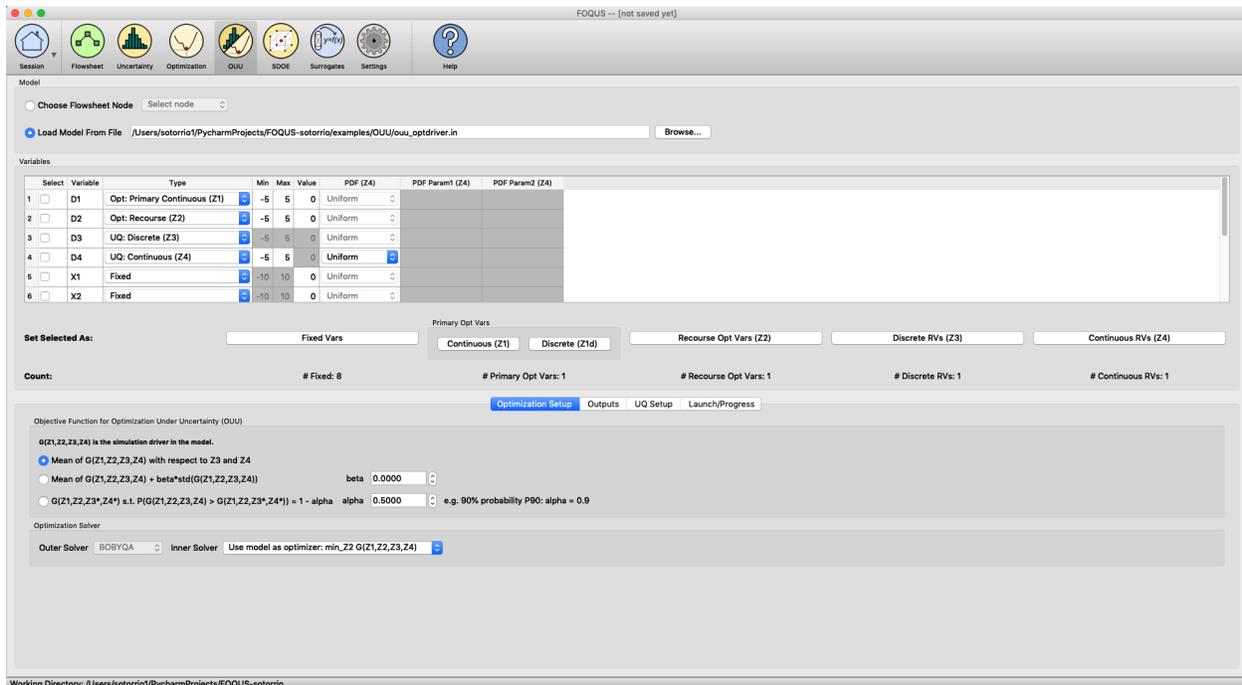


Fig. 1: Optimization Under Uncertainty Screen

- UQ Setup** allows users to set up the continuous uncertain parameters. There are two options: (1) FOQUS can generate a sample internally, or (2) a user-generated sample can be loaded into FOQUS. The sample size should be larger than the number of continuous uncertain parameters. Optionally, response surface can be turned on to enable the statistical moments to be computed more accurately even with small samples. Users can also select a smaller subset of the sample for building response surfaces and evaluate the response surfaces with the larger samples.
- Launch/Progress** has the 'Run OUU' button to launch OUU runs.

6.1.2 Tutorials

This section walks through a few examples of running OUU.

The files for these tutorials are located in: `examples/tutorial_files/OUU`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

Example 1: OUU with Discrete Uncertain Parameters Only

This example has only discrete uncertain parameters and the objective function is computed from the mean estimation with the scenarios from a sample file.

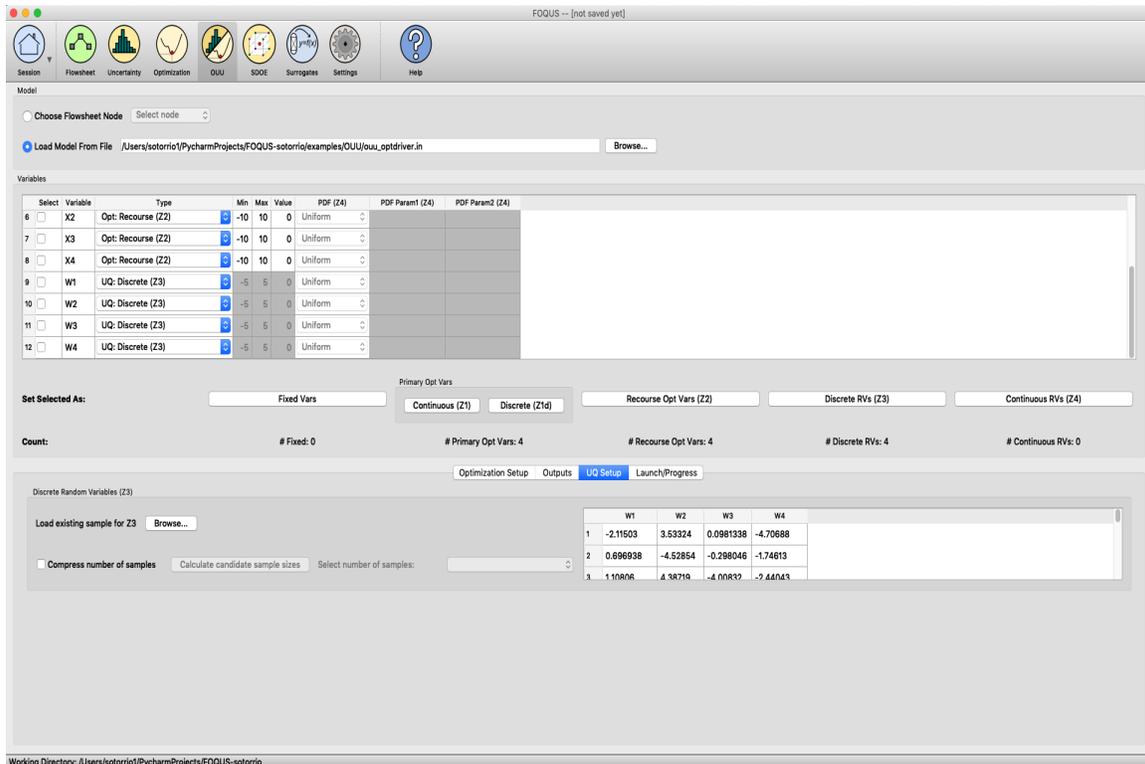


Fig. 2: OUU Example with Discrete Uncertain Parameters

1. Start FOQUS and click the 'OUU' icon.
2. Under 'Model', browse and load examples/OUU/ouu_optdriver.in.
3. Under 'Variables', set variable 1 – 4 as Z_1 , variable 5 – 8 as Z_2 , and variable 9 – 12 as Z_3 .
4. Under 'Optimization Setup', select the first objective function (default) and select 'use model as optimizer' as the 'Inner Solver'.
5. Under 'UQ Setup' and 'Discrete Random Variables', browse the examples/OUU/ directory and load the ex1_x3sample.smp sample file (see Figure [fig:ouu_ex1]).
6. Go to 'Launch/Progress' page, click 'Run OUU' and see OUU in action.

Example 2: OUU with Continuous Uncertain Parameters Only

This example has only continuous uncertain parameters and the objective function is computed from the mean estimation with a Latin hypercube sample of size 200 for Z_4 .

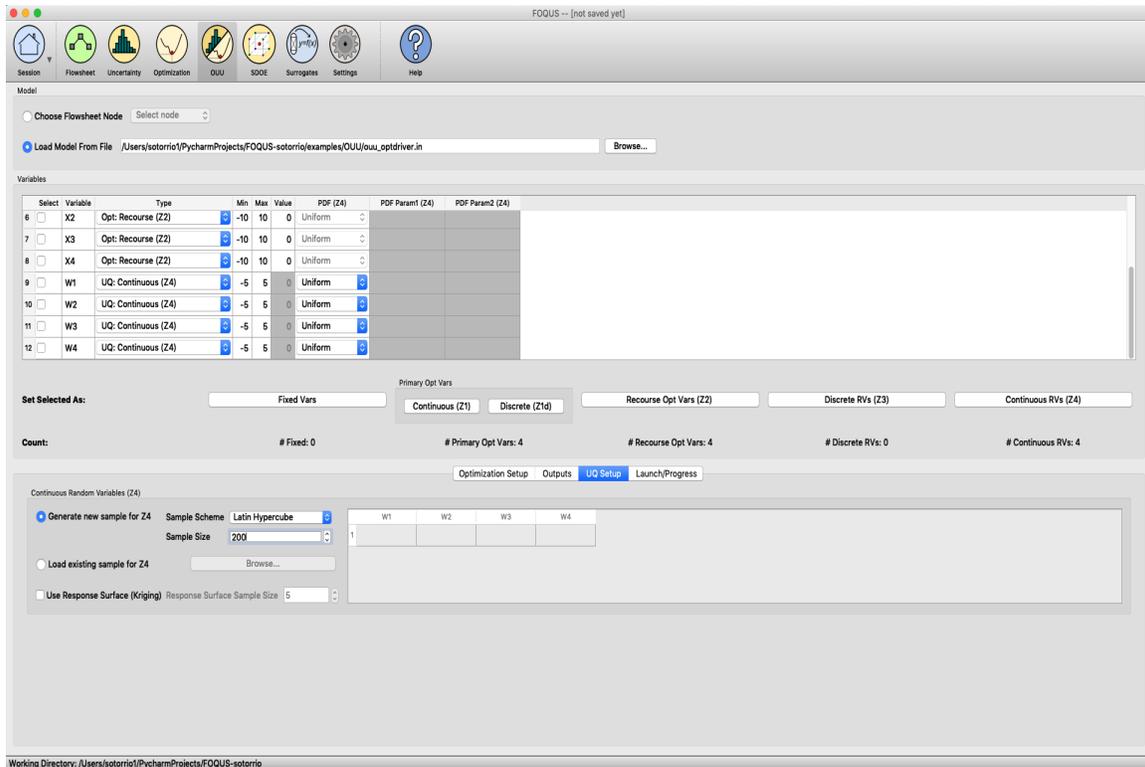


Fig. 3: OUU Example with Continuous Uncertain Parameters

1. Start FOQUS and click the 'OUU' icon.
2. Under 'Model', browse and load examples/OUU/ouu_optdriver.in.
3. Under 'Variables', set variable 1 – 4 as Z_1 , variable 5 – 8 as Z_2 , and variable 9 – 12 as Z_4 .
4. Under 'Optimization Setup', select the first objective function (default) and select 'use model as optimizer' as the 'Inner Solver'.
5. Under 'UQ Setup' and 'Continuous Random Variables', select 'Generate new sample for Z_4 ', set 'Sample Scheme' to 'Latin Hypercube' and set sample size to 200 (see Figure [fig:ouu_ex2]).
6. Go to 'Launch/Progress' page, click 'Run OUU' and see OUU in action.

Example 3: OUU with Continuous Uncertain Parameters and Response Surface

This example is similar to Example 2 except that response surfaces will be used on the Z_4 sample (that is, the Z_4 sample will be used to construct response surfaces and the means will be estimated from a large sample evaluated on the response surfaces).

1. Start FOQUS and click the ‘OUU’ icon.
2. Under ‘Model’, browse and load examples/OUU/ouu_optdriver.in.
3. Under ‘Variables’, set variable 1 – 4 as Z_1 , variable 5 – 8 as Z_2 , and variable 9 – 12 as Z_4 .
4. Under ‘Optimization Setup’, select the first objective function (default) and select ‘use model as optimizer’ as the ‘Inner Solver’.
5. Under ‘UQ Setup’ and ‘Continuous Random Variables’, select ‘Generate new sample for Z_4 ’, set ‘Sample Scheme’ to ‘Latin Hypercube’ and set sample size to 200.
6. Under ‘UQ Setup’ and ‘Continuous Random Variables’, check the ‘Use Response Surface’ box (see Figure [fig:ouu_ex2]).
7. Go to ‘Launch/Progress’ page, click ‘Run OUU’ and see OUU in action.

Example 4: OUU with Discrete and Continuous Uncertain Parameters

This example has both discrete and continuous parameters. The discrete scenarios will be loaded from a sample file. A Latin hypercube sample will be generated for the continuous variables.

1. Start FOQUS and click the ‘OUU’ icon.
2. Under ‘Model’, browse and load examples/OUU/ouu_optdriver.in.
3. Under ‘Variables’, set variable 1 – 4 as Z_1 , variable 5 – 8 as Z_2 , variable 9 as Z_3 , and variable 10 – 12 as Z_4 .
4. Under ‘Optimization Setup’, select the first objective function (default) and select ‘use model as optimizer’ as the ‘Inner Solver’.
5. Under ‘UQ Setup’ and ‘Discrete Random Variables’, browse the examples/OUU/ directory and load the ex456_x3sample.smp sample file.
6. Under ‘UQ Setup’ and ‘Continuous Random Variables’, select ‘Generate new sample for Z_4 ’, set ‘Sample Scheme’ to Latin hypercube and set ‘Sample Size’ to 100.
7. Go to ‘Launch/Progress’ page, click ‘Run OUU’ and see OUU in action.

Example 5: OUU with Mixed Uncertain Parameters and Response Surface

This example is similar to Example 4 except that response surfaces will be used to estimate the means for the continuous uncertain variables.

1. Start FOQUS and click the ‘OUU’ icon.
2. Under ‘Model’, browse and load examples/OUU/ouu_optdriver.in.
3. Under ‘Variables’, set variable 1 – 4 as Z_1 , variable 5 – 8 as Z_2 , variable 9 as Z_3 , and variable 10 – 12 as Z_4 .
4. Under ‘Optimization Setup’, select the first objective function (default) and select ‘use model as optimizer’ as the ‘Inner Solver’.
5. Under ‘UQ Setup’ and ‘Discrete Random Variables’, browse the examples/OUU/ directory and load the ex456_x3sample.smp sample file.

6. Under ‘UQ Setup’ and ‘Continuous Random Variables’, select ‘Generate new sample for Z_4 ’, set ‘Sample Scheme’ to Latin hypercube and set ‘Sample Size’ to 100.
7. Under ‘UQ Setup’ and ‘Continuous Random Variables’, check the ‘Use Response Surface’ box.
8. Go to ‘Launch/Progress’ page, click ‘Run OUU’ and see OUU in action.

Example 6: OUU with User-provided Samples and Response Surface

This example is similar to Example 4 except that a sample for Z_4 will be used (instead of the Latin hypercube sample generated internally).

1. Start FOQUS and click the ‘OUU’ icon.
2. Under ‘Model’, browse and load examples/OUU/ouu_optdriver.in.
3. Under ‘Variables’, set variable 1 – 4 as Z_1 , variable 5 – 8 as Z_2 , variable 9 as Z_3 , and variable 10 – 12 as Z_4 .
4. Under ‘Optimization Setup’, select the first objective function (default) and select ‘use model as optimizer’ as the ‘Inner Solver’.
5. Under ‘UQ Setup’ and ‘Discrete Random Variables’, browse the examples/OUU/ directory and load the ex456_x3sample.smp sample file.
6. Under ‘UQ Setup’ and ‘Continuous Random Variables’, check ‘Load existing sample for Z_4 ’ and load the Z_4 sample examples/OUU/ex6_x4sample.smp.
7. Go to ‘Launch/Progress’ page, click ‘Run OUU’ and see OUU in action.

Example 7: OUU with Large User-provided Samples and Response Surface

This example is similar to Example 5 except that a sample for Z_4 is provided (instead of generated internally).

1. Start FOQUS and click the ‘OUU’ icon.
2. Under ‘Model’, browse and load examples/OUU/ouu_optdriver.in.
3. Under ‘Variables’, set variable 1 – 4 as Z_1 , variable 5 – 8 as Z_2 , and variable 9 – 12 as Z_4 .
4. Under ‘Optimization Setup’, select the first objective function (default) and select ‘use model as optimizer’ as the ‘Inner Solver’.
5. Under ‘UQ Setup’ and ‘Continuous Random Variables’, check ‘Load existing sample for Z_4 ’ and load the Z_4 sample examples/OUU/ex7_x4sample.smp (10000 sample points).
6. Under ‘UQ Setup’ and ‘Continuous Random Variables’, check ‘Use Response Surface’ and set ‘Sample Size’ to 100.
7. Go to ‘Launch/Progress’ page, click ‘Run OUU’ and see OUU in action.

SURROGATE MODELING

7.1 Contents

7.1.1 Gradient Generation to Support Gradient-Enhanced Neural Networks

Neural networks are useful in instances where multivariate process data is available and the mathematical functions describing the variable relationships are unknown. Training deep neural networks is most efficient when samples of the variable derivatives, or gradients, are collected simultaneously with process data. However, gradient data is often unavailable unless the physics of the system are known and predetermined such as in fluid dynamics with outputs of known physical properties.

These gradients may be estimated numerically using solely the process data. The gradient generation tool described below requires a Comma-Separated Value (CSV) file containing process samples (rows), with inputs in the left columns and outputs in the rightmost columns. Multiple outputs are supported, as long as they are the rightmost columns, and the variable columns may have string (text) headings or data may start in row 1. The method produces a CSV file for each output variable containing gradients with respect to each input variable (columns), for each sample point (rows). After navigating to the FOQUS directory *examples/other_files/ML_AI_Plugin*, the code below sets up and calls the gradient generation method on the example dataset *MEA_carbon_capture_dataset_mimo.csv*:

```
# required imports
>>> import pandas as pd
>>> import numpy as np
>>> from generate_gradient_data import generate_gradients
>>>
>>> data = pd.read_csv(r"MEA_carbon_capture_dataset_mimo.csv") # get dataset
>>> data_array = np.array(data, ndmin=2) # convert to Numpy array
>>> n_x = 6 # we have 6 input variables, in the leftmost 6 columns

>>> gradients = generate_gradients(
>>>     xy_data=data_array,
>>>     n_x=n_x,
>>>     show_plots=False, # flag to plot regression results during gradient training
>>>     optimize_training=True, # will try many regression settings and pick the best
↪result
>>>     use_simple_diff=True # flag to use simple partials instead of chain rule formula;
↪defaults to False if not passed
>>> )
>>> print("Gradient generation complete.")

>>> for output in range(len(gradients)): # save each gradient array to a CSV file
>>>     pd.DataFrame(gradients[output]).to_csv("gradients_output" + str(output) + ".csv")
```

(continues on next page)

(continued from previous page)

```
>>> print("Gradients for output ", str(output), " written to gradients_output" +
↳str(output) + ".csv",)
```

Internally, the gradient generation methods automatically executes a series of actions on the dataset:

1. Import process data of size $(m, n_x + n_y)$, where m is the number of sample rows, n_x is the number of input columns and n_y is the number of output columns. Given n_x , the data is split into an input array X and an output array Y .

2. For each input x_i and each output y_j , estimate the gradient using a multivariate chain rule approximation. For example, the gradient of y_1 with respect to x_1 is calculated at each point as:

$$\frac{Dy_1}{Dx_1} = \frac{dy_1}{dx_1} \frac{dx_1}{dx_1} + \frac{dy_1}{dx_2} \frac{dx_2}{dx_1} + \frac{dy_1}{dx_3} \frac{dx_3}{dx_1} + \dots$$

where D/D represents the total derivative, d/d represents a partial derivative at each sample point. y_1, x_1, x_2, x_3 , and so on are vectors with values at each sample point m , and this formula produces the gradients of each output with respect to each input at each sample point by iterating through the dataset. The partial derivatives are calculated by simple finite difference. For example:

$$\frac{dy_1}{dx_1}(m_{1.5}) = \frac{y_1(m_2) - y_1(m_1)}{x_1(m_2) - x_1(m_1)}$$

where $m_{1.5}$ is the midpoint between sample points m_2 and m_1 . As a result, this scheme calculates gradients at the points between the sample points, not the actual sample points.

3. Train an MLP model on the calculated midpoint and midpoint-gradient values. After normalizing the data via linear scaling (see [Data Normalization For Neural Network Models](#)), the algorithm leverages a small neural network model to generate gradient data for the actual sample points. Passing the argument `optimize_training=True` will train models using the optimizers *Adam* or *RMSProp*, with activation functions *ReLU* or *Sigmoid* on hidden layers, using a *Linear* or *ReLU* activation function on the output layer, building 2 or 8 hidden layers with 6 or 12 neurons per hidden layer. The algorithm employs cross-validation to check the mean-squared-error (MSE) loss on each model and uses the model with the smallest error to predict the sample gradients.

4. Predict the gradients at each sample point from the regressed model. This produces n_y arrays with each having size (m, n_x) - the same size as the original input array X .

5. Concatenate the predicted gradients into a single array of size (m, n_x, n_y) . This is the single object returned by the gradient generation method.

7.1.2 Machine Learning & Artificial Intelligence Flowsheet Model Plugins

In addition to data-driven model generation, surrogates may be specified by importing external Python classes. FOQUS supports conversion of custom Pymodel scripts as well as neural network model files into flowsheet node surrogates. The FOQUS session script will automatically load model files from the corresponding working directory folders when the application is launched.

- **Plugin** – Selecting this model type in the Node Editor displays available Python model classes, which typically contain initialization and run methods to define the model expressions. To use this tool, users must develop a Pymodel script (see the examples in *FOQUS.foqus_lib.framework.pymodel*) as a guide) and place the file in the appropriate folder `user_plugins` in the working directory, as shown below. This model type is demonstrated in Section [Tutorial 5: Surrogates with the Flowsheet](#).
- **ML_AI** – Selecting this model type in the Node Editor displays available neural network models; this tool currently supports TensorFlow Keras model files saved in Hierarchical Data Format 5 (.h5), the standard Keras SavedModel format (folder containing .pb data files), or serialized to an architecture dictionary (.json) with separately saved model weights (.h5). Additionally, this tool supports PyTorch models saved in the standard format (.pt), and Scikit-learn and Surrogate Modeling Toolbox models serialized in the standard Python pickle format (.pkl). The examples folder contains demonstrative training and class scripts for models containing no custom

layer (see below for more information on adding custom layers), a custom layer with a preset normalization option and a custom layer with a custom normalization function, as well as models saved in all supported file formats. To use this tool, users must train and export a machine learning model and place the file in the appropriate folder *user_ml_ai_plugins* in the working directory, as shown below. Optionally, users may save Keras models with custom attributes to display on the node, such as variable labels and bounds. While training a Keras model with custom attributes is not required to use the plugin tool, users must provide the necessary class script if the Keras model does contain a custom object (see below for further information on creating custom objects). PyTorch, Scikit-learn, and Surrogate Modeling Toolbox models do not have this requirement and the class script does not need to exist in the plugins folder. This model type is used in the same manner as Pymodel Plugins, per the workflow in Section *Tutorial 5: Surrogates with the Flowsheet*.

Custom	Model	Attributes
---------------	--------------	-------------------

The high-level neural network library of Keras integrates with TensorFlow’s machine learning library to train complex models within Python’s user-friendly framework. Keras models may be largely split into two types: **Sequential** which build linearly connected model layers, and **Functional** which build multiple interconnected layers in a complex system. More information on TensorFlow Keras model building is described by (*Wu et al. 2020*). Users may follow the recommended workflow to install and use TensorFlow in a Python environment, as described in the TensorFlow documentation: <https://www.tensorflow.org/install>.

When importing TensorFlow Keras models, users should ensure their Python environment contains the same Keras package version used to train the model files. TensorFlow offers limited compatibility between versions. The example files include models trained with TensorFlow 2.3 and 2.7; users with TensorFlow 2.7 should use the 2.7 models.

The ML AI Plugin supports adding neural networks of either type to FOQUS nodes; if a custom object is needed, only the Functional API supports serializing custom attributes. If a model is saved with a custom input layer as shown below, FOQUS will automatically read and import the custom attributes into the Node Editor.

PyTorch offers an optimized tensor library for deep learning. While Keras connects dependent layers sequentially or simultaneously, PyTorch more explicitly uses prior layers as functional inputs for later layers in the neural network.

Similar to the built-in “custom object” registration feature in Keras, PyTorch allows the creation of custom layers while defining the “forward” advancement method that builds the network prior to training. Users may obtain a great deal of usage standards and best practices information as described in the PyTorch documentation:

<https://pytorch.org/docs/stable/index.html>.

Scikit-learn offers a machine learning library for predictive data analysis for a wide range of classification and regression problems, including neural networks. To train deep learning neural networks, the package utilizes a multi-layer regressor that optimizes squared-loss using LBFSG or stochastic gradient descent algorithms. These models offer less flexibility than TensorFlow Keras or Torch models, while providing a much simpler syntax for generating and leveraging neural networks. Users may find further information on the Scikit-learn package in the documentation: <https://scikit-learn.org/stable/index.html> and further information on deep learning capabilities as well: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor.

Surrogate Modeling Toolbox is an open-source Python package supporting a number of surrogate modeling methods, including gradient-enhanced neural network (GENN) models. GENN models train parameters by minimizing a modified Least Squares Estimator which accounts for partial derivative predictions, leading to better accuracy on fewer training points compared to non-gradient-enhanced models. Gradient methods are applicable when training use cases where system data is generally known, such as continuous physics-based problems like aerodynamics. If gradient data is not known, users may run a gradient generation tool provided within FOQUS and can consult the tool documentation here: *Gradient Generation to Support Gradient-Enhanced Neural Networks*. Users may find further information on GENN models within Surrogate Modeling Toolbox in the documentation:

https://smt.readthedocs.io/en/stable/_src_docs/surrogate_models/genn.html.

The examples files located in *FOQUS.examples.other_files.ML_AI_Plugin* show how users may train new models or re-save loaded models with a custom layer.

Currently, FOQUS supports the following custom attributes:

- *input_labels* – list of string objects containing labels for each input variable (default: x1, x2, x3, ...)
- *input_bounds* – list of tuple (pair) objects containing upper and lower bounds for each input variable (default: (0, 1E5))
- *output_labels* – list of string objects containing labels for each output variable (default: z1, z2, z3, ...)
- *output_bounds* – list of tuple (pair) objects containing upper and lower bounds for each output variable (default: (0, 1E5))
- *normalized* – Boolean flag for whether the user is passing a normalized neural network model; to use this flag, users must train their models with data normalized according to a specific scaling form and add all input and output bounds custom attributes. The section below details scaling options.
- *normalization_form* - string flag required when *normalization* is *True* indicating a scaling option for FOQUS to automatically scale flowsheet-level inputs to model inputs, and unscale model outputs to flowsheet-level outputs. The section below details scaling options.
- *normalization_function* - optional string argument that is required when a ‘Custom’ *normalization_form* is used. The section below details scaling options.

Data Normalization For Neural Network Models

In practice, large neural networks often tend towards overfitting and blurring of features; this is a particular issue with data varying between many orders of magnitude. Normalizing the input data using the input bounds simplifies internal calculations, reduces prediction error and minimizes the risk of feature loss. The simplest and most common approach is to linearly scale the data such that the lower bound becomes 0 and the upper bound becomes 1:

$$x_{norm} = \frac{x_{data} - x_{min}}{x_{max} - x_{min}}$$

$$z_{norm} = \frac{z_{data} - z_{min}}{z_{max} - z_{min}}$$

This scaling approach generalizes to a common formula:

$$x_{norm} = \frac{f(x_{data}) - f(x_{min})}{f(x_{max}) - f(x_{min})}$$

$$z_{norm} = \frac{f(z_{data}) - f(z_{min})}{f(z_{max}) - f(z_{min})}$$

FOQUS supports three scaling methods in this form: linear, base 10 logarithmic and base 10 exponential. Additionally, FOQUS supports two modified base 10 scaling options. Users may also write their own normalization functions and pass a string for FOQUS to parse internally via SymPy, a Python library for symbolic mathematics. It is the responsibility of the user to ensure string objects are valid SymPy expressions, and FOQUS will automatically scale and unscale using input and output variable bounds. For example, a custom version of ‘Log’ scaling following SymPy syntax (*not* Python or Latex syntax) would take the form below:

```
>>> ...
>>> self.normalized = True
>>> self.normalization_form = "Custom"
>>> self.normalization_function = "(log(datavalue, 10) - log(dataminimum, 10))/
↳(log(datamaximum, 10) - log(dataminimum, 10))"
>>> ...
```

The line below follows Python syntax and not SymPy syntax, and would yield the following error message:

```
>>> self.normalization_function = "(log10(datavalue) - log10(dataminimum))/
↳(log10(datamaximum) - log10(dataminimum))"
"ValueError: Model attribute normalization_function has value (log10(datavalue) -
↳log10(dataminimum))/(log10(datamaximum) - log10(dataminimum)) which is not a valid
↳SymPy expression. Please refer to the latest documentation for syntax guidelines and
↳standards: https://docs.sympy.org/latest/index.html"
```

Note that ‘value’, ‘minimum’ and ‘maximum’ are common reserved method names within Python and other modules, and such the labels ‘datavalue’, ‘dataminimum’ and ‘datamaximum’ are used instead. Detailed messages will appear in the console log for similar errors with specific causes. Custom expressions must use ‘value’, ‘minimum’ and ‘maximum’ to be recognized by FOQUS. More information on SymPy syntax, structure and standards may be found in their latest release documentation: <https://docs.sympy.org/latest/index.html>.

Note that users must implement desired data normalization during model training, and both of these steps occur externally to FOQUS. Users should ensure that data normalization results in an accurate neural network model without overfitting before loading into FOQUS. Available scaling options and required flags are summarized in the table below:

Table 1: Data Normalization Options

	Variable Bounds	<i>normal- ized</i>	<i>normaliza- tion_form</i>	Scaling Formula	<i>normaliza- tion_function</i>
None	Optional (not re- quired)	Must be <i>False</i> absent	or exclud- ing (not required)	$datascaled = datavalue$	Recommend excluding (not required)
Linear	Required	Must be <i>True</i>	‘Linear’	$datascaled = \frac{datavalue - dataminimum}{datamaximum - dataminimum}$	Recommend excluding (not required)
Log Base 10	Required	Must be <i>True</i>	‘Log’	$datascaled = \frac{\log_{10}(datavalue) - \log_{10}(dataminimum)}{\log_{10}(datamaximum) - \log_{10}(dataminimum)}$	Recommend excluding (not required)
Power	Required	Must be <i>True</i>	‘Power’	$datascaled = \frac{10^{datavalue} - 10^{dataminimum}}{10^{datamaximum} - 10^{dataminimum}}$	Recommend excluding (not required)
Log Base 10 Modi- fied	Required	Must be <i>True</i>	‘Log 2’	$datascaled = \log_{10} \left(9 * \frac{datavalue - dataminimum}{datamaximum - dataminimum} \right)$	Recommend excluding (not required)
Power Modified	Required	Must be <i>True</i>	‘Power 2’	$datascaled = \frac{1}{9} * \frac{10^{datavalue} - 10^{dataminimum}}{10^{datamaximum} - 10^{dataminimum}}$	Recommend excluding (not required)
Custom	Required	Must be <i>True</i>	‘Custom’	$datascaled = f(datavalue, dataminimum, datamaximum)$	Must be a String with proper SymPy syntax

Usage

Example

The following code snippet demonstrates the Python syntax to train and save a Keras model with custom attributes; users should refer to the examples folder for usage of non-Keras neural network trainers. The use of Dropout features in training is not required, but decreases the risk of overfitting by minimizing the number of parameters in large models. Similarly, normalizing data often results in more accurate models since features are less likely to be blurred during fitting. Users may then enter unscaled input values and return unscaled output values in the Node Editor. Note that the custom object class script containing the class and the NN model file itself must all share the same name to import the custom attributes into a FOQUS node. If certain custom attributes are not used, it is best if users do not include them in the custom class definition; for example, the attribute *normalization_function* is not required in this example and therefore is excluded in the code below. See *FOQUS.examples.other_files.ML_AI_Plugin.mea_column_model_training_customnormform.py* for an example implementing a custom normalization function.

Users must ensure the proper script name is used in the following places, replacing *example_model* with the desired model name:

- Custom class signature, *class example_model(tf.keras.layers.Layer):*
- Creating a callable object, *super(example_model, self).__init__()*
- Defining the class CONFIG, *config = super(example_model, self).get_config()*
- Creating the model, *layers = example_model(*
- Saving the model, *model.save('example_model.h5')*
- The file names of the .h5 model file and custom class script.

See the example files in *FOQUS.examples.other_files.ML_AI_Plugin* for complete syntax and usage. The folder contains a second model with no custom layer to demonstrate the plugin defaults. The default output values are not calculated, so the node should be run to obtain the correct output values for the entered inputs.

To run the models, copy the appropriate model files or folders ('h5_model.h5', 'saved_model/', 'json_model.json', 'json_model_weights.h5') and any custom layer scripts ('model_name.py') into the working directory folder 'user_ml_ai_models'. As mentioned earlier, PyTorch, Scikit-learn and Surrogate Modeling Toolbox models only require the model file ('pt_model.pt', 'skl_model.pkl' or 'smt_model.pkl'). For example, the model name below is 'mea_column_model' and is saved in H5 format, and the files

FOQUS.examples.other_files.ML_AI_Plugin.TensorFlow_2-10_Models.mea_column_model.h5 and
FOQUS.examples.other_files.ML_AI_Plugin.mea_column_model.py should be copied to

FOQUS-wd.user_ml_ai_models. For users with older versions of TensorFlow who wish to test the example models, some model files are provided in versions 2.3 and 2.7 as well as 2.10. Generally, TensorFlow is backwards compatible for models two versions back (i.e., loading models trained in version 2.3 using version 2.5, or loading models trained in version 2.8 using version 2.10 is supported).

To distinguish between H5 models and json models with H5 weight files, FOQUS requires the convention ('model1.h5', 'model1.py') and ('model2.json', 'model2_weights.h5', 'model2.py') when naming model files. Users should note that defining network layers and training the network is independent of saved file format, and only the code after *model.summary()* in the script below will change. See the 'training_customnormform' example scripts for specific syntax to save models as each Keras file format and non-Keras file type.

```
# Required imports
>>> import numpy as np
>>> import pandas as pd
>>> import tensorflow as tf

# Example follows the sequence below:
# 1) Main Code at end of file to import data and create model
```

(continues on next page)

(continued from previous page)

```

# 2) Call create_model() to define inputs and outputs
# 3) Call custom layer object to define network structure, which uses
#     call() to define layer connections and get_config to attach
#     attributes to the custom layer
# 4) Back to create_model() to compile and train model
# 5) Back to code at end of file to save the model

# custom class to define Keras NN layers and serialize (register) objects
>>> @tf.keras.utils.register_keras_serializable() # first non-imports line to include
↳ in working directory example_model.py
>>> class mea_column_model(tf.keras.layers.Layer):
    # give training parameters default values, and set attribute defaults to None
>>>     def __init__(self, n_hidden=1, n_neurons=12,
>>>                 layer_act='relu', out_act='sigmoid',
>>>                 input_labels=None, output_labels=None,
>>>                 input_bounds=None, output_bounds=None,
>>>                 normalized=False, normalization_form='Linear',
>>>                 **kwargs):
>>>
>>>         super(mea_column_model, self).__init__() # create callable object

    # add attributes from training settings
>>>         self.n_hidden = n_hidden
>>>         self.n_neurons = n_neurons
>>>         self.layer_act = layer_act
>>>         self.out_act = out_act

    # add attributes from model data
>>>         self.input_labels = input_labels
>>>         self.output_labels = output_labels
>>>         self.input_bounds = input_bounds
>>>         self.output_bounds = output_bounds
>>>         self.normalized = True # FOQUS will read this and adjust accordingly
>>>         self.normalization_form = 'Linear' # tells FOQUS which scaling form to use

    # create lists to contain new layer objects
>>>         self.dense_layers = [] # hidden or output layers
>>>         self.dropout = [] # for large number of neurons, certain neurons
                                # can be randomly dropped out to reduce overfitting

>>>         for layer in range(self.n_hidden):
>>>             self.dense_layers.append(
>>>                 tf.keras.layers.Dense(
>>>                     self.n_neurons, activation=self.layer_act))

>>>         self.dense_layers_out = tf.keras.layers.Dense(
>>>             2, activation=self.out_act)

    # define network layer connections
>>>     def call(self, inputs):
>>>
>>>         x = inputs # single input layer, input defined in create_model()

```

(continues on next page)

(continued from previous page)

```

>>>     for layer in self.dense_layers: # hidden layers
>>>         x = layer(x) # h1 = f(input), h2 = f(h1), ... using act func
>>>     for layer in self.dropout: # no dropout layers used in this example
>>>         x = layer(x)
>>>     x = self.dense_layers_out(x) # single output layer, output = f(h_last)
>>>
>>>     return x
>>>
>>>     # attach attributes to class CONFIG
>>>     def get_config(self):
>>>         config = super(mea_column_model, self).get_config()
>>>         config.update({ # add any custom attributes here
>>>             'n_hidden': self.n_hidden,
>>>             'n_neurons': self.n_neurons,
>>>             'layer_act': self.layer_act,
>>>             'out_act': self.out_act,
>>>             'input_labels': self.input_labels,
>>>             'output_labels': self.output_labels,
>>>             'input_bounds': self.input_bounds,
>>>             'output_bounds': self.output_bounds,
>>>             'normalized': self.normalized,
>>>             'normalization_form': self.normalization_form,
>>>         })
>>>     return config
>>>
>>>     # method to create model
>>>     def create_model(data):
>>>
>>>         inputs = tf.keras.Input(shape=(np.shape(data)[1],)) # create input layer
>>>
>>>         layers = mea_column_model( # define the rest of network using our custom class
>>>             input_labels=xlabels,
>>>             output_labels=zlabels,
>>>             input_bounds=xdata_bounds,
>>>             output_bounds=zdata_bounds,
>>>             normalized=True,
>>>             normalization_form='Linear',
>>>         )
>>>
>>>         outputs = layers(inputs) # use network as function outputs = f(inputs)
>>>
>>>         model = tf.keras.Model(inputs=inputs, outputs=outputs) # create model
>>>
>>>         model.compile(loss='mse', optimizer='RMSprop', metrics=['mae', 'mse'])
>>>
>>>         model.fit(xdata, zdata, epochs=500, verbose=0) # train model
>>>
>>>         return model
>>>
>>>     # Main code

```

(continues on next page)

(continued from previous page)

```

# import data
>>> data = pd.read_csv(r'MEA_carbon_capture_dataset_mimo.csv')

>>> xdata = data.iloc[:, :6] # here there are 6 input variables/columns
>>> zdata = data.iloc[:, 6:] # the rest are output variables/columns
>>> xlabel = xdata.columns.tolist() # set labels as a list (default) from pandas
>>> ylabel = zdata.columns.tolist() # is a set of IndexedDataSeries objects
>>> xdata_bounds = {i: (xdata[i].min(), xdata[i].max()) for i in xdata} # x bounds
>>> zdata_bounds = {j: (zdata[j].min(), zdata[j].max()) for j in zdata} # z bounds

# normalize data - linear scaling is performed manually before training
>>> xmax, xmin = xdata.max(axis=0), xdata.min(axis=0)
>>> zmax, zmin = zdata.max(axis=0), zdata.min(axis=0)
>>> xdata, zdata = np.array(xdata), np.array(zdata)
>>> for i in range(len(xdata)):
>>>     for j in range(len(xlabel)):
>>>         xdata[i, j] = (xdata[i, j] - xmin[j])/(xmax[j] - xmin[j])
>>>     for j in range(len(ylabel)):
>>>         zdata[i, j] = (zdata[i, j] - zmin[j])/(zmax[j] - zmin[j])

>>> model_data = np.concatenate((xdata,zdata), axis=1) # Keras requires a Numpy array
↳as input

# define x and z data, not used but will add to variable dictionary
>>> xdata = model_data[:, :-2]
>>> zdata = model_data[:, -2:]

# create model
>>> model = create_model(xdata)
>>> model.summary()

# save model
>>> model.save('mea_column_model.h5')

```

After training and saving the model, the files should be placed in the working directory folder as shown below; if FOQUS cannot find the custom class due to a missing or misnamed script, the node will not load the attributes. As noted above, only the custom class lines should be included in the script:

Upon launching FOQUS, the console should include the lines boxed in red below to show the model files have been successfully loaded:

The model will then appear in the Node Editor menu:

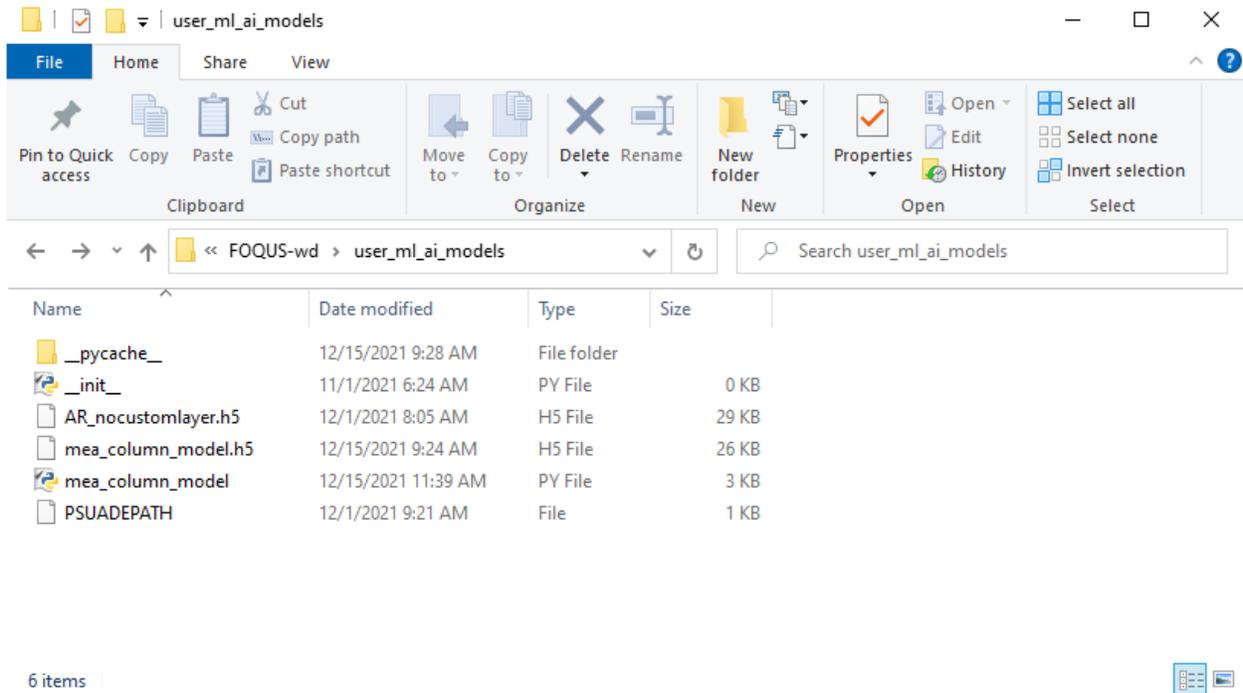
7.1.3 Surrogate

Models

Overview

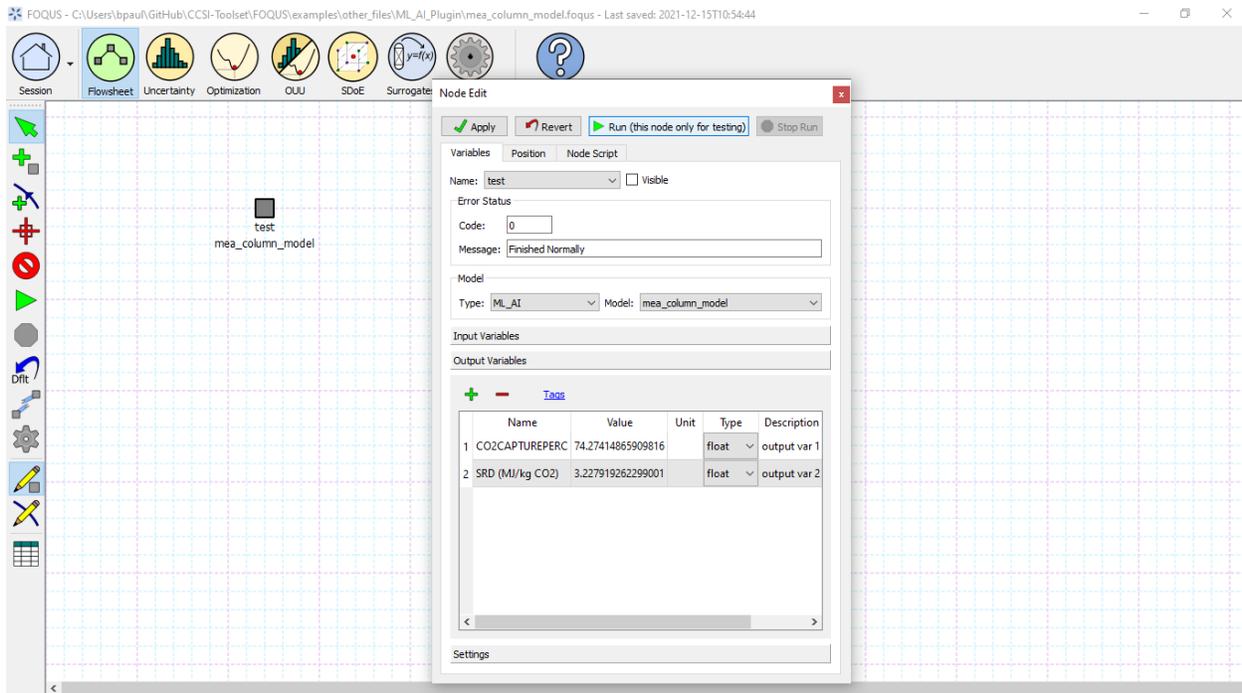
Large-scale computational models are crucial tools to analyze complex systems. When coupled with uncertainty quantification and optimization methods, the resulting computational expense becomes intractable. In order to face the computational burden, surface approximation methods, black box models, or surrogate models are commonly used. FOQUS provides a selection of surrogate modeling tools all using a similar work-flow. This section provides an overview of the surrogate modeling features and capabilities. The details of each tool are provided in the tutorial sections.

The following surrogate modeling tools are currently available:



```

Anaconda Prompt (Miniconda3) - foqus
(ccsi-foqus-dev) C:\Users\bpaul>foqus
2021-12-15 11:53:39.771760: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2021-12-15 11:53:39.771832: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
2021-12-15 11:53:41.971 - DEBUG - foqus.foqus_lib.foqus - Working directory set to C:\Users\bpaul\GitHub\CCSI-Toolset\FOQUS-wd
2021-12-15 11:53:41.983 - DEBUG - foqus.foqus_lib.framework.session.session - Initializing session, log file: C:\Users\bpaul\GitHub\CCSI-Toolset\FOQUS-wd\logs\foqus.log, Position: 612526
2021-12-15 11:53:41.984 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Loading Plugin: c:\users\bpaul\github\ccsi-toolset\foqus\foqus_lib\framework\surrogate\ACOSSO.py
2021-12-15 11:53:41.987 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Loading Plugin: c:\users\bpaul\github\ccsi-toolset\foqus\foqus_lib\framework\surrogate\ALAMO.py
2021-12-15 11:53:41.989 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Loading Plugin: c:\users\bpaul\github\ccsi-toolset\foqus\foqus_lib\framework\surrogate\BSS-ANOVA.py
2021-12-15 11:53:41.992 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Loading Plugin: c:\users\bpaul\github\ccsi-toolset\foqus\foqus_lib\framework\optimizer\BFGS.py
2021-12-15 11:53:42.091 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Loading Plugin: c:\users\bpaul\github\ccsi-toolset\foqus\foqus_lib\framework\optimizer\NLOpt.py
2021-12-15 11:53:42.093 - INFO - foqus.NLOpt - Failed to import the nlopt package
2021-12-15 11:53:42.094 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Loading Plugin: c:\users\bpaul\github\ccsi-toolset\foqus\foqus_lib\framework\optimizer\OptCMA.py
2021-12-15 11:53:42.269 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Loading Plugin: c:\users\bpaul\github\ccsi-toolset\foqus\foqus_lib\framework\optimizer\PSUADE.py
2021-12-15 11:53:42.271 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Loading Plugin: c:\users\bpaul\github\ccsi-toolset\foqus\foqus_lib\framework\optimizer\Sample.py
2021-12-15 11:53:42.273 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Loading Plugin: c:\users\bpaul\github\ccsi-toolset\foqus\foqus_lib\framework\optimizer\LSQP.py
2021-12-15 11:53:42.274 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Loading Plugin: c:\users\bpaul\github\ccsi-toolset\foqus\foqus_lib\framework\optimizer\SM_Optimizer.py
2021-12-15 11:53:42.277 - INFO - foqus.SM_Optimizer - Failed to import the required packages for SM Optimizer solver
2021-12-15 11:53:42.278 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Loading Plugin: c:\users\bpaul\github\ccsi-toolset\foqus\foqus_lib\framework\optimizer\Snobfit.py
2021-12-15 11:53:42.280 - INFO - foqus.Snobfit - Failed to import SQSnobFit and SQCommon packages used to access the snobfit solver
2021-12-15 11:53:42.282 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Removing plugin, due to missing dependency: NLOpt
2021-12-15 11:53:42.282 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Removing plugin, due to missing dependency: SM_Optimizer
2021-12-15 11:53:42.282 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Removing plugin, due to missing dependency: Snobfit
2021-12-15 11:53:42.283 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Loading Plugin: C:\Users\bpaul\GitHub\CCSI-Toolset\FOQUS-wd\user_plugins\alamo_surrogate.py
2021-12-15 11:53:42.285 - INFO - foqus.foqus_lib.framework.plugins.pluginSearch - Loading Plugin: C:\Users\bpaul\GitHub\CCSI-Toolset\FOQUS-wd\user_plugins\alamo_tutorial.py
2021-12-15 11:53:42.288 - INFO - foqus.foqus_lib.framework.ml_ai_models.mlaiSearch - Loading ML_AI Model: C:\Users\bpaul\GitHub\CCSI-Toolset\FOQUS-wd\user_ml_ai_models\AR_nocustomlayer.h5
2021-12-15 11:53:42.288 - INFO - foqus.foqus_lib.framework.ml_ai_models.mlaiSearch - Loading ML_AI Model: C:\Users\bpaul\GitHub\CCSI-Toolset\FOQUS-wd\user_ml_ai_models\mea_column_model.h5
2021-12-15 11:53:42.291 - DEBUG - foqus.foqus_lib.framework.sim.turbineConfiguration - turbine configuration application url="http://localhost:8000/TurbineLite/application/"
    
```



- ACOSSO – Adaptive Component Selection and Shrinkage Operator is a regularization method for simultaneous model fitting and variable selection based in nonparametric regression methods. ACOSSO is suitable for approximating models with many inputs and no sharp changes.
- ALAMO – Automated Learning of Algebraic Models for Optimization generates algebraic models from data sets. These surrogate models are ideal for equation oriented optimization problems (which are easily differentiable), such as super structure optimization.
- BSS-ANOVA – Bayesian Smoothing Spline Analysis of Variance is a method similar to ACOSSO.
- iREVEAL – Surrogate models for CFD simulations using Kriging or Neural Networks. It contains special features specifically designed for working with CFDs.
- Keras Neural Networks (keras_nn) – Surrogate models leveraging neural networks in TensorFlow Keras, a high-level machine learning library that supports training sequential or interconnected graph-based models.
- PyTorch Neural Networks (pytorch_nn) – Surrogate models leveraging neural networks in PyTorch, an optimized tensor library for deep learning that supports training sequential or interconnected graph-based models.
- Scikit-learn Neural Networks (scikit_nn) – Surrogate models leveraging neural networks in Scikit-learn, a predictive data analysis and surrogate modeling library that supports multi-layer regression and classification.

Data

Selection

The **Data** tab allows the selection of training data to be used to generate a surrogate model (*Surrogate Data Form*). If the session is associated with a flowsheet data (results from a single flowsheet run, optimization runs, or UQ samples), then the flowsheet data is available to be the training data and the table will be populated accordingly.

Fig. 1: Surrogate Data Form

1. **Run** the surrogate modeling method.

2. **Stop** the surrogate modeling method.
3. **Surrogate modeling tool** enables the user to select the desired surrogate modeling tool from the **Tool** drop-down list.
4. **Description** of the selected surrogate method.
5. **Add Samples** enables the user to generate new training data using a model specified in the flowsheet or an emulator (i.e., a basic response surface provided as part of the UQ module).
6. **Flowsheet Results** are summarized below.
7. The data table has a **Menu** drop-down list that contains display, import/export, and edit commands.
8. Select a data filter from the **Current Filter** drop-down for current data display.
9. Add or edit new data filters from **Edit Filters**. This dialog is shown in Figure *Sort1 Data Filter Results*.
10. The **Display** table displays the results of flowsheet evaluations stored in the FOQUS session file. The columns are:
 - **SetName** is a name assigned to samples. This is typically equivalent to one UQ sample run or one optimization run.
 - **ResultName** is a string representing a result name.
 - **Error** is the simulation result status; 0 indicates success, other numbers represent an error. A column for each node displays the error status of each node.
 - **Time** displays the time when the result was stored.
 - **Elapsed Time** describes how long a result took to calculate.
 - **Tags** enables a list of string labels to be applied to results. This could be used to mark results to be used for a particular purpose such as model validation.
 - The remaining columns display the input and output variables.

Filters can be used to select data. See Section *Tutorial 4: Flowsheet Result Data* for more information on creating filters to the results. The “All” and “None” filters are available by default. These can be used, for example, to assign all the data as a training set, or to split the data into a separate training set and a test set.

Variables

The **Variables** section is illustrated in Figure *Surrogate Variable Selection*. This section allows selection of input and output variables used in a surrogate model. Some surrogate methods such as ALAMO may generate and run additional samples while building surrogates. The **Min/Max** columns provide bounds on the variables. Selecting the checkbox next to the variable **Name** indicates that it should be included in the surrogate generation. Failure to select a checkbox for any variables will result in error during surrogate generation.

Fig. 2: Surrogate Variable Selection

Note : The input and output variables that are displayed in this section are the ones present in the FOQUS flowsheet nodes. If the user would like to include additional output variables (calculated based on the original node variables) in the surrogate model, it is recommended to create them in the output section of the flowsheet node itself, from the very beginning. With this approach, the calculated variables will be a part of the surrogate variable selection, and their relation with the other node variables can be defined in the nodedescript.

Method**Settings**

The **Method Settings** table is illustrated in Figure *Surrogate Settings*. The settings available in this table depend on the surrogate tool. A description of each setting is provided in the third column of the table.

Fig. 3: Surrogate Settings

Execution

Clicking **Run** starts the surrogate model building process. The execution monitor displays after **Run** is clicked (see Figure *Surrogate Status Monitor*). The execution monitor displays the status of the surrogate build. The messages displayed depends on the surrogate tool.

Fig. 4: Surrogate Status Monitor

After a successful execution and model building, the results are displayed. Note that in this case, the surrogate modeling tool ends with an error, the errors are displayed in this window. After surrogate generation completes, one or two Python files will be generated depending on the tool. Each tool generates a file that encodes the surrogate model as a general Python script that can be used to evaluate output values for UQ analyses within the UQ module.

The other file, if available, is a FOQUS flowsheet plugin model that allows the surrogate to be run in a FOQUS flowsheet. The next version of FOQUS will generate a FOQUS flowsheet plugin model (i.e., the second file) for all surrogate tools.

7.1.4 Tutorial**Tutorial****1:****ALAMO**

This tutorial focuses on the use of the ALAMO tool for building algebraic surrogate models. ALAMO builds simplified algebraic models, which are particularly well suited for rigorous equation oriented optimization. To keep the execution of this tutorial fast, a toy problem is used. In this case study the flowsheet calculations and sample generation are done within FOQUS, alternatively, the user can provide a simulation model such as: Excel, Aspen plus, Aspen custom modeler, etc.

Note: Before starting this tutorial the ALAMO product must be downloaded from the products page on the CCSI website. The path for the ALAMO executable file must be set in FOQUS settings (see Section *Settings*).

The FOQUS file (**Surrogate_Tutorial_1.foqus**), where Steps 1 to 42 of this tutorial have been completed is located in: `examples/tutorial_files/Surrogates`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

Flowsheet

Setup

1. Open FOQUS.
2. Name the session “Surrogate_Tutorial_1” (Figure *Session Set Up*).

Fig. 5: Session Set Up

3. Navigate to the Flowsheet Editor (Figure *Flowsheet Setup*).
4. Add a Flowsheet Node named “eq.”
5. Display the Node Editor by clicking the **Node Editor** toggle button.

Fig. 6: Flowsheet Setup

The **Node Editor** displays (Figure *Node Variables*). The first step to setting up the node for this problem is to add input and output variables to the node.

6. If the input variables table is not displayed as shown in Figure *Node Variables*, click the **Variables** tab and then click the **Input Variables** toolbox section.
7. Add the variables “x1” and “x2” by clicking the **Add** icon (+) above the input table.
8. Edit the **Min/Max** value for both variables to be “-10.0” and “10.0.”
9. Add two output variables “z1” and “z2.”

Fig. 7: Node Variables

To keep the execution time short, the node will not be assigned to a simulation model and calculations are performed directly in FOQUS.

10. Click on the **Node Script** tab in the Node Editor to enter the test equation (this step replaces the use of a simulator).
11. Enter the following equations (Figure *Node Script*):

```
f["z1"] = x["x1"] + x["x2"]
f["z2"] = x["x1"]**2 + x["x2"]**2
```

The node script calculations are written in Python. The dictionary “f” stores output values while the dictionary “x” stores input values.

12. Test the model by running the flowsheet with the value “2” for “x1” and “x2.” After running, the output variables should have the values “4.0” for “z1” and “8.0” for “z2.”

Creating

Initial

Samples

There are two ways to start an ALAMO run: (1) generate a set of initial data, (2) use ALAMO’s adaptive sampling with no initial data and let ALAMO generate its own samples. Adaptive sampling can be used with initial data to generate more points if needed. In this case, initial data is provided and adaptive sampling is used.

13. Select the UQ tool by clicking on the **Uncertainty** button on the Home window (Figure *Add a New Sample Ensemble*).
14. Click the **Add New** button.

Fig. 8: Node Script

15. The **Add New Ensemble - Model Selection** dialog will appear. Click **OK** to set up the sampling scheme.

Fig. 9: Add a New Sample Ensemble

16. The sample ensemble setup dialog displays (Figure *Sample Distributions*). Select **Choose sampling scheme**.
17. Click the **All Variable** button.
18. Select the **Sampling scheme** tab.

Fig. 10: Sample Distributions

19. The **Sampling scheme** dialog should display (Figure *Sample Methods*). Select “Latin Hypercube” from the list.
20. Set the **# of samples** to “1000.”
21. Click **Generate Samples**.
22. Click **Done**.
23. Once the samples have been generated a new sample ensemble displays in the UQ tool window (Figure *Run Samples*). Click **Launch** to run and generate the samples.

Data

Selection

Initial and validation data can be specified by creating filters that specify subsets of flowsheet data. In this tutorial only initial data will be used. A filter must be created to separate the results of the single test run from the UQ samples.

24. Click on the **Surrogates** button from the Home window. The surrogate tool displays *Surrogate Data*.
25. Select “ALAMO” from the **Tool** drop-down list.
26. Click **Edit Filters** in the **Flowsheet Results** section to create a filter.
27. Figure *Data Filter Dialog* displays the Data Filter Editor.
28. Add the filter for initial data.
 1. Click **New Filter**, and enter “f1” as the filter name.
 2. Type the **Filter expression**: `c(“set”) == “UQ_Ensemble”`.
29. Click **Done**.

Variable

Selection

In this section, input and output variables need to be selected. Generally, any input variables that vary in the data set should be selected. However, in some cases, variables may be found to have no, or very little, effect on the outputs. Only the output variables of interest need to be selected. Note: Each output is independent from each other and for the model building, selecting one output is the same as selecting more.

30. Select the **Variables** tab (Figure *Variable Selection*).
31. Select the checkbox for both input variables.

Fig. 11: Sample Methods

Fig. 12: Run Samples

Fig. 13: Surrogate Data

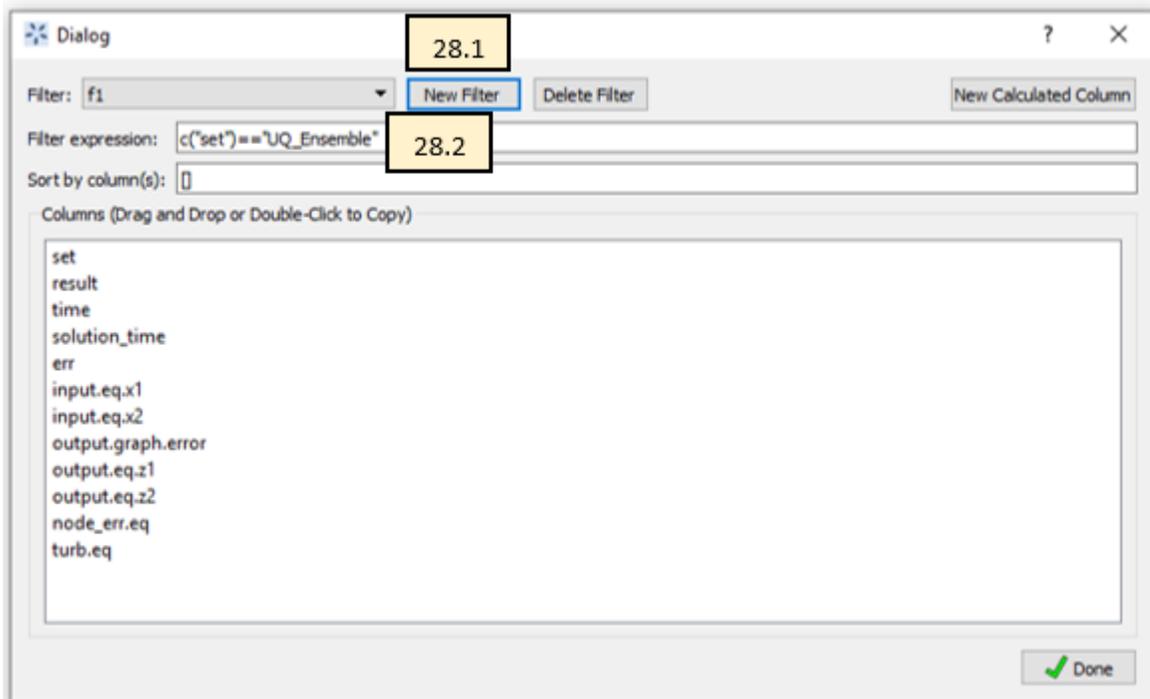


Fig. 14: Data Filter Dialog

32. Select the checkbox for both output variables.

Fig. 15: Variable Selection

Method

Settings

The most important feature to generate “good” algebraic models is to configure the settings accordingly to the problem to be solved. Each setting has a good description in FOQUS. The JSON parser is used to read method settings values. Strings must be contained in quotes. Lists have the following format: [element 1, element 2].

33. Click on the **Method Settings** tab (see Figure *ALAMO Method Settings* and *ALAMO Method Settings Continued*).
34. Set the **FOQUS Model (for UQ)** to “alamo_surrogate_uq.py.”
35. Set the **FOQUS Model (for Flowsheet)** to “alamo_surrogate_fs.py”
36. Set **Initial Data Filter** to “f1”
37. Set **SAMPLER** to select the adaptive sampling method: “None” “Random” or “SNOBFIT.” Use “None” in this tutorial.
38. Set **MONOMIALPOWER** to select the single variable term powers to [1,2,3].
39. Set **MULTI2POWER** to select the two variable term powers to [1].
40. Select functions to be considered as basis functions (**EXPCFNS**, **LOGCFNS**, **SINCFNS**, **COSCFNS**, **LINFCNS**, **CONSTANT**).
41. Leave the rest of settings as default (see Table *ALAMO Method Settings*).
42. Save this FOQUS session for use in the ACOSSO and BSS-ANOVA tutorials.

Execution

43. Click the **Run** icon at the top of the window.
44. The **ALAMO Execution** tab starts displaying execution file path, sub-directories, input files, and output files.
 1. ALAMO version.
 2. License Information.
 3. Step 0 displays the data set to be used by ALAMO.
 4. Step 1 displays the modeler used by ALAMO to generate the algebraic model.
 5. Once the surrogate model has finished, the equations are displayed in the execution window. It may be necessary to scroll up a little. The result is shown in Figure *ALAMO Execution*.
 6. Finally, the statistics display the quality metrics of the models generated.

Setting Name	Value	Description
1 Initial Data Filter	"f1"	Filter to be applied to the initial data set.
2 Validation Data Filter	"all"	Data set used to compute model errors at the validation phase.
3 MAXTIME	2000	Maximum total execution time in seconds.
4 MINPOINTS	0	Convergence is assessed only if the simulator is able to compute the output variables for at least MINPOINTS out of the data points requested by ALAMO.
5 NSAMPLE	0	Number of data points to be generated by sampling before any model is built.
6 MAXSIM	5	Maximum number of successive simulator failures allowed before quit
7 SAMPLER	"None"	Adaptive sampling method to be used. If adaptive sampling is used, also set MAXITER below.
8 MAXITER	1	Maximum number of ALAMO iterations, 1 = no adaptive sampling, 0 = no limit
9 PRESET	-111111	Value to be used if the simulation fails.
10 MONOMIALPOWER	[1, 2, 3]	Vector of monomial powers considered in basis functions. Use an empty vector for none.
11 MULTI2POWER	[1]	Vector of powers to be considered for pairwise combinations in basis functions. Empty vector for none.
12 MULTI3POWER	[]	Vector of three variables combinations of powers to be considered as basis functions.
13 RATIOPOWER	[]	Vector of ratio combinations of powers to be considered in the basis functions.
14 EXPFCNS	<input checked="" type="checkbox"/>	Use or not of exponential functions as basis functions in the model.
15 LOGFCNS	<input checked="" type="checkbox"/>	Logarithmic functions are considered as basis functions if true; otherwise, they are not considered.
16 SINFCNS	<input type="checkbox"/>	Sine functions are considered as basis functions if true; otherwise, they are not considered.
17 COSFCNS	<input type="checkbox"/>	Cosine functions are considered as basis functions if true; otherwise, they are not considered.
18 LINFCNS	<input checked="" type="checkbox"/>	Linear functions are considered as basis functions if true; otherwise, they are not considered.
19 CONSTANT	<input checked="" type="checkbox"/>	A constant will be considered as a basis function if true; otherwise, its not considered.
20 CUSTOMBAS	[]	A list of user-supplied custom basis functions can be provided by the user.
21 MODELER	"BIC"	Fitness metric to be used for model building.

Fig. 16: ALAMO Method Settings

Setting Name	Value	Description
23 SCREENER	"No Screening"	Regularization method is used to reduce the number of potential basis functions before optimization.
24 SCALEZ	<input type="checkbox"/>	If used, the variables are scaled prior to the optimization problem is solved.
25 GAMS	"gams"	GAMS path is needed. GAMS is the software used to solve the optimization problems.
26 GAMSSOLVER	"BARON"	Name of preferred GAMS solver for solving ALAMO's mixed-integer quadratic subproblems.
27 SOLVEMIP	<input type="checkbox"/>	GAMS will be used to solve ALAMO's MIPs/MIQPs if checked
28 FUNFORM	"Fortran"	Format for printing basis functions and models found by ALAMO. Fortran must be selected to generate FOQUS UQ and flowsheet models.
29 MIPOPTCA	0.05	Absolute convergence tolerance for mixed-integer optimization problems.
30 MIPOPTCR	0.0001	Relative convergence tolerance for mixed-integer optimization problems.
31 LINEARERROR	<input type="checkbox"/>	If true, a linear objective is used when solving the mixed-integer optimization problems.
32 CONREG	<input type="checkbox"/>	Specify whether a constraint regression is used or not.
33 CRNCUSTOM	<input type="checkbox"/>	If true, constraints need to be entered in variables tab.
34 CRNINITIAL	0	Number of random bounding points at which constraints are sampled initially.
35 CRNMAXITER	10	Maximum allowed constrained regressions iterations.
36 CRNVIOL	100	Number of bounding points added per round per bound in each iteration.
37 CRNTRIALS	100	Number of random trial bounding points per round of constrained regression.
38 CRTOL	0.001	Tolerance within which custom constraints must be satisfied. Real greater than 1e-5 is expected
39 Input File	"alamo.alm"	File name for ALAMO input file.
40 FOQUS Model (for UQ)	"alamo_surrogate_uq.py"	.py file for UQ analysis.
41 FOQUS Model (for Flowsheet)	"alamo_surrogate_fs.py"	.py file flowsheet plugin, saved to user_plugins in the working directory.
42 Pyomo Model for Optimization	"alamo_surrogate_pyomo_optim.py"	.py file, meant for surrogate based optimization to be used within FOQUS optimizer plugin, saved to user_plugins in the working directory.
43 Standalone Pyomo Model for Optimization	"alamo_surrogate_pyomo_optim_standalone.py"	.py file, meant for surrogate based optimization to be used standalone, saved to user_plugins in the working directory.

Fig. 17: ALAMO Method Settings Continued

Fig. 18: ALAMO Execution

Results

The results are exported as a PSUADE driver file that can be used perform UQ analysis of the models, and a FOQUS Python plugin model that allows it to be used in a FOQUS flowsheet. The equations can also be viewed in the results section.

See tutorial Section *Tutorial 4: Surrogates with UQ Tools* and *Tutorial 5: Surrogates with the Flowsheet* for information about analyzing the model with the UQ tools or running the model on the flowsheet.

As mentioned in section 1.5 the method settings are very important. A brief description and hints are included in Table *ALAMO Method Settings*.

Method Settings	Description
Initial Data Filter	Filter to be applied to the initial data set. Data filters help
Validation Data filter	Data set used to compute model errors at the validation pl
SAMPLER	Adaptative sampling method to be used. Options: “None”
MAXTIME	Maximum execution time in seconds. This time includes
MINPOINTS	Convergence is assessed only if the simulator is able to co
PRESET	Value to be used if the simulator fails. This value must be
MONOMIALPOWERS	Vector of monomial powers to be considered as basis func
MULTI2POWER	Vector of pairwise combination of powers to be considere
MULTI3POWER	Vector of three variables combinations of powers to be co
EXPCFNS, LOGCFNS, SINCFNS, COSCFNS, LINCFSN, CONSTANT	Use or not of exp, log, sin, cos, linear, and constant functi
RATIOPOWER	Vector of ratio combinations of powers to be considered i
Radial Basis Functions	Radial basis functions centered around the data set provid
RBF parameter	Constant penalty used in the Gaussian radial basis functio
Modeler	Fitness metric to be used for model building. Options: BI
ConvPen	Convex penalty term. Used if Convex Penalty is selected.
Regularizer	Regularization method is used to reduce the number of po
Tolrelmetric	Convergence tolerance for the chosen fitness metric is nee
ScaleZ	If used, the variables are scaled prior to the optimization p
GAMS	GAMS is the software used to solve the optimization prob
GAMS Solver	Solver to be used by GAMS to solve the optimization prob
MIPOPTCR	Relative convergence tolerance for the optimization proble
MIPOPTCA	Absolute convergence tolerance for mixed-integer optimiz
LINEARERROR	If true, a linear objective function is used when solving th
CONREG	Specify whether constraint regression is used or not, if tru
CRNCUSTOM	If true, Custom constraints are entered in the Variable tab
CRNINITIAL	Number of random bounding points at which constraints a
CRNMAXITER	Maximum allowed constrained regressions iterations. Cor
CRNVIOL	Number of bounding points added per round per bound in
CRNTRIALS	Number of random trial bounding points per round of con
CUSTOMBAS	A list of user-supplied custom basis functions can be prov

This tutorial covers the ACOSSO surrogate modeling method. The Adaptive Component Selection and Shrinkage Operator (ACOSSO) surface approximation was developed under the Smoothing Spline Analysis of Variance (SS-ANOVA) modeling framework (*Storlie et al. 2011*). As it is a smoothing type method, ACOSSO works best when the underlying function is somewhat smooth. For functions which are known to have sharp changes or peaks, etc., other methods may be more appropriate. Since it implicitly performs variable selection, ACOSSO can also work well when there are a large number of input variables. The ACOSSO procedure also allows for categorical inputs (*Storlie et al. 2013*).

This tutorial uses the same flowsheet and sample setup as the ALAMO tutorial in Section *Tutorial 1: ALAMO*.

The FOQUS file for this tutorial is **Surrogate_Tutorial_1.foqus**, and this file is located in:
`examples/tutorial_files/Surrogates`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

The statistics software “R” is also required to use ACOSSO and BSS-ANOVA. Before starting this tutorial, you will need to install R version 3.1 or later (see <https://cran.r-project.org/>).

Once R is installed, you will need to install the “quadprog” package. ACOSSO requires this package for solving quadratic programming problems. You will only need to perform this step once.

1. Start R. In Windows, this must be done with administrative privileges. Either run this from an administrator account, or right-click “R x64 3.1.2” and click “Run with administrator” and type in administrator credentials.
2. Inside the R console, type:
 - `install.packages('quadprog')`
 - `library(quadprog)`
 - `q()`

The first line installs the package. If prompted for a CRAN mirror, select the one closest to you geographically. The second line loads the package. The last line quits R. If prompted to save workspace image, choose ‘y’.

Once you have done these steps, ACOSSO is ready to be invoked inside FOQUS.

1. Set the path to the RScript executable.
 1. Click the **Settings** button in the Home window.
 2. Change the RScript path if necessary. The **Browse** button opens a file browser that can be used to set the path.
2. Complete the ALAMO tutorial in Section *Tutorial 1: ALAMO* through Step 32, load the FOQUS session saved after completing the ALAMO tutorial, or load the “Surrogate_Tutorial_1.foqus” file from the `examples/tutorial_files/Surrogates` folder.
3. Click the **Surrogates** button in the Home window (Figure *ACOSSO Session Set Up*).
4. Select “ACOSSO” in the **Tool** drop-down list.
5. Select the **Method Settings** tab.
6. Set “Data Filter” to “Initial.”
7. Set “Use Flowsheet Data” to “Yes.”
8. Set “FOQUS Model (for UQ)” to “ACOSSO_Tutorial_UQ.py.”

9. Set “FOQUS Model (for Flowsheet)” to “ACOSSO_Tutorial_FS.py.”
10. Click the **Run** icon (Figure *ACOSSO Session Set Up*).

Fig. 19: ACOSSO Session Set Up

11. The execution window will automatically display. While ACOSSO is running, the execution window may show warnings, but this is normal.
12. When the run completes, a UQ driver file is created, allowing the ACOSSO surrogate to be used as a user-defined response surface in UQ analyses. (See Section *Tutorial 4: Surrogates with UQ Tools*.)
13. ACOSSO also produces a flowsheet plugin; however.

Tutorial**3:****BSS-ANOVA**

This tutorial covers the BSS-ANOVA surrogate modeling method. The Bayesian Smoothing Spline ANOVA (BSS-ANOVA) is essentially a Bayesian version of ACOSSO (Reich et al. 2009). It is Gaussian Process (GP) model with a non-conventional covariance function that borrows its form from SS-ANOVA. It tackles the high dimensionality (of inputs) on two fronts: (1) variable selection to eliminate uninformative variables from the model and (2) restricting the level of interactions involved among the variables in the model. This is done through a fully Bayesian approach which can also allow for categorical input variables with relative ease. Since it is closely related to ACOSSO, it generally works well in similar settings as ACOSSO. The BSS-ANOVA procedure also allows for categorical inputs (Storlie et al. 2013). In this current implementation, BSS-ANOVA is more computationally intensive than ACOSSO, so ACOSSO is preferred for faster surrogate generation.

This tutorial uses the same flowsheet and sample setup as the ALAMO tutorial in Section *Tutorial 1: ALAMO*.

The FOQUS file for this tutorial is **Surrogate_Tutorial_1.foqus**, and this file is located in:
`examples/tutorial_files/Surrogates`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

The statistics software “R” is also required to use ACOSSO and BSS-ANOVA. Before starting this tutorial, you will need to install R version 3.1 or later (see <http://cran.r-project.org/>).

1. Set the path to the RScript executable.
 1. Click the **Settings** button from the Home window.
 2. Change the RScript path if necessary. The **Browse** button opens a file browser that can be used to set the path.
2. Complete the ALAMO tutorial in Section *Tutorial 1: ALAMO* through Step 32, load the FOQUS session saved after completing the ALAMO tutorial, or load the “Surrogate_Tutorial_1.foqus” file from the `examples/tutorial_files/Surrogates` folder.
3. Click the **Surrogates** button from the Home window (Figure *BSS-ANOVA Session Set Up*).
4. Select “BSS-ANOVA” in the **Tool** drop-down list.
5. Select the **Method Settings** tab.
6. Set “Data Filter” to “Initial.”
7. Set “Use Flowsheet Data” to “Yes.”
8. Set “FOQUS Model (for UQ)” to “bssanova_tutorial_uq.py.”

9. Set “FOQUS Model (for Flowsheet)” to “bssanova_tutorial_fs.py.”
10. Click the **Run** icon (Figure *BSS-ANOVA Session Set Up*).

Fig. 20: BSS-ANOVA Session Set Up

11. The execution window will automatically display. While BSS-ANOVA is running, the execution window may show warnings, but this is normal.
12. When the run completes, a UQ driver file is created, allowing the BSS-ANOVA surrogate to be used as a user-defined response surface in UQ analyses. (See Section *Tutorial 4: Surrogates with UQ Tools*.)
13. BSS-ANOVA also produces a flowsheet plugin.

Tutorial 4: Surrogates with UQ Tools

For the purpose of this tutorial, we will use ACOSSO to demonstrate the use of a surrogate within the UQ module. The steps are the same regardless of the surrogate tool chosen.

To perform the UQ analysis, Python is required for use the “User Regression” response surface that will be used.

Before starting this tutorial, you will need to install Python 2.7.x (not Python 3). (See <https://www.python.org/downloads/>). In addition, if *.py files have been re-associated with other executables (e.g. editors), please change the association back to python.exe.

The FOQUS file for this tutorial is **Rosenbrock_no_vectors.foqus**, and this file is located in:
`examples/tutorial_files/Surrogates`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

1. Load a fresh session by clicking the Session button from the Home window. Select Open Session and then navigate to the above-mentioned folder, and select “Rosenbrock_no_vectors.foqus.” This will load a session with a simple flowsheet containing a single node.
2. Click Settings and ensure that (1) FOQUS Flowsheet Run Method is set to “Local”, and that (2) proper paths are set for PSUADE and RScript.
3. Train an ACOSSO surrogate of this node by clicking the **Surrogates** button from the Home window.
 1. Click Add Samples and select “Use Flowsheet”. This will display the Simulation Ensemble Setup dialog.
 2. Within this dialog, ensure all variables are set to “Variable” type in the Distributions tab. In the Sampling scheme tab, select “Monte Carlo” as your sampling scheme, set the number of samples to 100, and then click Generate Samples to generate the set of input values. Click Done to return to the Surrogates screen.
 3. Once sample generation completes, click the Uncertainty button from the Home window.
 4. Click the Launch button to generate the samples.
 5. Click the Surrogates button from the Home window. The Data tab of the Surrogates screen should now displays a Flowsheet Results table that is populated with the values of the new input samples.
 6. From the Variables tab, select all of the checkboxes. (There should be six checkboxes for input variables and one checkbox for output variable.) Here, you are defining the inputs and outputs for your surrogate function.
 7. From the Method Settings tab, note the name of the file next to “FOQUS Model (for UQ)”. This will be the name of the UQ driver file that contains the Python code that implements the surrogate function.

8. On top of this screen, select “ACOSSO” as your surrogate tool from the Tool drop-down list and then click on the green arrow to start training the surrogate.
 9. Once complete, a popup window will display, reminding you of the location of the drive file. Note the location as you will need this information later inside the UQ module.
4. Perform a response-surface-based uncertainty analysis by clicking the **Uncertainty** button from the Home window.
 1. In the Uncertainty Quantification Simulation Ensembles table. A row corresponding to the ensemble that was just generated for surrogate training should be displayed. This same ensemble can be used or a new one can be created to be used as the test data set for analysis. In the row corresponding to the ensemble to be analyzed, click the Analyze button to proceed. This action will bring up an analysis dialog.
 2. Within this analysis dialog, navigate to “Analysis” section. For Step 1, select “Response Surface”. For Step 3, select “User Regression” in the first drop-down list. Lastly, for “User Regression File”, browse to the same location as the UQ driver file that was generated within the Surrogates module. (This is the same location that was previously noted from the popup message.) At this point, your surrogate function is now set up as a user-defined response surface and all response-surface-based UQ analyses are accessible.
 3. Click Validate (Step 4) to perform response surface validation. Once complete, a figure with cross-validation results will be displayed: a histogram of errors to the left and a plot of predicted values versus actual values to the right. For more information, refer to the UQ Tutorial in Section[*tutorial.uq.rs*]. The response surface validation result is saved in the “Analyses Performed” section.
 4. Once a “Response Surface” has been validated, other UQ analysis options are available. Choose “Uncertainty Analysis” in Step 5 and click Analyze to perform uncertainty analysis using your ACOSSO surrogate.
 5. Save the FOQUS session after all the required uncertainty analysis results are saved.

During validation, if the error, “RSAnalyzer: RSTest_hs.m does not exist.” displays, this is likely caused by incompatibility with the surrogate and the test data. An example scenario might be your test data has six inputs, but your surrogate assumes five inputs. This is easily fixed by returning to the Surrogates screen, clicking on the **Variables** tab, and making sure the appropriate selections are made (i.e., check off six inputs instead of just five).

Important Note

Before opening the FOQUS file containing the response surface validation result, ensure that the corresponding UQ driver file is present in the same path in the machine, from where it was accessed in the “User Regression File” section, for validation. This is important to avoid problems while opening the FOQUS file, which may happen if it isn’t able to find the UQ driver file.

Ideally, the UQ driver file should be present in the FOQUS working directory itself to avoid confusion.

Tutorial 5: Surrogates with the Flowsheet

This section provides a brief tutorial for using the flowsheet plugin models generated by surrogate modeling methods.

In a future FOQUS release all surrogate modeling methods will produce a model that can be run in a FOQUS flowsheet. **Currently iREVEAL (part of the CCSI Toolset) does not produce a flowsheet model.**

Before doing this tutorial complete the ALAMO tutorial in Section :ref:`sec.surrogate.alamo`.

1. Open FOQUS. If FOQUS has not been closed since completing the ALAMO tutorial, close it and reopen it. There is a known issue where existing flowsheet model plugins may not update until FOQUS is restarted.
2. Enter “FS_Plugin_Tutorial” as the Session Name.
3. Click the **Flowsheet** button from the Home window.
4. Click the **Add Node** icon in the left toolbar (see Figure *Plugin Flowsheet*).
5. Click a location for the node in the Flowsheet area.

6. Enter “model” for the node name (without quotes).
7. Click the **Node Editor** icon in the left toolbar (see Figure *Plugin Flowsheet*).
8. In the **Node Editor**, select “Plugin” from the Model **Type** drop-down list.
9. Select “ALAMO_Tutorial_FS” from the **Model** drop-down list.
10. Set the **Value** of the **Input Variables** “eq.x1” to 2.
11. Set the **Value** of the **Input Variables** “eq.x2” to 3.
12. Click the **Run** icon in the left toolbar (see Figure *Plugin Flowsheet*).
13. Wait for the Flowsheet evaluation to complete. It should finish successfully.
14. Check the value of the **Output Variables**; the approximate values should be $z_1 = 5$ and $z_2 = 13$.

Fig. 21: Plugin Flowsheet

Tutorial 6: Neural Networks

This tutorial covers the TensorFlow Keras Neural Network surrogate modeling method via the plugin “keras_nn”. The Surrogates module also supports PyTorch and Scikit-learn Neural Network surrogate modeling methods, which follow the workflow below using the “pytorch_nn” and “scikit_nn” plugins, respectively.

More information on TensorFlow Keras model building is described by (*Wu et al. 2020*). Users may follow the recommended workflow to install and use TensorFlow in a Python environment, as described in the TensorFlow documentation: <https://www.tensorflow.org/install>.

Users may obtain a great deal of usage standards and best practices information as described in the PyTorch documentation: <https://pytorch.org/docs/stable/index.html>.

Users may find further information on the Scikit-learn package in the documentation: <https://scikit-learn.org/stable/index.html> and further information on deep learning capabilities as well: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor.

The FOQUS file for this tutorial is **Simple_flow.foqus**, and this file is located in:
`examples/tutorial_files/Flowsheets/Tutorial_4`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

The Python package *tensorflow* must be installed to use this surrogate plugin. Before starting this tutorial, you will need to install the package by referring to the installation instructions in the TensorFlow documentation.

1. Load the “Simple_flow.foqus” file from the `examples/tutorial_files/Flowsheets/Tutorial_4` folder.
2. Click the **Surrogates** button from the Home window (Figure *Keras Neural Network Session Set Up*).
3. Select “keras_nn” in the **Tool** drop-down list. For PyTorch, select “pytorch_nn”. For Scikit-learn, select “scikit_nn”.
4. Select the **Variables** tab and select the desired input and output variables. Note that “graph.error” should not be selected.
5. Select the **Method Settings** tab.
6. Set “Initial Data Filter” to “All”.

7. Set “n_hidden” to “1”, “n_neurons” to “12”, “layer_act” to “relu”, and “out_act” to “sigmoid”.
8. Ensure there are values present for “numpy_seed”, “random_seed”, and “tensorflow_seed” so results are reproducible.
9. Set “epoch” to “500”, “verbose” to “0” (i.e. epoch results will not print during training) and ensure “output_file” has a file name of the form “user_ml_ai_models/[NAME].keras”. NAME may be changed, if desired. The model will be saved to the working directory folder user_ml_ai_models. For PyTorch, the file extension should be “.pt”. For Scikit-learn, the file extension should be “.pkl”.
10. Click the **Run** icon (Figure *Keras Neural Network Session Set Up*).

Fig. 22: Keras Neural Network Session Set Up

10. The execution window will automatically display. While the regression training is finished running, TensorFlow Keras will display some information on the model size and shape.
11. When the run completes, the script produces a saved model file which is compatible with the Machine Learning & Artificial Intelligence Plugin. (See Section *Machine Learning & Artificial Intelligence Flowsheet Model Plugins*.)

In the future, these plugins will support additional items yielding enhanced surrogate modeling capabilities.

- Expanded data normalization options
- Offline data loading from a file, e.g. CSV, Excel, PSUADE
- Support for regression of multi-output neural networks
- Automatic regression parameter selection during model training from user-supplied parameter options

SEQUENTIAL DESIGN OF EXPERIMENTS (SDOE)

8.1 Contents

8.1.1 Sequential Design of Experiments (SDOE)

Experimenters often begin an experiment with imperfect knowledge of the underlying relationship they seek to model, and may have a variety of goals that they would like to accomplish with the experiment. In this chapter, we describe how sequential design of experiments can help make the best use of resources and improve the quality of learning. We describe the different types of space filling designs that can help accomplish this, define basic terminology, and show a common sequence of steps that are applicable to many experiments. We show the basics for the types of designs supported in the SDOE module, and provide some examples to illustrate the methods.

A sequential design of experiments strategy allows for adaptive learning based on incoming results as the experiment is being run. The SDOE module in FOQUS allows the experimenter to flexibly incorporate this strategy into their designed experimental planning to allow for maximally relevant information to be collected. Statistical design of experiments is an important strategy to improve the amount of information that can be gleaned from the overall experiment. It leverages principles of putting experimental runs where they are of maximum value, the interdependence of the runs to estimate model parameters, and robustness to the variability of results that can be obtained when the same experimental conditions are repeated. There are two major categories of designed experiments: those for which a physical experiment is being run, and designs for a computer experiment where the output from a computer model (based on underlying science or engineering theory) is explored. There are also experimental situations when the goal is to collect both from a physical experiment as well as the computer model to compare them and to calibrate some of the computer model parameters to best match what is observed. The methods available in the SDOE module can be beneficial for all three of these cases. They present opportunities for accelerated learning through strategic selection and updating of experimental runs that can adapt to multiple goals.

The current version of the SDOE module has functionality that can produce flexible space-filling designs. Currently, three types of space-filling designs are supported:

Uniform Space Filling (USF) designs space design points evenly, or uniformly, throughout the user-specified input space. These designs are common in physical and computer experiments where the goal is to have data collected throughout the region. They are well suited to exploration, and being able to predict results at a new input combination, as there will be some data available close by. To use the Uniform Space Filling design capability in the SDOE module, the only requirement for the user is that the candidate set contains a column for each of the inputs and a row for each possible run. It is also recommended (but not required) to have an index column to be able to track which rows of the candidate set are selected in the constructed design.

Non-Uniform Space Filling (NUSF) designs maintain the goal of having design points spread throughout the desired input space, but add a feature of being able to emphasize some regions more than others. This adds flexibility to the experimentation, when the user is able to tune the design to have as close to uniform as desired or as strongly concentrated in one or more regions as desired. This is newly developed capability, which has just been introduced into the statistical and design of experiments literature, and has been added to the SDOE module. It provides the experimenter with the ability to tailor the design to what is needed. To use the Non-Uniform Space Filling design

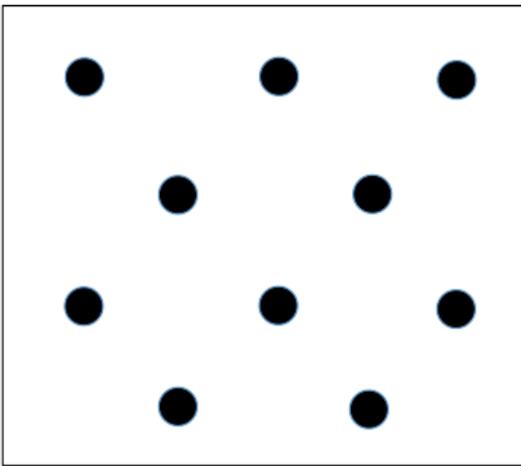
capability in the SDoE module, the requirements are that the candidate set contains (a) one column for each of the inputs to be used to construct the design, and (b) one column for the weights to be assigned to each candidate point, where larger values are weighted more heavily and will result in a higher density of points close to those locations.

The Index column is again recommended, but not required.

Input-Response Space Filling (IRSF) designs seek to spread points evenly throughout the input space, but simultaneously also spread points evenly throughout the response space. This is another newly developed capability, recently introduced into the statistical and design of experiments literature, and has been added to the SDoE module.

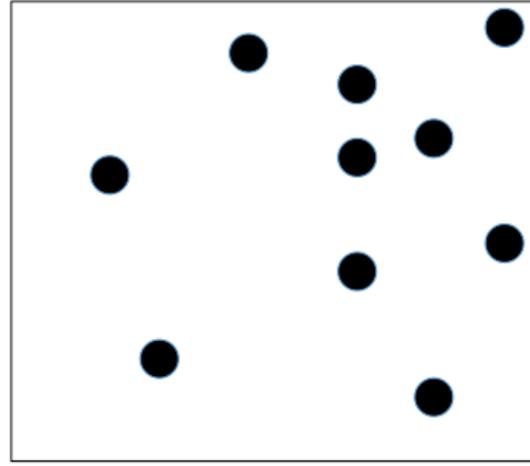
It provides the experimenter with the ability to choose a design from a spectrum of “best” designs with different weights for space-filling in the input and response spaces. Thus, if space-filling is more important to the experimenter in one dimension than another, or if an equal balance is desired, the experimenter can choose the design with those desired qualities from the Pareto front of designs given. This capability could be especially useful, for example, in a multi-step system where a current response variable would become another input in the future, for which space-filling would be desired in a future design. To use the Input-Response Space Filling design capability in the SDoE module, the requirements are that the candidate set contains (a) one column for each of the inputs to be used to construct the design, and (b) one column for each of the responses (at least one is required). Therefore, there must exist some model such that a likely response can be calculated for each input combination. This type of design is only recommended in situations when the model for the likely response values has been previously validated.

Uniform Space Filling design



User provides:
- Candidate set of possible
input combinations

Non-Uniform Space Filling design



User provides:
- Candidate set of possible
input combinations with a
weight for each

Fig. 1: Comparison of USF and NUSF designs

Key features of all approaches available in this module are: a) designs will be constructed by selecting from a user-provided candidate set of input combinations, and b) historical data, which has already been collected can be integrated into the design construction to ensure that new data are collected with a view to account for where data are already available.

Why**Space-Filling****Designs?**

Space-filling designs are a design of experiments strategy that is well suited to both physical experiments with an accompanying model to describe the process and to computer experiments. The idea behind a space-filling design is that the design points are spread throughout the input space of interest. If the goal is to predict values of the response for a new set of input combinations within the ranges of the inputs, then having data spread throughout the space means that there should be an observed data point relatively close to where the new prediction is sought, regardless of the new location.

In addition, if there is a model for the process, then having data spread throughout the input space means that the consistency of the model to the observed data can be evaluated at multiple locations to look for possible discrepancies and to quantify the magnitude of those differences throughout the input space.

Hence, for a variety of criteria, a space-filling design might serve as good choice for exploration and for understanding the relationship between the inputs and the response without making a large number of assumptions about the nature of the underlying relationship. As we will see in subsequent sections and examples, the sequential approach allows for great flexibility to leverage what has been learned in early stages to influence the later choices of designs. In addition, the candidate-based approach that is supported in this module has the advantage that it can make the space-filling approach easier to adapt to design space constraints and specialized design objectives that may evolve through the stages of the sequential design.

We begin with some basic terminology that will help provide structure to the process and instructions below.

- **Input factors** – these are the controllable experimental settings that are manipulated during the experiment. It is important to carefully define the ranges of interest for the inputs (eg. Temperature in [200°C,400°C]) as well as any logistical or operational constraints on these input factors (eg. Flue Gas Rate < 1000 kg/hr when Temperature > 350°C)
- **Input combinations (or design runs)** – these are the choices of settings for each of the input factors for a particular run of the experiment. It is assumed that the implementers of the experiment are able to set the input factors to the desired operating conditions to match the prescribed choice of settings. It is not uncommon for the experimenter to not have perfect control of the input settings, but in a designed experiment, it is important to have a target value for each input and also to record the observed value if in fact it is different than what was intended. This allows for more precise estimation of the model and improved prediction.
- **Input space (or design space)** – the region of interest for the input factors in which the experiment will be run. This is typically constructed by combining the individual input factor ranges, and then adapting the region to take into account any constraints. Any suggested runs of the experiment will be located in this region. The candidate set of runs used by the SDoE module should provide coverage of all regions of this desired input space.
- **Responses (or outputs)** – these are the measured results obtained from each experimental run. Ideally, these are quantitative summaries (measured by a numeric value or possibly a vector of numeric values) of a characteristic of interest resulting from running the process at the prescribed set of operating conditions (eg. CO₂ capture efficiency is a typical response of interest for CCSI).
- **Design criterion / Utility function** – this is a mathematical expression of the goal (or goals) of the experiment that is used to guide the selection of new input combinations, based on the prior information before the start of the experiment and during the running of the experiment. The design criterion can be based on a single goal or multiple competing goals, and can be either static throughout the experiment or evolve as goals change in importance over the course of the experiment. Common choices of goals for the experiment are:
 1. exploring the region of interest,
 2. improving the precision (or reducing the uncertainty) in the estimation of model parameters,
 3. improving the precision of prediction for new observations in the design region,
 4. assessing and quantifying the discrepancy between the model and data, or
 5. optimizing the value of responses of interest.

An ideal design of experiment strategy uses the design criterion to evaluate potential choices of input combinations to maximize the improvement in the criterion over the available candidates. If the optimal design strategy is sequential, then the goal is to use early results from the beginning of the experiment to guide the choice of new input combinations based on what has already been learned about the responses.

Matching the Design Type to Experiment Goals

At different stages of the sequential design of experiments, different objectives are common. We outline a common progression of objectives for experiments that we have worked with in the CCSI project. Typically, an initial **pilot** study is conducted to show that the right data can be collected and that measurements can be made with the required precision. Often no designed experiment is used for this small study as it is just to establish viability to proceed.

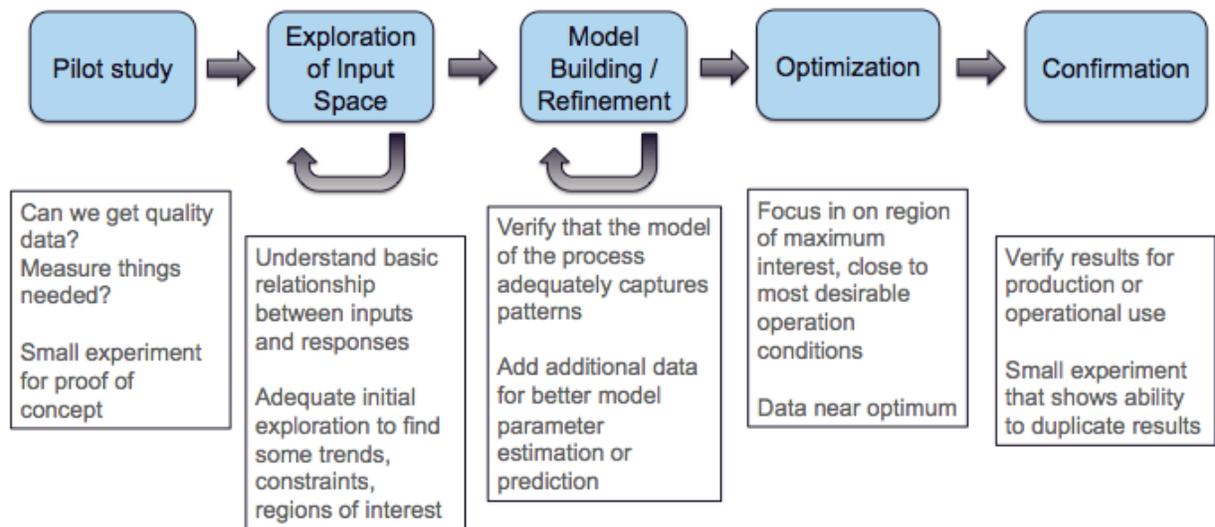


Fig. 2: SDOE sequence of steps

Once the viability of the experimental set-up and measurement system has been established, it is common to proceed to the next step of **exploration**. This is appropriate if little is known about the response and its characteristics. Hence, a first experiment may have the goal of gaining some preliminary understanding of the characteristics of the response across the input region of interest. Depending on how easy it is to collect and process data, this exploration might be done in a single first experiment, or there may be opportunities to do several smaller stages (this is shown in the figure above with the recursive arrow). It is particularly beneficial to do the exploration step in smaller stages if there is uncertainty about what areas of the input space are feasible. This can help save resources by exploring slowly and eliminating regions where there are problems.

After initial exploration, a common next step in the sequence of experiments is **model building** or **model refinement**. For many CCSI experiments, the physical experiments are being collected in conjunction with an underlying science-based model. If a model does not already exist, then one might be developed based on the initial data collected in the previous stage. If a model already exists, then it can be refined by collecting new data where (a) there is maximum uncertainty in prediction, or (b) where there are discrepancies between the data and the model. In this way, the data collection from a physical experiment is used to calibrate the model and provide feedback about where model performance needs improvement (both resolving inaccurate characterization of features and high uncertainty). Often after the first set of data, some regions of the input space perform well, while others have issues.

It is ideal to target new data in regions where it can be most beneficially used to improve the model.

After the experimenter has confidence in the model, it can then be used for **optimization**. This involves using the model to predict regions with desirable values of the response(s) of interest. Often the experiments associated with this stage focus on a smaller region of the input space close to where the optimum lies. The final stage, **confirmation**

is often a very small experiment located right at the location where the model says the response is optimal. The goal of this stage is to verify that the results predicted by the model are matched with what is observed from experimental data. As with the pilot study, often this final stage involves only a small number of runs and no formal designed experiment is run.

We now illustrate these stages with a simple example involving 2 inputs where the candidate set fills a rectangular region defined by the range of each input. In the first stage, the **pilot** study (the two orange dots) are used to establish viability of the test method and measurement system. The second stage, an initial **exploratory experiment** (six blue dots) spreads the points throughout the defined region of interest. Here we start to see the benefit of using a sequential approach as the blue dots take into account the locations where the orange pilot data were collected.

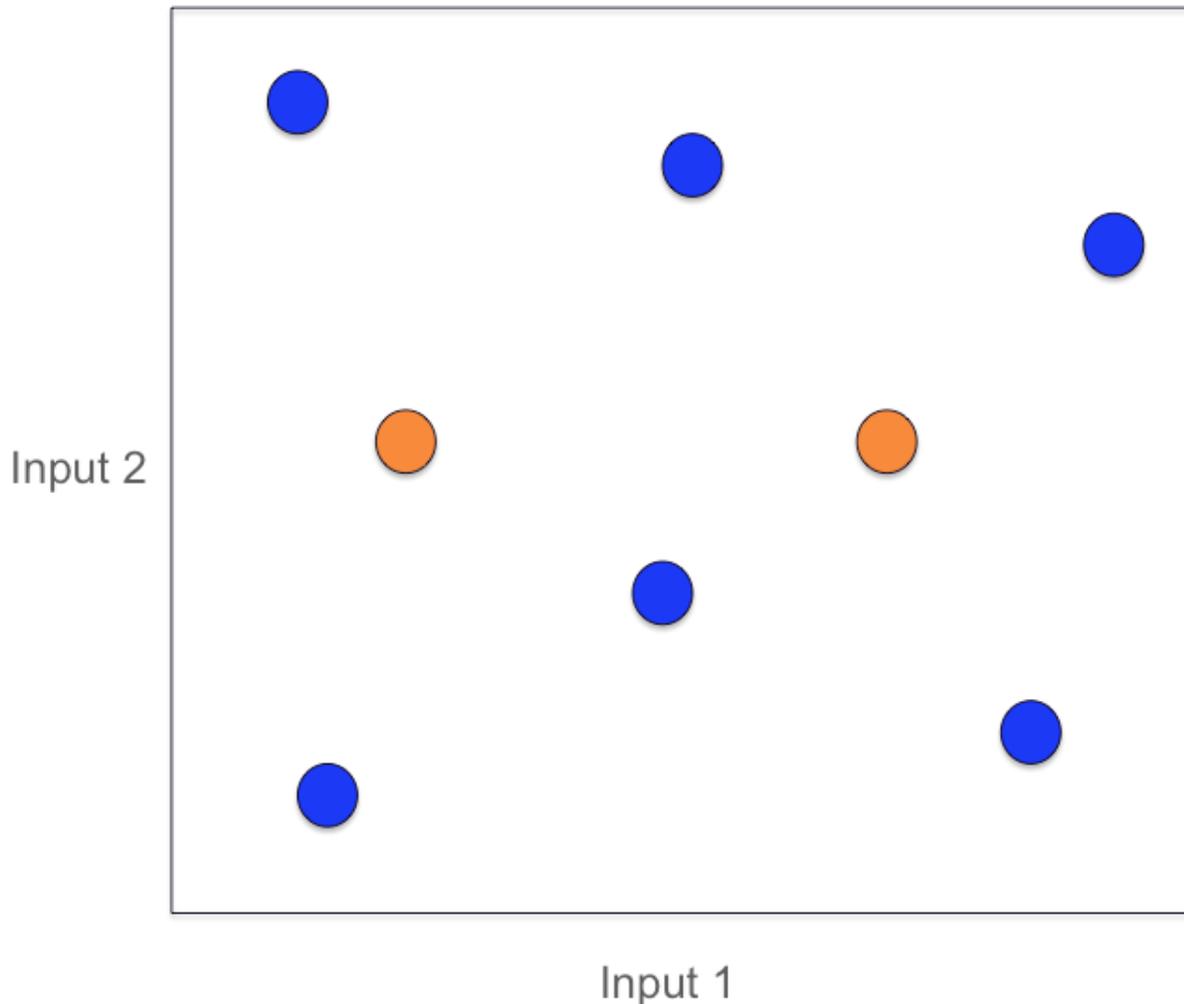


Fig. 3: SDoE Pilot study (orange) and Exploration (blue) stage

Based on this exploration, it may be discovered that one portion of the region (top right) is not viable for data collection, or is not desirable for the observed response values. Hence, in future experiments no data should be collected here. At this point, an initial model is constructed to combine what is known from the experimental data with the underlying science.

In the next stage of experimentation, some additional runs are added (red dots) that are used for **model refinement**. These are placed in regions where there is larger uncertainty in the model predictions and also seek to fill in empty space.

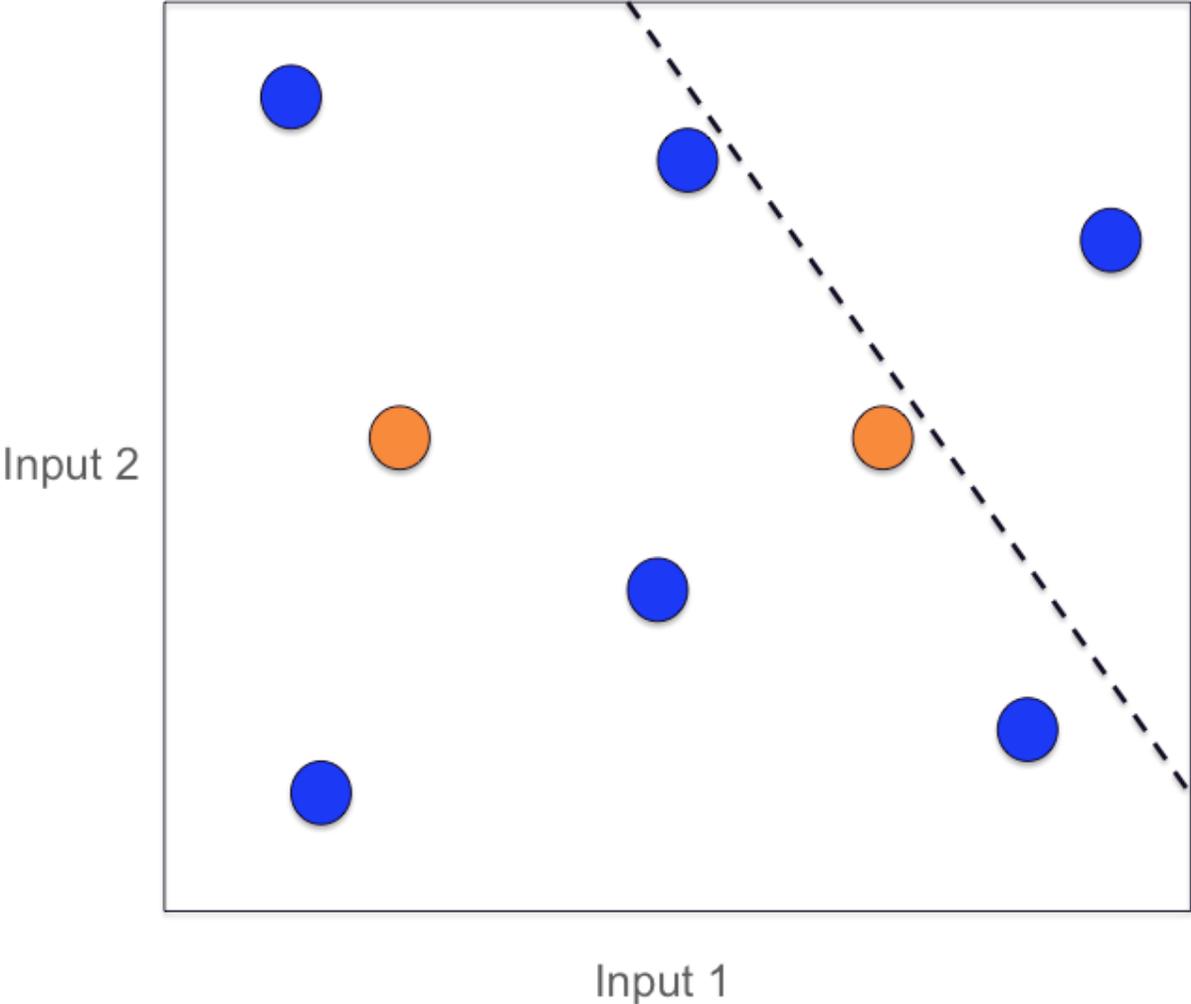


Fig. 4: New Constraint added (dashed black line)

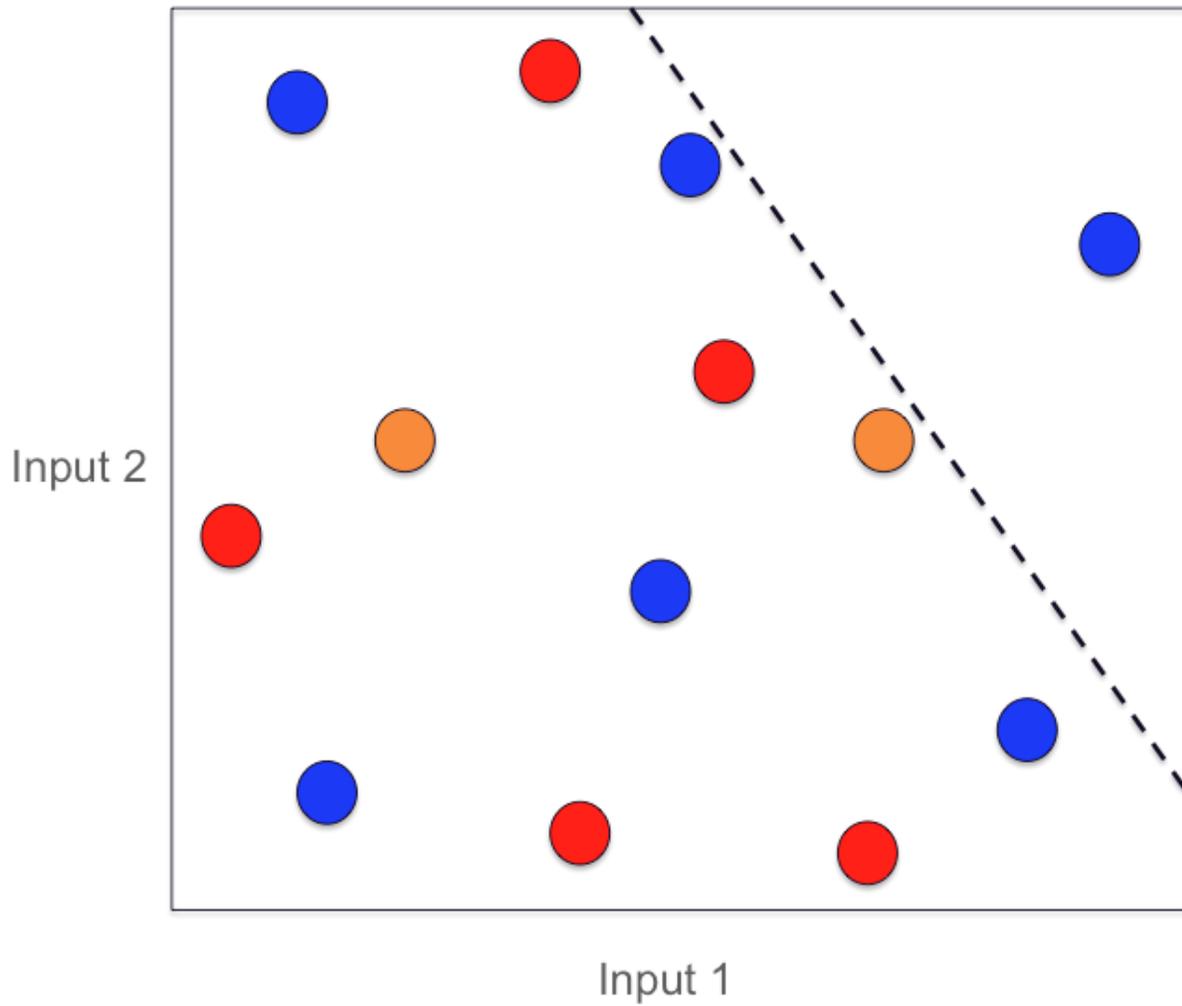


Fig. 5: Model Refining stage of experimentation (red dots)

With the updated model based on the additional data, a region where good response values are possible is identified. This becomes the focus of another experiment for **optimizing** the response. The oval indicates the region of desirable responses, and the three green dots indicate the new input combinations collected to provide additional information.

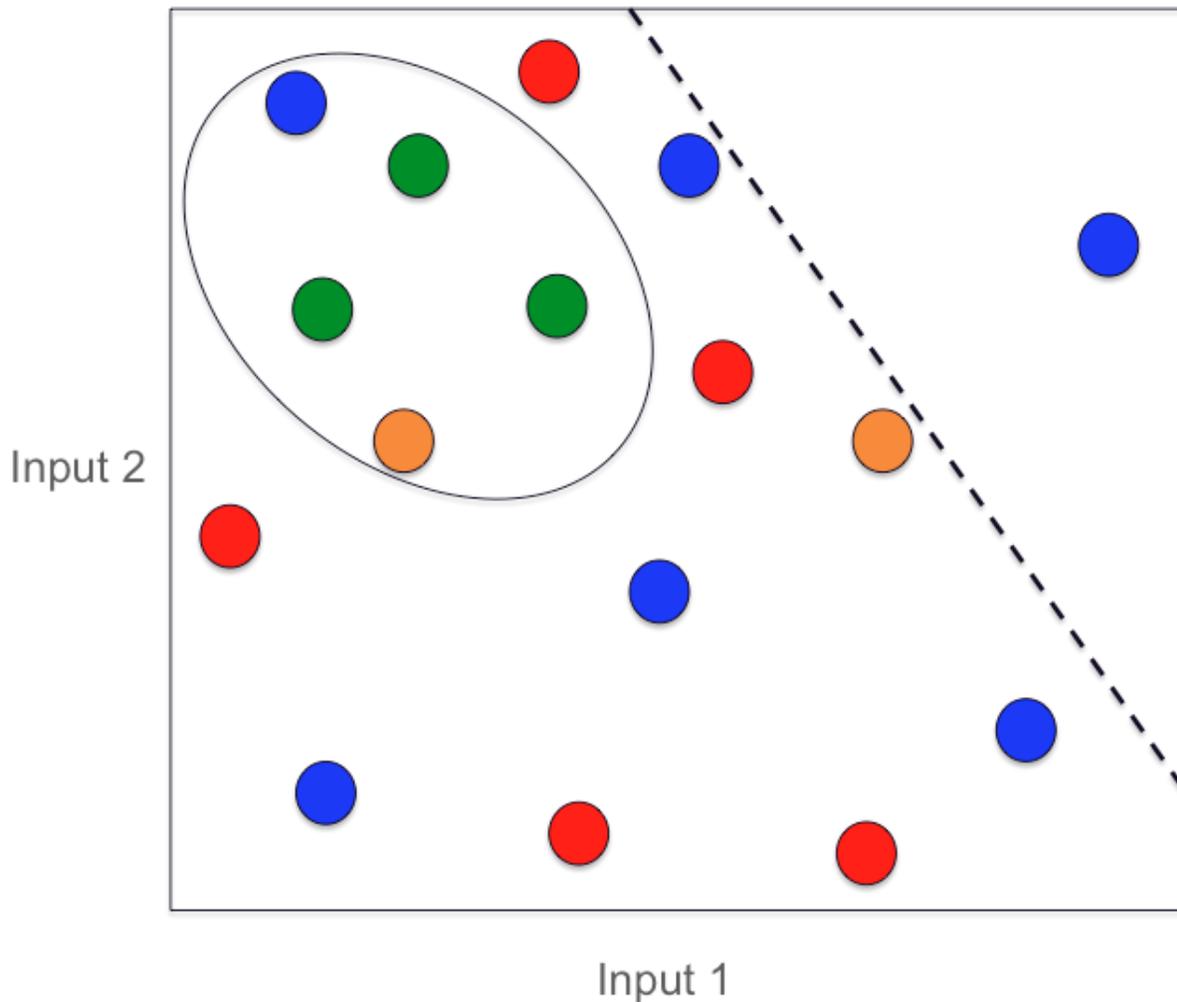


Fig. 6: The optimal region for the responses (oval) with additional runs (green dots)

The final data collection involves two **confirmation** runs (black dots) at the identified optimal location to verify that results are observed to match what the model predicts.

To conclude this example, we illustrate the power of the sequential approach to collecting data. In the figure below, we show the 18 runs collected with the sequential approach (on left) and a typical 18-run space filling design (on right). Both these experiments have the same total budget, but the sequential approach avoids placing much data in the undesirable top right corner as well as has much more data concentrated close to where the overall optimal combination of inputs is located.

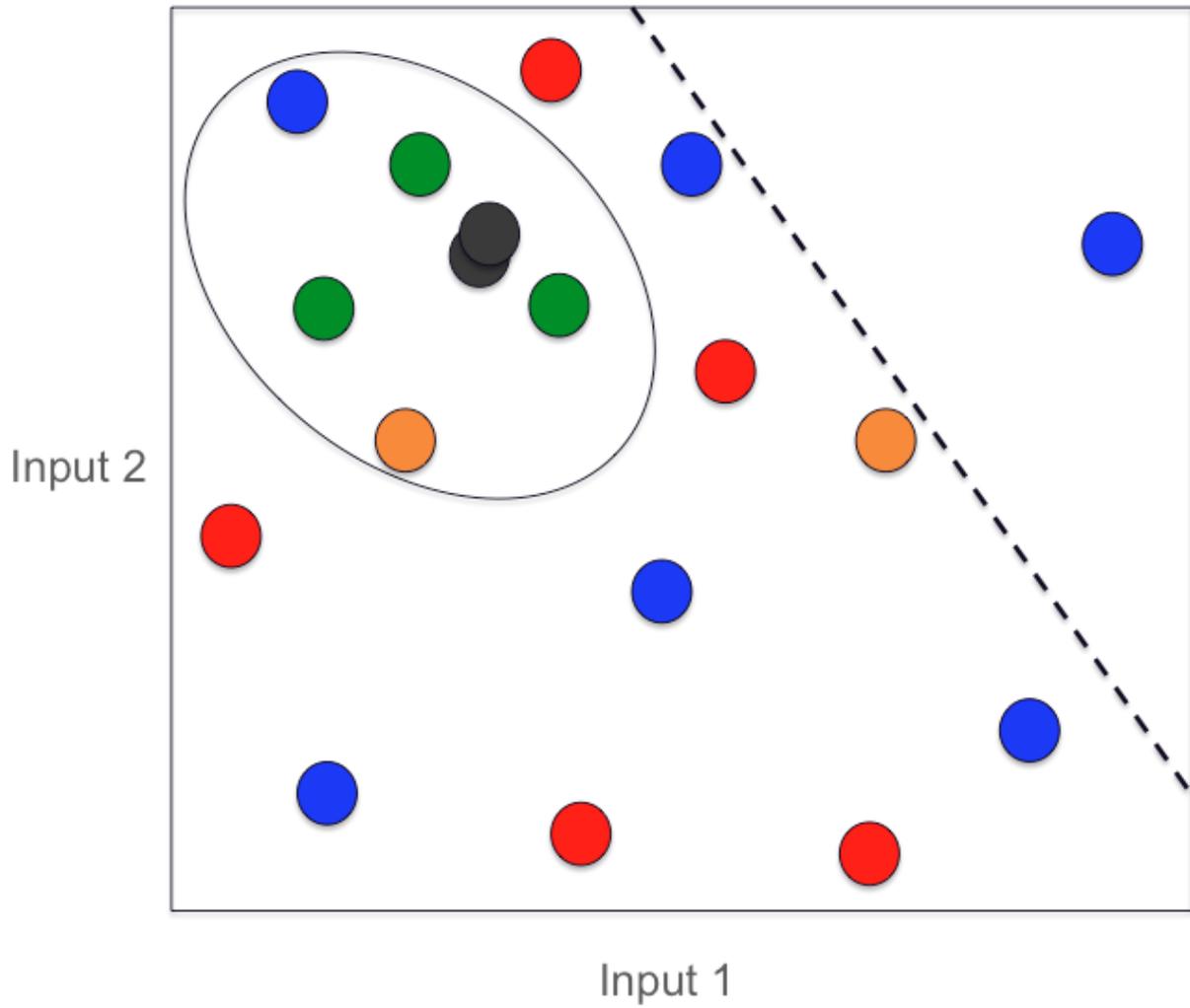


Fig. 7: SDOE confirmation runs (black dots)

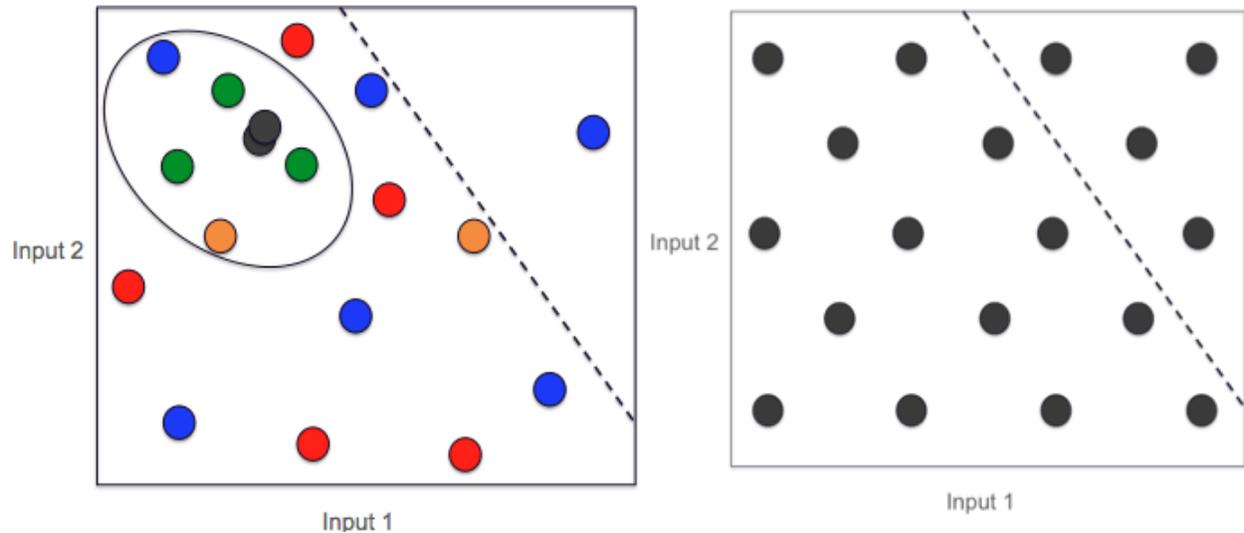


Fig. 8: A comparison of 2 18-run experiments: On left, the sequential approach. On right, the single experiment approach.

8.1.2 Using the SDOE Module - The Basics

In this section, we will describe the basic steps in for creating a design with this module. We first give details for the Uniform Space Filling Design, and then give a second set of details for a Non-Uniform Space Filling design.

When you first click on the **SDOE** button from the main FOQUS homepage, a first window appears. To create a design, the progression of steps takes you through the **Ensemble Selection** box (top left), then a transition triggered by the **Confirm** button to the **Ensemble Aggregation** box, and finally there are optional changes that can be made in the box at the bottom of the window. The final step in this window is to click on which type of design do you want to construct **Uniform Space Filling**, **Non Uniform Space Filling**, or **Input-Response Space Filling**.

Creating a New Candidate Set

To create a new candidate set the user can choose between two options: loading from an existing file or generating a new candidate set providing some ranges for each input.

Note: To use this feature you need to install the latest version of PSUADE. For more details go to section *Install Optional Software*

Loading from File

In the **Ensemble Selection** box, click on the **Load from File...** button to select the file(s) for the construction of the design. Several files can be selected and added to the box listing the chosen files.

For each of the files selected using the pull-down menu, identify them as either a **Candidate** file or a **History** file. **Candidate** .csv files are comprised of possible input combinations from which the design can be constructed. The columns of the file should contain the different input factors that define the dimensions of the input space. The rows of the file each identify one combination of input values that could be selected as a run in the final design. Typically, a good candidate file will have many different candidate runs listed, and they should fill the available ranges of the

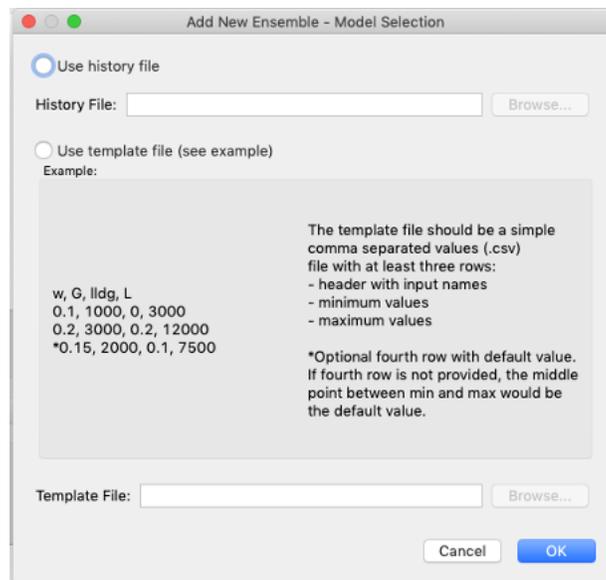
inputs to be considered. Leaving gaps or holes in the input space is possible, but generally should correspond to a region where it is not possible (or desirable) to collect data.

History .csv files should have the same number of columns for the input space as the candidate file (with matching column names) and represent data that have already been collected. The algorithm for creating the design aims to place points in different locations from where data have already been obtained, while filling the input space around those locations.

Both the **Candidate** and **History** files should be .csv files that have the first row as the Column heading. The Input columns should be numeric. Additional columns are allowed and can be identified as not necessary to the design creation at a later stage.

Generating a New Candidate Set

In the **Ensemble Selection** box, click on the **Add New...** button to select the file for the construction of the candidate set. The following menu will appear:



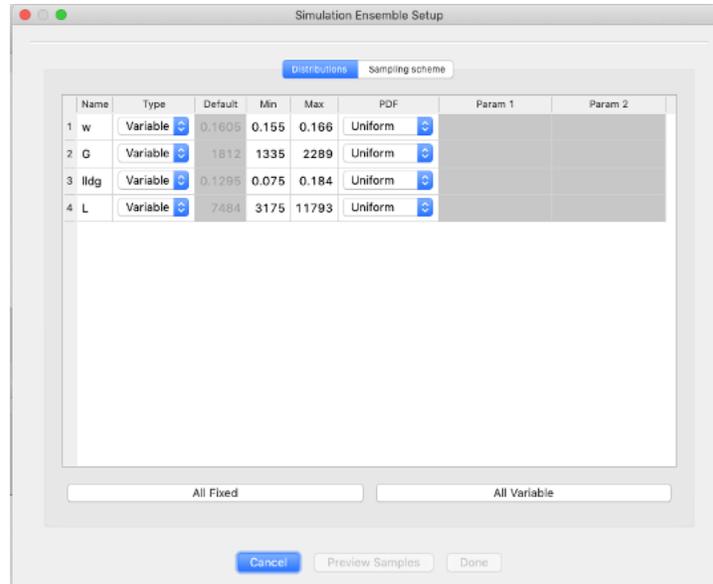
The user can select between two options: using a history file or a template file.

1. **History File.** An existing .csv file with historical data is required. If this option is selected, then the inputs to be used in the candidate set are extracted from the columns of the file.
2. **Template File.** The template file should be a simple comma separated values (.csv) file with at least three rows:
 - Header with input names
 - Minimum values
 - Maximum values

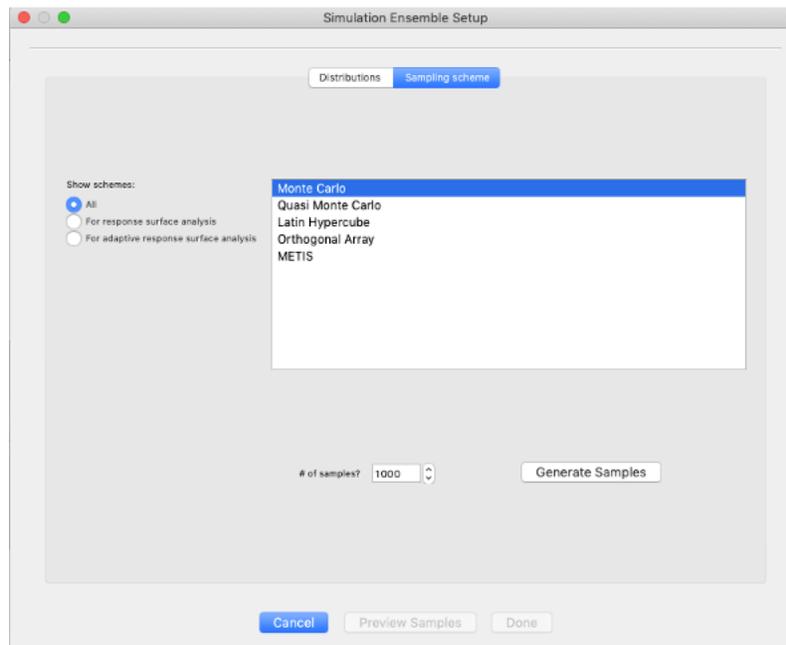
An optional fourth row with default value can be added. If fourth row is not provided, the middle point between min and max becomes the default value.

Note: Choosing History or Template file won't change the next steps.

Once the user has decided on which file to use, click on the OK button and the following dialog will pop up:

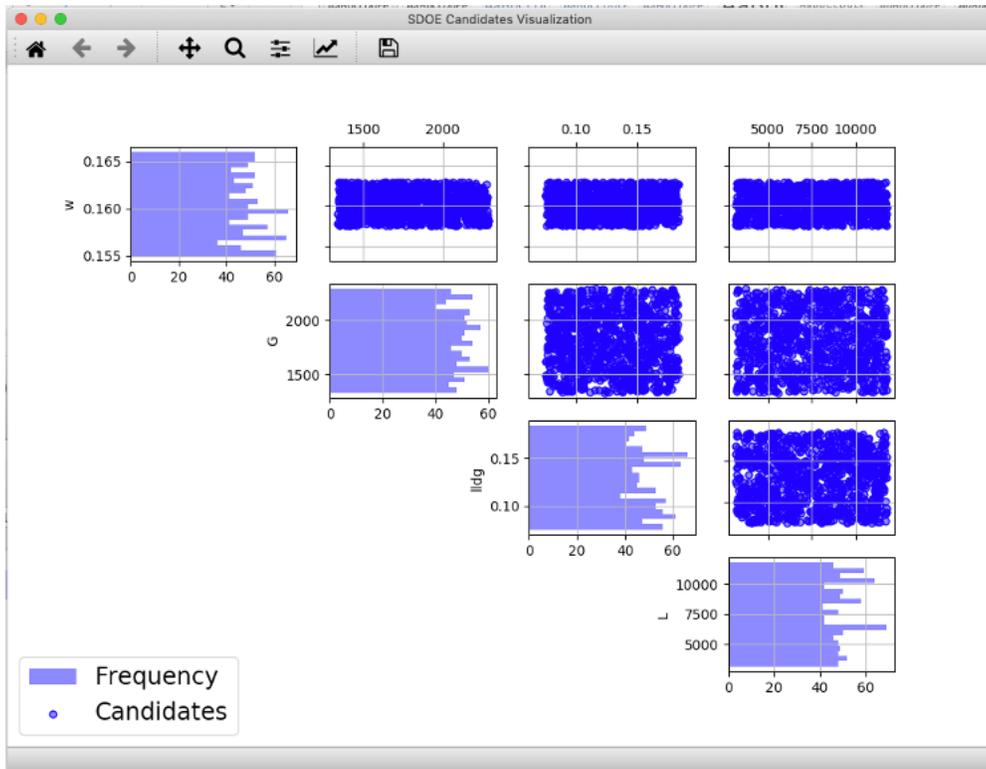
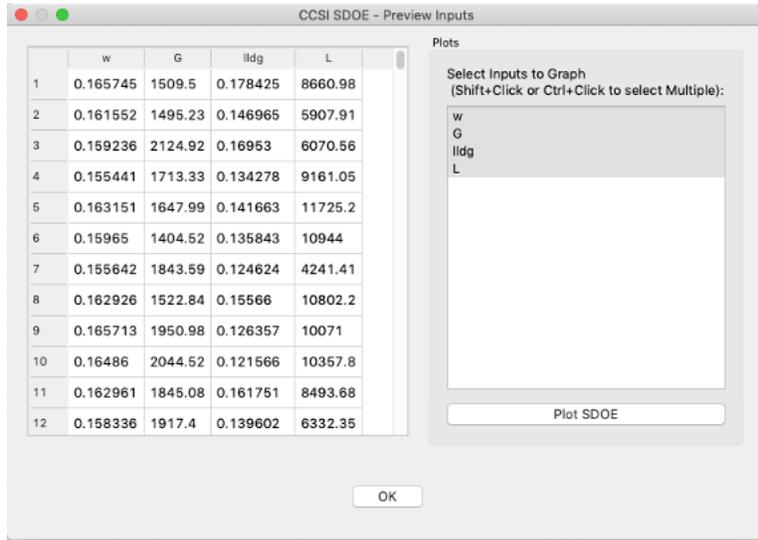


Decide on the type of input (variable or fixed) and the probability distribution function desired. If the input is set to variable, the user can modify minimum and maximum values to define the range over which sampling should be drawn. If input is set to fixed, then user selects the default value that will be used for all samples. Then click on the **Sampling Scheme** tab and you'll see the following menu:



Choose sampling scheme to be used. The Monte Carlo and Quasi Monte Carlo options provide candidates that are scattered arbitrarily throughout the input space, while Latin Hypercube, Orthogonal Array and METIS provide different approaches for structured distribution throughout the input space. Next, choose the number of samples you want to generate and click on **Generate Samples** button. The user can preview the samples by clicking on the **Preview Samples** button.

On the left-hand side table, you can explore the generated data for the different inputs and on the right-hand side list you can select which inputs you want to plot.



Once you are happy with the samples generated then click on **Done** button, so the new candidate set gets saved and populated in the **Ensemble Selection** box.

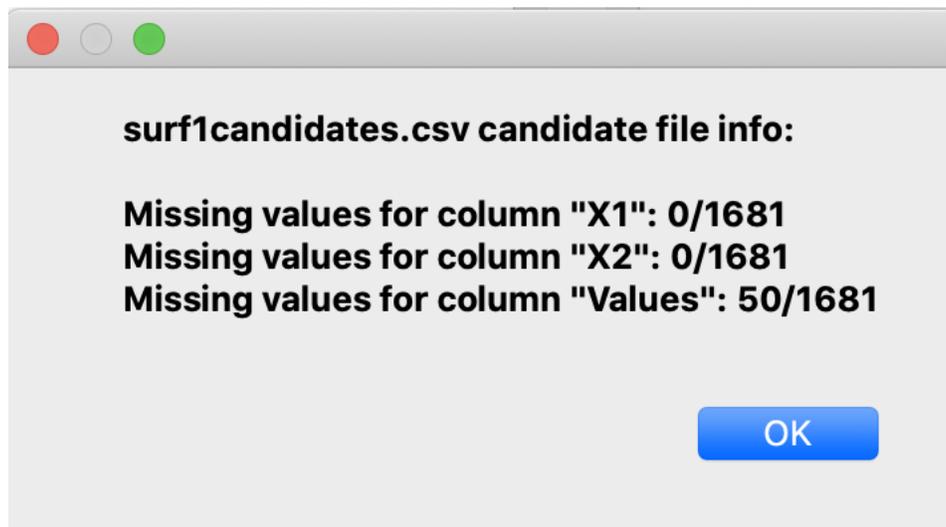
Using the Data Imputation Feature

There is a chance the user wants to use a candidate set to generate a design (NUSF or IRSF only) that has missing values either in the weights column (NUSF) or the response column (IRSF). The data imputation feature, uses the response surface capability integrated from the Uncertainty Quantification (UQ) module to impute those missing values and return the user a complete candidate set ready to use for the design creation.

The new complete candidate set is given as a separate file from the original set. It is very important that the original incomplete candidate set be deleted or deselected (unchecked) before continuing. If not, the new and old candidate sets will be combined, and will still contain missing values.

Note: To use this feature you need to install the latest version of PSUADE. For more details go to section *Install Optional Software*

When the user clicks on the **Load Existing Set** button in the **Design Setup** section and selects a candidate set, the program will run a quick file scan and show the missing values (if any) for all the existing columns.



We can see in this specific candidate set, there are 50 missing values for the **Values** column (that could be either our weight column in NUSF or our response column in IRSF). Click **OK** and the incomplete candidate set will get loaded in the **Design Setup** table. Notice there are new elements that become available in the candidate row (**Response Surface** and **Validate RS**).

The user needs to select a response surface from the ones available in the drop-down menu:

1. Polynomial, Linear
2. Polynomial, Quadratic
3. Polynomial, Cubic
4. MARS
5. Gaussian Process

For this particular example we will be using MARS. Once the response surface is selected, click the **Validate RS** button. A response surface validation plot will be generated and also an informative message window will pop up

FOQUS -- [not saved yet]

Session Flowsheet Uncertainty Optimization OOU SDoE Surrogates Settings Help

Sequential Design of Experiments

Space-filling DoE (SDoE)
 Robust optimality-based DoE (ODOE)

Design Setup

Generate New Candidate Set Load Existing Set Clone Selected Delete Selected Save Selected

Select	File Type	Visualize	File Name	Response Surface	(c)
1 <input checked="" type="checkbox"/>	Candidate	View	surffcandidates.csv	Polynomial ->	Linear

- If you have a candidate set:
 - Press Load Existing Set and upload. Set File Type to Candidate.
 - If you also have previous data, press Load Existing Set and upload. Set File Type to Previous Data.
- If you do not have a candidate set, press "Generate New Candidate Set"
- When finished identifying previous data and candidate sets, press Continue.

Continue

Design Construction

Candidate File	Descriptor	Visualize
Previous Experiments/Data		
Output Directory		
Design Method		

Back to Selection Open SDoE Dialog

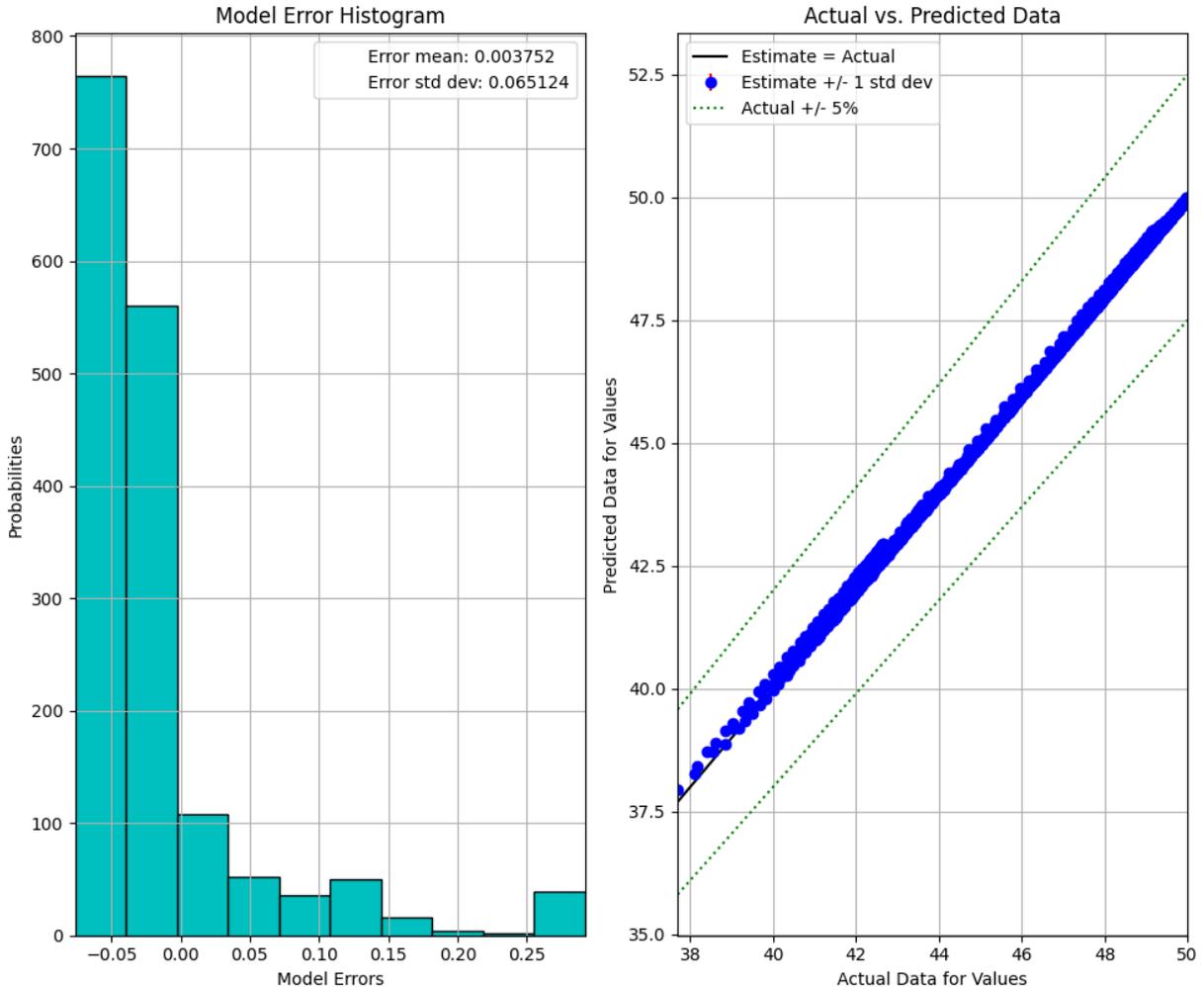
Inspection / Deletion / Output Value Modification Filtering

Select Variables (columns) and/or Sample Points (rows) for Deletion.
 Type new values for outputs in the appropriate cells.

Reset Table Perform Deletion then Save as New Ensemble

Sample #	Variables	X1	X2	Values
1	<input type="checkbox"/>	-1.00000	-1.00000	37.69600
2	<input type="checkbox"/>	-1.00000	-0.95000	38.16250
3	<input type="checkbox"/>	-1.00000	-0.90000	38.60600

telling the user to confirm if the response surface satisfy their needs or to select a different response surface and validate again if it doesn't.



The validation plot is a plot of expected values based on the model versus the actual values. The better the model fits, the closer the points will be around the diagonal line.

Since this particular validation plot looks good (note the points falling along the diagonal line), user can go ahead and confirm the response surface by clicking the **Confirm RS** button.

After clicking the **Confirm RS** button, the **Impute** button under the **Impute Data** column will become available. Click the **Impute** button to impute the missing values.

When the program is done imputing the missing values, it will populate the **Design Setup** table with a new candidate set that will be an exact copy of the previous candidate set but with the missing values filled up.

If the user clicks on the **View** button under the **Visualize** column in the **Design Setup** table, the **Preview Inputs** dialog will show up. Click on the **Plot SDoE** button and you will be able to see the original and the imputed values (in red).

If the user is happy with the updated and complete candidate set, they can go ahead and perform business as usual. Just remember to either delete the incomplete candidate set using the **Delete Selected** button or simply uncheck the checkbox under the **Select** column in the **Design Setup** table.

FOQUS -- [not saved yet]

Session Flowsheet Uncertainty Optimization OOU SDoE Surrogates Settings Help

Sequential Design of Experiments

Space-filling DoE (SDoE)
 Robust optimality-based DoE (ODOE)

Design Setup

Name	Response Surface	(cont'd)	Validate RS	Confirm RS	Impute Data
lates.csv	MARS ->	MARS	Validate RS	Confirm RS	Impute

Design Construction

Candidate File	Descriptor	Visualize
Previous Experiments/Data		
Output Directory		
Design Method		

Select Variables (columns) and/or Sample Points (rows) for Deletion.
Type new values for outputs in the appropriate cells.

Sample #	Variables	X1	X2	Values
1	<input type="checkbox"/>	-1.00000	-1.00000	37.69600
2	<input type="checkbox"/>	-1.00000	-0.95000	38.16250
3	<input type="checkbox"/>	-1.00000	-0.90000	38.60600

FOQUS -- [not saved yet]

Session Flowsheet Uncertainty Optimization OOU SDoE Surrogates Settings Help

Sequential Design of Experiments

Space-filling DoE (SDoE)
 Robust optimality-based DoE (ODoE)

Design Setup

me	Response Surface	(cont'd)	Validate RS	Confirm RS	Impute Data
ates.csv	Polynomial ->	Linear	Validate RS	Confirm RS	Impute

• If you have a candidate set:
 1. Press Load Existing Set and upload. Set File Type to Candidate.
 2. If you also have previous data, press Load Existing Set and upload. Set File Type to Previous Data.

• If you do not have a candidate set, press "Generate New Candidate Set"
 • When finished identifying previous data and candidate sets, press Continue.

Design Construction

Candidate File	Descriptor	Visualize
Previous Experiments/Data		
Output Directory		
Design Method		

Inspection / Deletion / Output Value Modification Filtering

Select Variables (columns) and/or Sample Points (rows) for Deletion.
Type new values for outputs in the appropriate cells.

Sample #	Variables	X1	X2	Values
1	<input type="checkbox"/>	-1.00000	-1.00000	37.69600
2	<input type="checkbox"/>	-1.00000	-0.95000	38.16250
3	<input type="checkbox"/>	-1.00000	-0.90000	38.60600

FOQUS -- [not saved yet]

Session Flowsheet Uncertainty Optimization OOU SDoE Surrogates Settings Help

Sequential Design of Experiments

Space-filling DoE (SDoE)
 Robust optimality-based DoE (ODOE)

Design Setup

Select	File Type	Visualize	File Name	Response Surface
1 <input checked="" type="checkbox"/>	Candidate	View	surf1candidates.csv	Polynomial ->
2 <input checked="" type="checkbox"/>	Candidate	View	surf1candidates_linear_imputed.csv	Polynomial ->

- If you have a candidate set:
 - Press Load Existing Set and upload. Set File Type to Candidate.
 - If you also have previous data, press Load Existing Set and upload. Set File Type to Previous Data.
- If you do not have a candidate set, press "Generate New Candidate Set"
- When finished identifying previous data and candidate sets, press Continue.

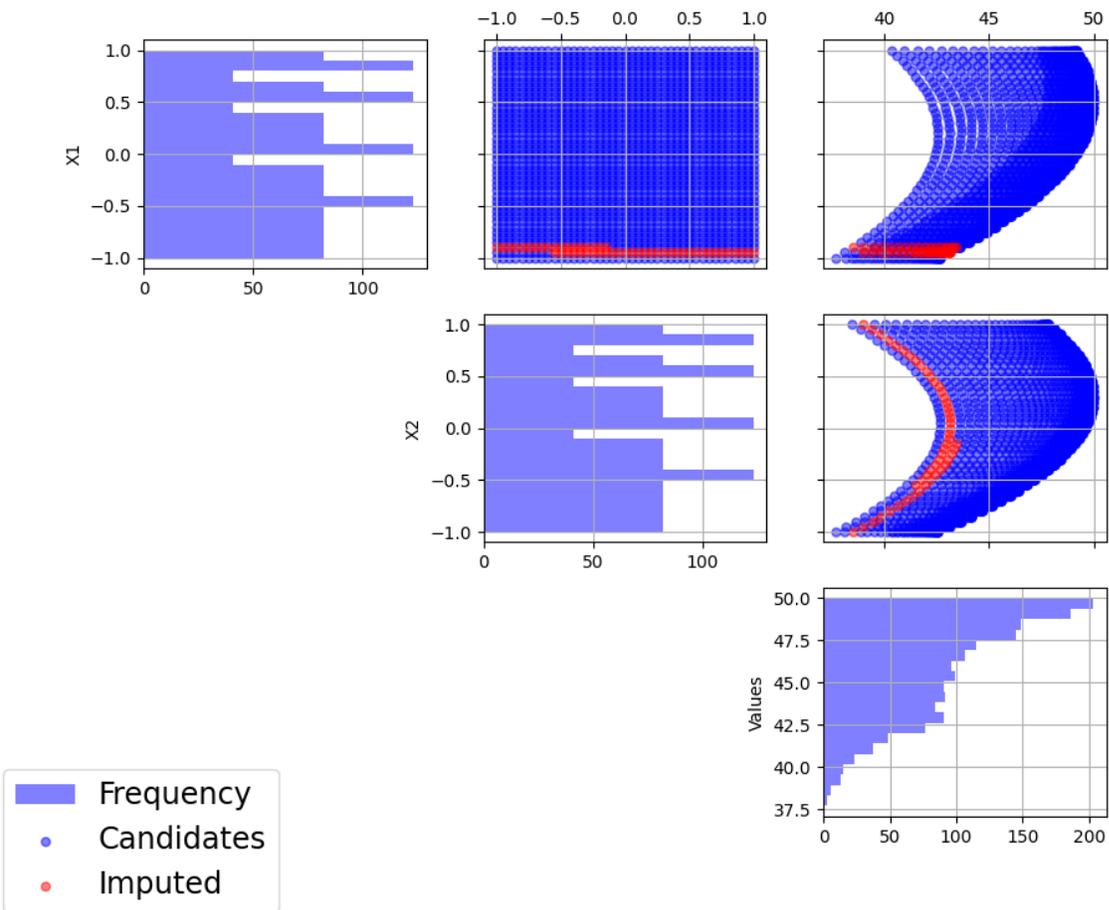
Design Construction

	Descriptor	Visualize
Candidate File		
Previous Experiments/Data		
Output Directory		
Design Method		

Inspection / Deletion / Output Value Modification Filtering

Select Variables (columns) and/or Sample Points (rows) for Deletion.
Type new values for outputs in the appropriate cells.

Sample #	Variables	X1	X2	Values
1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	<input type="checkbox"/>	-1.00000	-1.00000	37.69600
2	<input type="checkbox"/>	-1.00000	-0.95000	38.16250
3	<input type="checkbox"/>	-1.00000	-0.90000	38.60600



Basic Steps for a Uniform Space Design

We now consider some details for each of these steps:

1. In the **Design Setup** box, click on the **Load Existing Set** button to select the file(s) for the construction of the design. Several files can be selected and added to the box listing the chosen files.

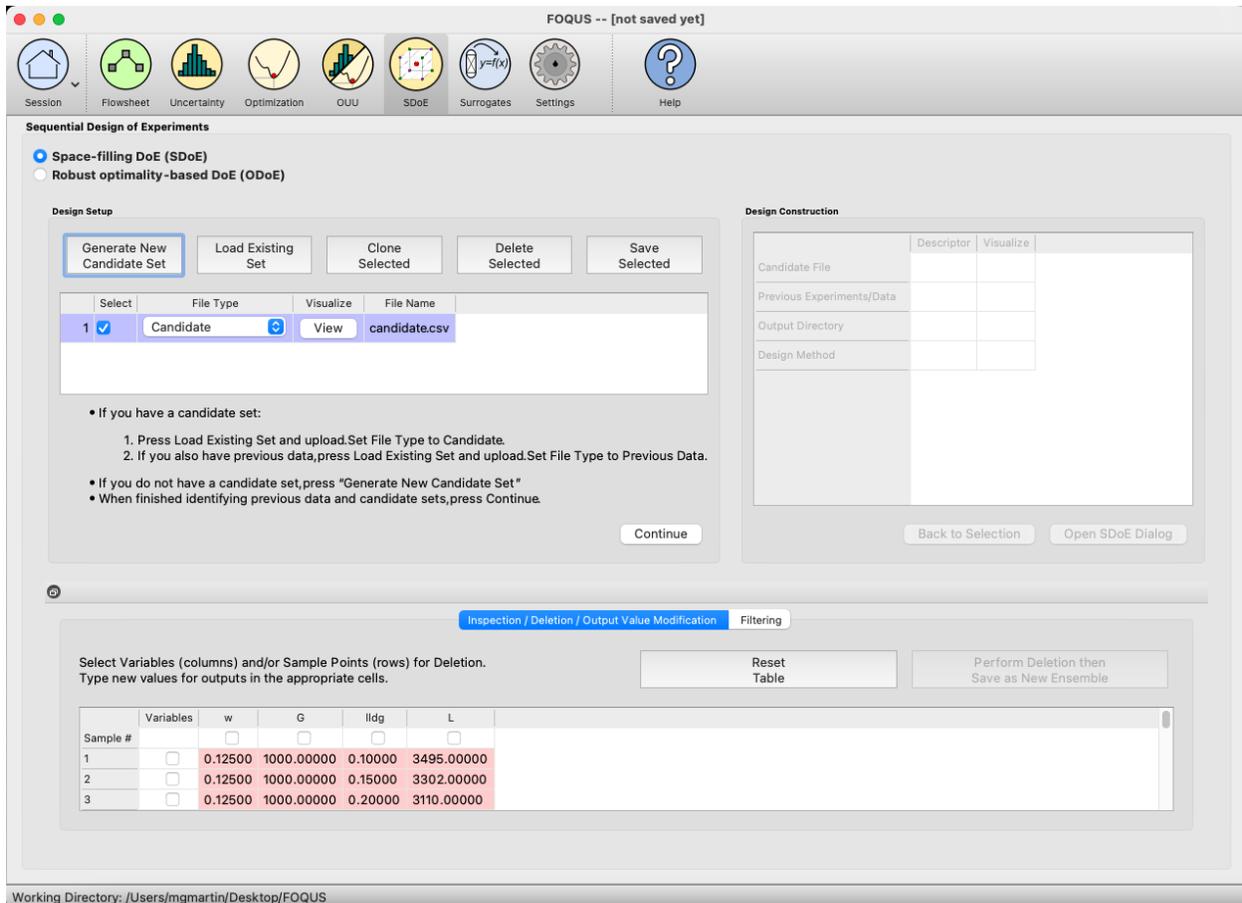


Fig. 9: SDOE Home Screen

2. For each of the files selected using the pull-down menu, identify them as either a **Candidate** file or a **Previous Data** file. **Candidate** .csv files are comprised of possible input combinations from which the design can be constructed. The columns of the file should contain the different input factors that define the dimensions of the input space. The rows of the file each identify one combination of input values that could be selected as a run in the final design. Typically, a good candidate file will have many different candidate runs listed, and they should fill the available design region to be considered. Leaving gaps or holes in the input space is possible, but generally should correspond to a region where it is not possible (or desirable) to collect data. The flexibility of the candidate set approach allows for linear and non-linear constraints for one or more of the inputs to be incorporated easily.

Previous Data .csv files should have the same number of columns for the input space as the candidate file (with matching column names), and represent data that have already been collected. The algorithm for creating the design aims to place points separated from where data have already been obtained, while filling the input space around those locations. If the experiment is being run sequentially, the Previous Data file should use the input values that were actually implemented, not the target values from the previous designed experiment.

Both the **Candidate** and **Previous Data** files should be .csv files that have the first row as the Column heading. The Input columns should be numeric. Additional columns are allowed and can be identified as not necessary to the

design creation at a later stage.

- Click on the **View** button to open the **Preview Inputs** pop-up window, to see the list of columns contained in each file. The left hand side displays the first few rows of input combinations from the file. Select the columns that you wish to see graphically in the right hand box, and then click on **Plot SDOE** to see a scatterplot matrix of the data.

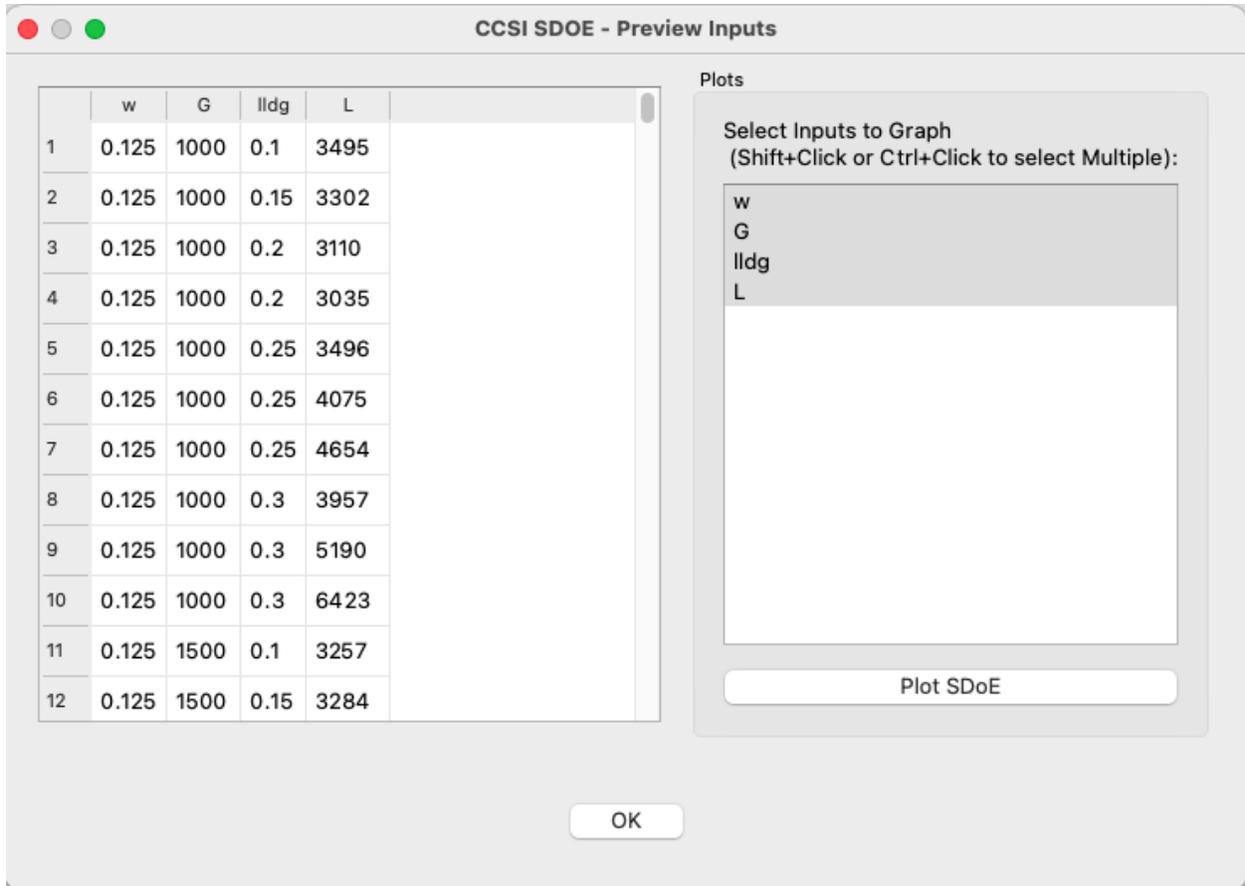


Fig. 10: SDOE view candidate set inputs

The plot shows histograms of each of the inputs on the diagonals to provide a view of the distribution of values as well as the range of each input. The off-diagonals show pairwise scatterplots of each pair of inputs. This should provide the experimenter with the ability to assess if the ranges specified and any constraints for the inputs have been appropriately captured by the specified candidate set. In addition, repeating this process for any previous data will provide verification that the already observed data have been suitably summarized. Candidate set values are shown in gray, while previous data, if provided, is shown in pink.

- Once the data have been verified for both the **Candidate** and **Previous Data** files (if a Previous Data file has been included), click on the **Continue** button to make the **Design Construction** window active.
- If more than one **Candidate** file was specified, then the **aggregate_candidates.csv** file that was created will have combined these files into a single file. Similarly if more than one **Previous Data** file was specified, then the **aggregate_previousData.csv** file will have been created with all runs from all these files. If only a single file was selected for either the **Candidate** and **Previous Data** files, then their corresponding aggregated files will be the same as the original.

Note: There are options to view the aggregated files for both the candidate and previous data files, simply scroll to the right of the Design Construction window, with a similar interface as was shown in step 3. In addition, a single plot

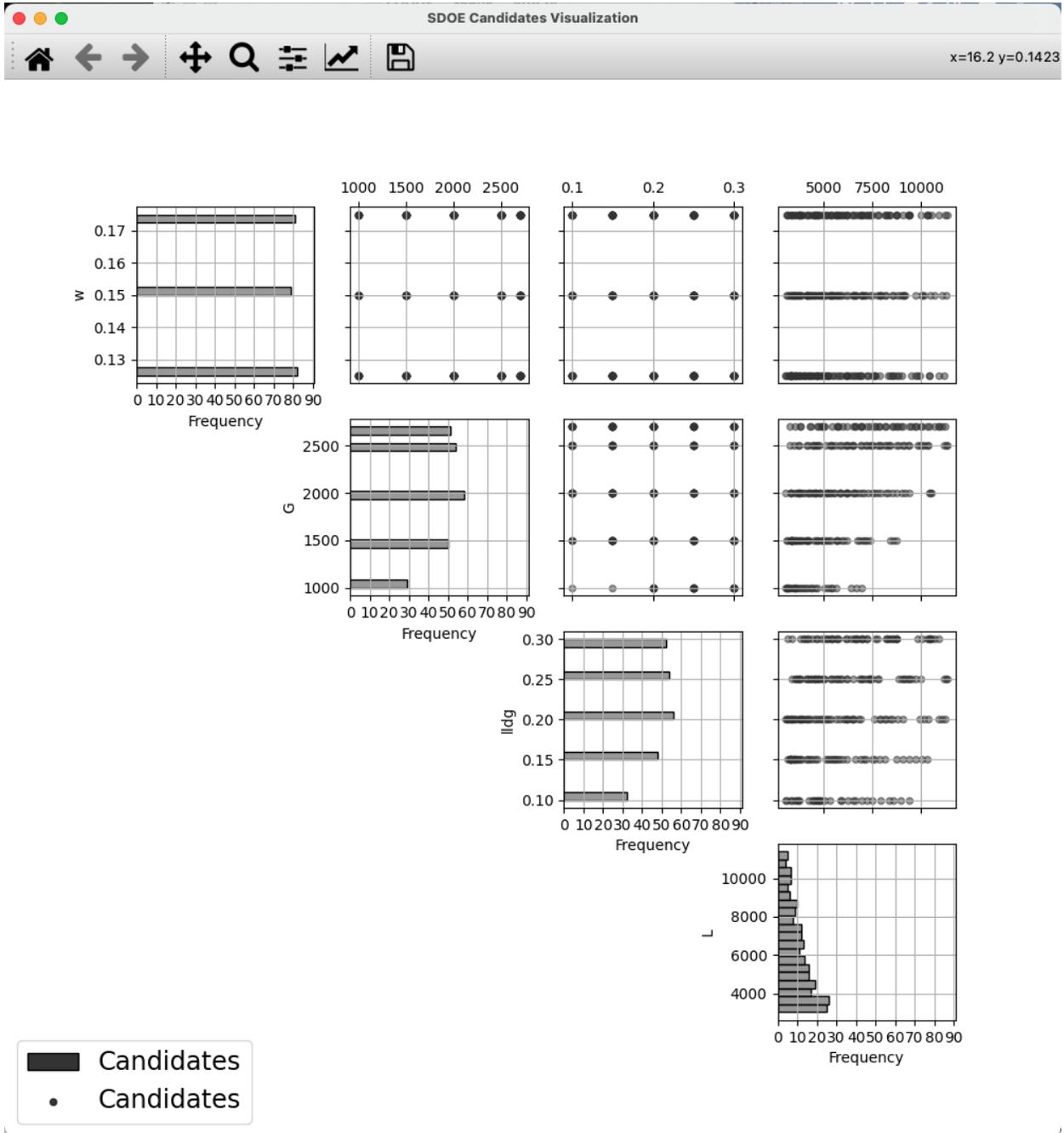


Fig. 11: SDOE plot of candidate set inputs

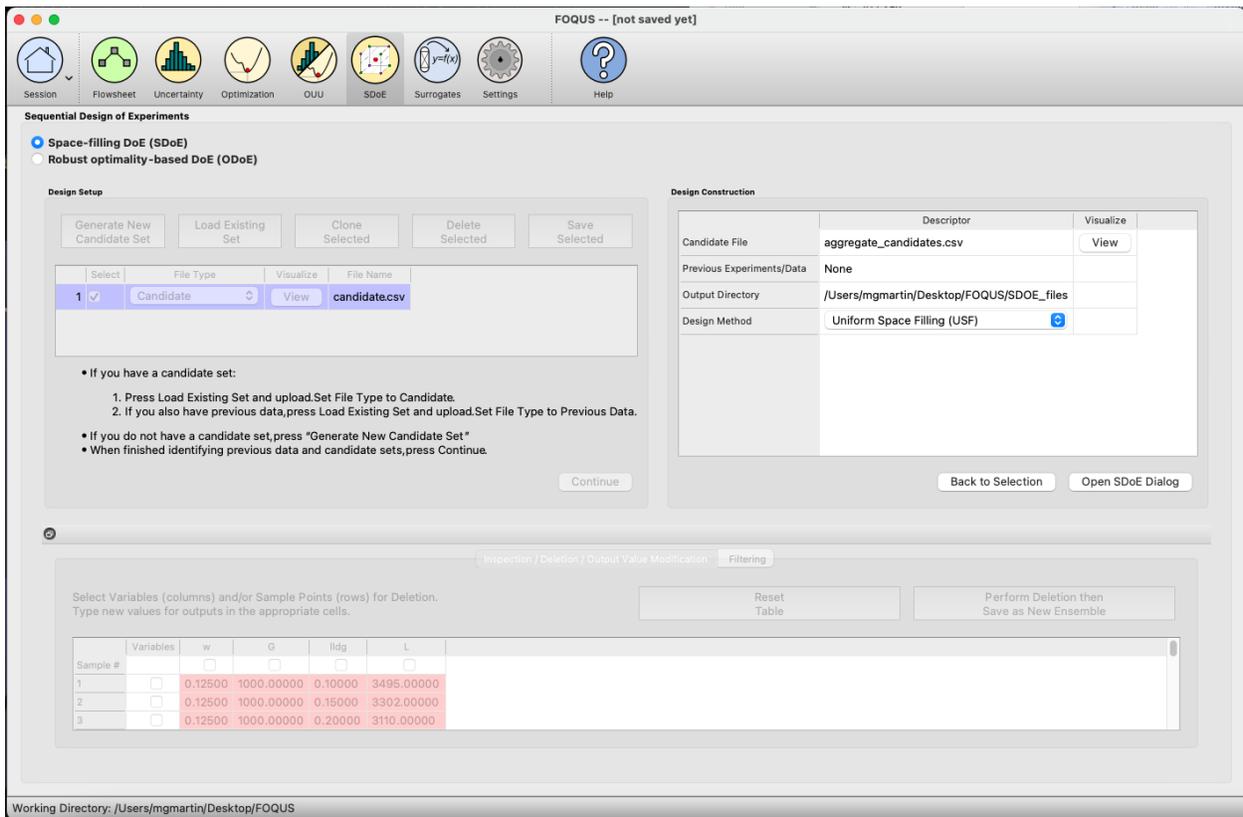


Fig. 12: SDOE Design Construction

of the combined candidate and previous data files can be viewed. In this plot the points representing the candidate locations and points of already collected data from the previous data file are shown in different colors.

6. Once the data have been verified as the desired set to be used for the design construction, then click on the **Uniform Space Filling** button at the bottom right corner of the **Design Construction** window, then select **Open SDoE Dialog**. This opens the second SDoE window, which allows for specific design choices to be made.

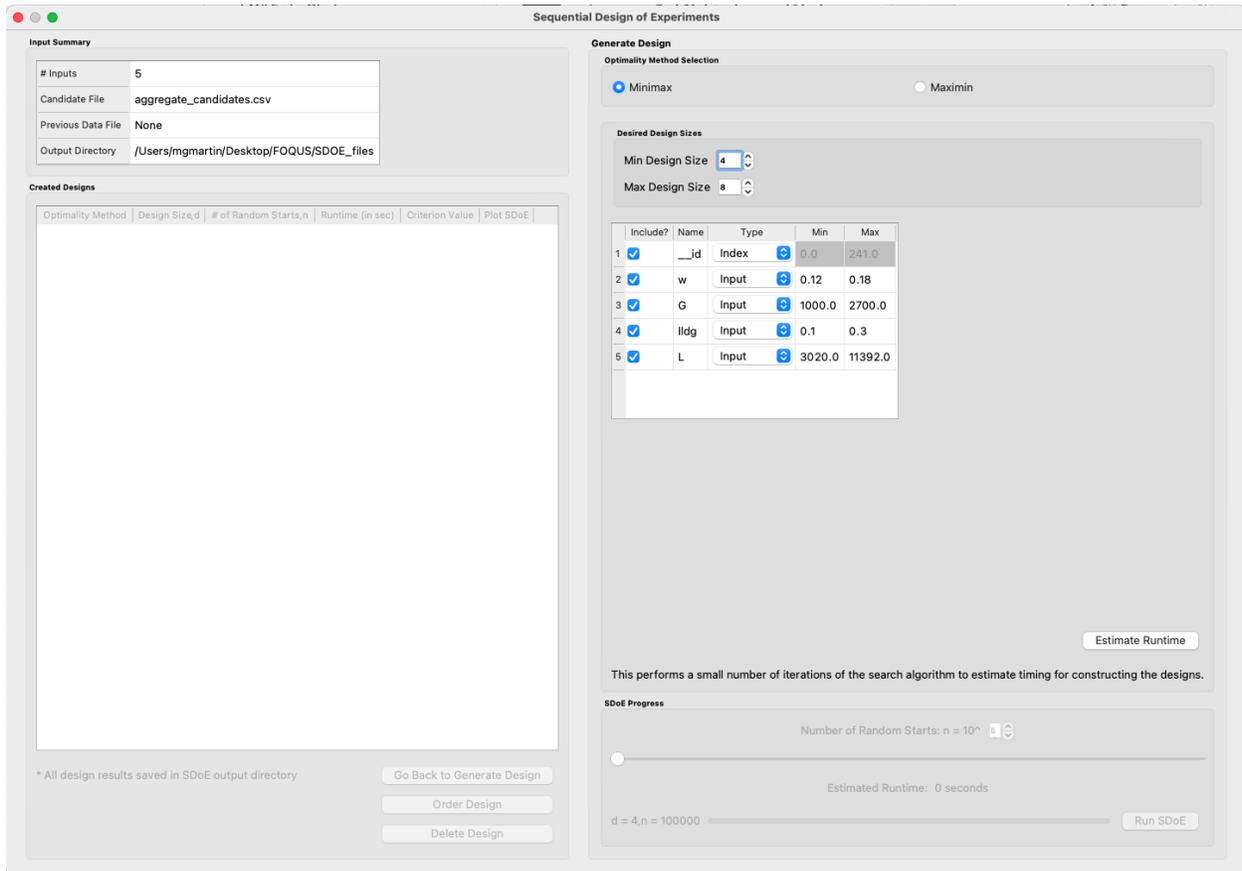


Fig. 13: SDOE second window

7. The first choice to be made for the design is whether to optimize using **minimax** or **maximin**. The first choice, **minimax**, looks to choose design points that minimize the maximum distance that any point in the input space (as characterized by the candidate set and previous data, if it is available) is away from a design point. Hence, the idea here is that if we want to use data to help predict new outcomes throughout the input space, then we never want to be too far away from a location where data was observed.

The second choice, **maximin** looks to choose a design where the design points are as far away from each other as possible. In this case, the design criterion is looking to maximize how close any point is from their nearest neighbor. In practice the two design criterion often give similar designs, with the **maximin** criterion tending to push the chosen design points closer to the edges of the specified regions.

Hint: If there is uncertainty about some of the edge points in the candidate set being viable options, then **minimax** would be preferred. If the goal is to place points throughout the input space with them going right to the edges, then **maximin** would be preferred. Note, that creating the designs is relatively easy, so we recommend trying both approaches to examine them and then choose which is preferred based on the summary plots that are provide later.

8. The next choice falls under **Desired Design Size**, where the experimenter can select the sizes of designs to be created. The **Min Design Size** specifies the smallest design size to be created. Note that the default value is set at **2**, which would lead to choosing the best two design runs from the candidate set to fill the space (after taking into account any previous data that have already been gathered).

The **Max Design Size** specifies the largest design size to be created. The default value is set at **8**, which means that if this combination were used, designs would be created of size 2, 3, 4, 5, 6, 7 and 8. The number of integers between **Min Design Size** and **Max Design Size** determines the total number of searches that the SDoE algorithm will perform. Hence, it is prudent to make a thoughtful choice for this range, that balances design sizes that are potentially of interest with the waiting time for the designs to be created. In the figure above, the **Min Design Size** has been changed to 4, so that only the designs of size 4, 5, 6, 7 and 8 will be created.

9. Next, there are options for the columns of the candidate set to be used for the construction of the design. Under **Include?** in the box on the right hand side, the experimenter has the option of whether particular columns should be included in the space-filling design search. Uncheck a box, if a particular column should not be included in the space filling criterion search.

Next select the **Type** for each column. Typically most of the columns will be designated as **Inputs**, which means that they will be used to construct the best uniform space filling design. In addition, we recommend including one **Index** column which contains a unique identifier for each run of the candidate set. This makes it easier to track which runs are included in the constructed designs. If no **Index** column is specified, a warning appears later in the process, but this column is not strictly required.

Notice there is a new variable included in the first row of this box called **__id**. This column is an automatically-generated index of all rows of the candidate set, meaning the column counts up from 1, uniquely identifying each row. For example, if the candidate set contains 50 rows excluding the row of column names, the **__id** column would be 1, 2, 3, ..., 49, 50. The **Include** box next to **__id** can be unchecked if including this index column is not desired, but again, it is highly encouraged to have an index column to easily identify which candidate set rows are chosen in the design. The **__id** column **Type** is automatically set to **Index**. If using a different variable as the index column, make sure to uncheck the **Include** box next to **__id** and also change the **Type** of the desired index column to **Index**.

Finally, the **Min** and **Max** columns in the box allow the range of values for each input column to be specified. The default is to extract the smallest and largest values from the candidate and previous data files, and use these as the **Min** and **Max** values, respectively. This approach generally works well, as it scales the inputs to be in a uniform hypercube for comparing distances between the design points.

Hint: The default values for **Min** and **Max** can generally be left at their defaults unless: (1) the range of some inputs represent very different amounts of change in the process. For example, if temperature is held nearly constant, while a flow rate changes substantially, then it may be desirable to extend the range of the temperature beyond its nominal values to make the amount of change in temperature more commensurate with the amount of change in the flow rate. This is a helpful strategy to make the calculated Euclidean distance between any points a more accurate reflection of how much of an adjustment each input requires. (2) if changes are made in the candidate or previous data files. For example, if one set of designs are created from one candidate set, and then another set of designs are created from a different candidate set. These designs and the achieved criterion value will not be comparable unless the range of each input has been fixed at matching values.

10. Once the design choices have been made, click on the **Estimate Runtime** button. This performs a small number of iterations of the search algorithm to calibrate the timing for constructing and evaluating the designs. The time taken to generate a design is a function of the size of the candidate set, the size of the design, as well as the dimension of the input space. The slider below **Estimate Runtime** now indicates an estimate of the time to construct all of the designs across the range of the **Min Design Size** and **Max Design Size** specified. The smallest **Number of Random Starts** is $10^3 = 1000$, and is generally too small to produce a good design, but this will run very quickly and so might be useful for a demonstration. However, it would generally be unwise to use a design generated from this small a set of random starts for an actual experiment. Powers of 10 can be chosen with an **Estimated Runtime** provided below the slider.

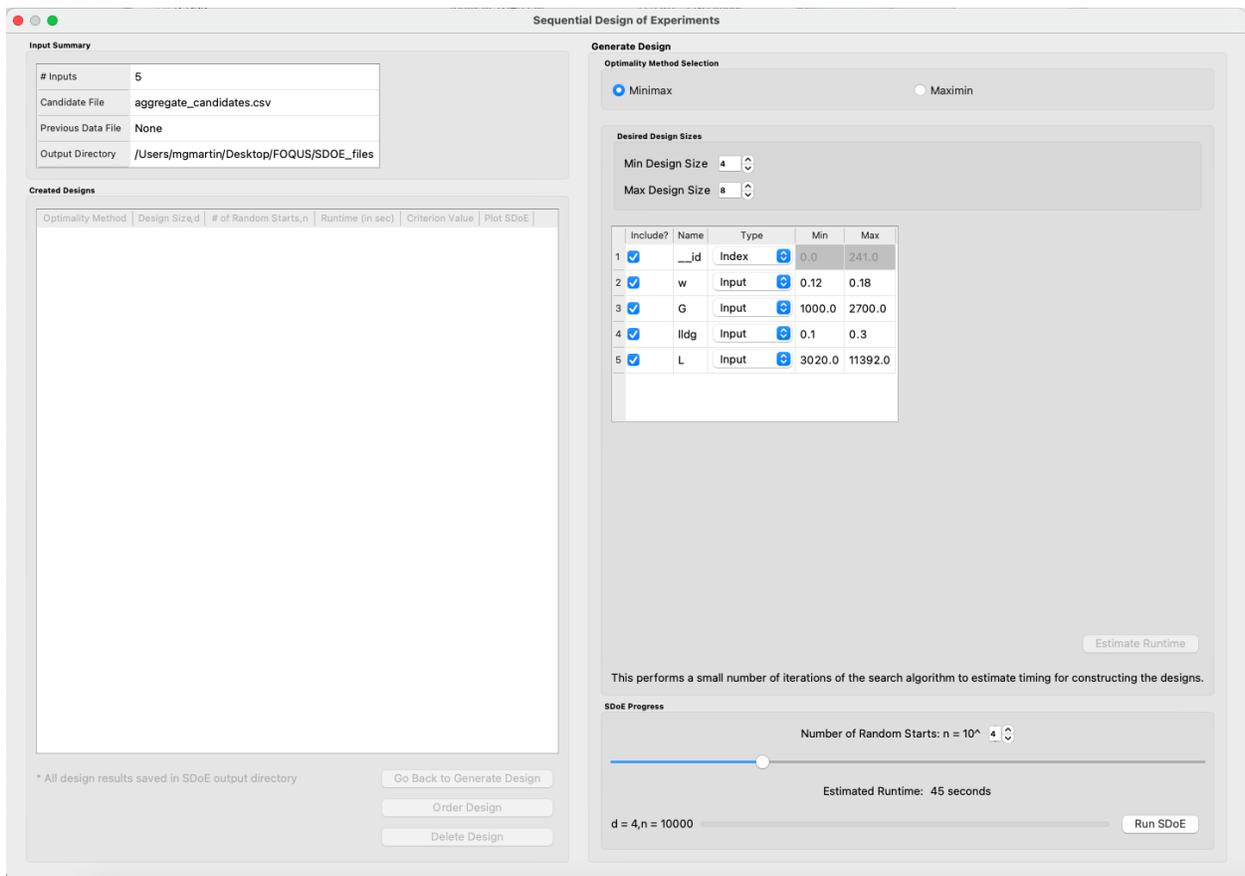


Fig. 14: SDOE second window after clicking Estimate Runtime

Hint: The choice of **Number of Random Starts** involves a trade-off between the quality of the design generated and the time spent waiting to generate the design. The larger the chosen number of random starts, the better the design is likely to be. However, there are diminishing gains for increasingly large numbers of random starts. If running the actual experiment is expensive, it is generally recommended to choose as large a number of random starts as possible for the available time frame, to maximize the quality of the constructed design.

11. Once the slider has been set to the desired **Number of Random Starts**, click on the **Run SDOE** button, and initiate the construction of the designs. The progress bar indicates how design construction is advancing through the chosen range of designs between the specified **Min Design Size** and **Max Design Size** values.

12. When the SDOE module has completed the design creation process, the left window **Created Designs** will be populated with files containing the results. The column entries summarize the key features of each of the designs, including **Optimality Method** (whether minimax or maximin was selected), **Design Size** (d, the number of runs in the created design), **# of Random Starts**, **Runtime** (number of seconds needed to create the design), **Criterion Value** (the value obtained for the minimax or maximin criterion for the saved design).

The screenshot shows the 'Sequential Design of Experiments' software interface. On the left, the 'Input Summary' panel shows: # Inputs: 5, Candidate File: aggregate_candidates.csv, Previous Data File: None, Output Directory: /Users/mgmartin/Desktop/FOQUS/SDOE_files. Below this is the 'Created Designs' table:

	Optimality Method	Design Size,d	# of Random Starts,n	Runtime (in sec)	Criterion Value	Plot SDOE
1	minimax	4	10000	8.52	0.04	View
2	minimax	5	10000	8.69	0.08	View
3	minimax	6	10000	8.84	0.08	View
4	minimax	7	10000	8.91	0.08	View
5	minimax	8	10000	9.0	0.09	View

On the right, the 'Generate Design' panel shows 'Optimality Method Selection' with 'Minimax' selected. Below are 'Desired Design Sizes' sliders for Min Design Size and Max Design Size. A table lists input variables:

Include?	Name	Type	Min	Max
1	id	Index	0.0	241.0
2	w	Input	0.12	0.18
3	G	Input	1000.0	2700.0
4	lldg	Input	0.1	0.3
5	L	Input	3020.0	11392.0

At the bottom, the 'SDoE Progress' section shows a slider for 'Number of Random Starts: n = 10^4' and 'Estimated Runtime: 46 seconds'. A 'Run SDOE' button is visible.

Fig. 15: SDOE Created Designs

13. To see details of the design, the **View** button at the right hand side of each design row can be selected to show a table of the design, as well as a pairwise scatterplot of any subset of the input columns for the chosen design. The table and plot of the design are similar in characteristics to their counterparts described above for the candidate set. Candidate points and previous data are still shown in gray and pink, respectively, while the newly selected design points are shown in blue.

14. To access the file with the generated design, go to the **SDOE_files** folder, and a separate folder will have been

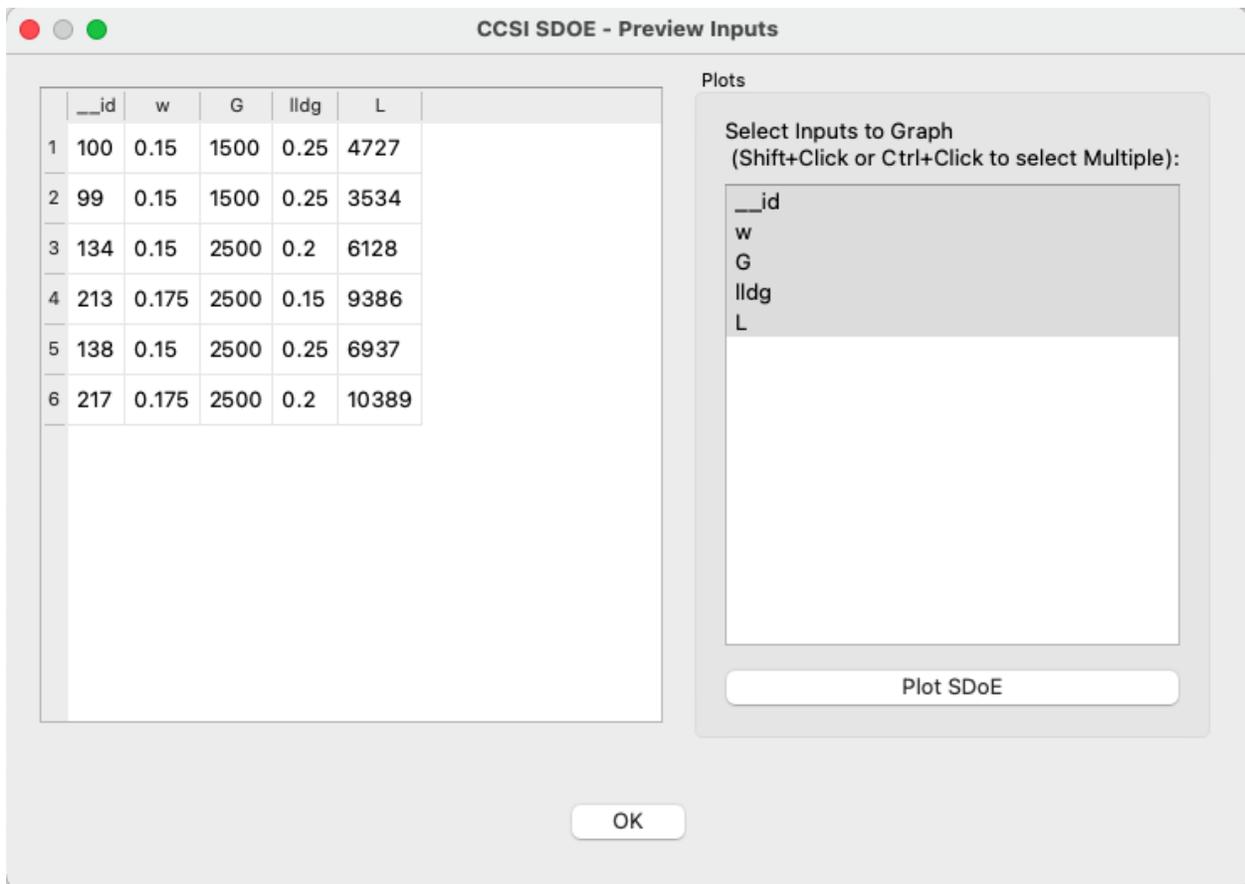


Fig. 16: SDOE table of created design

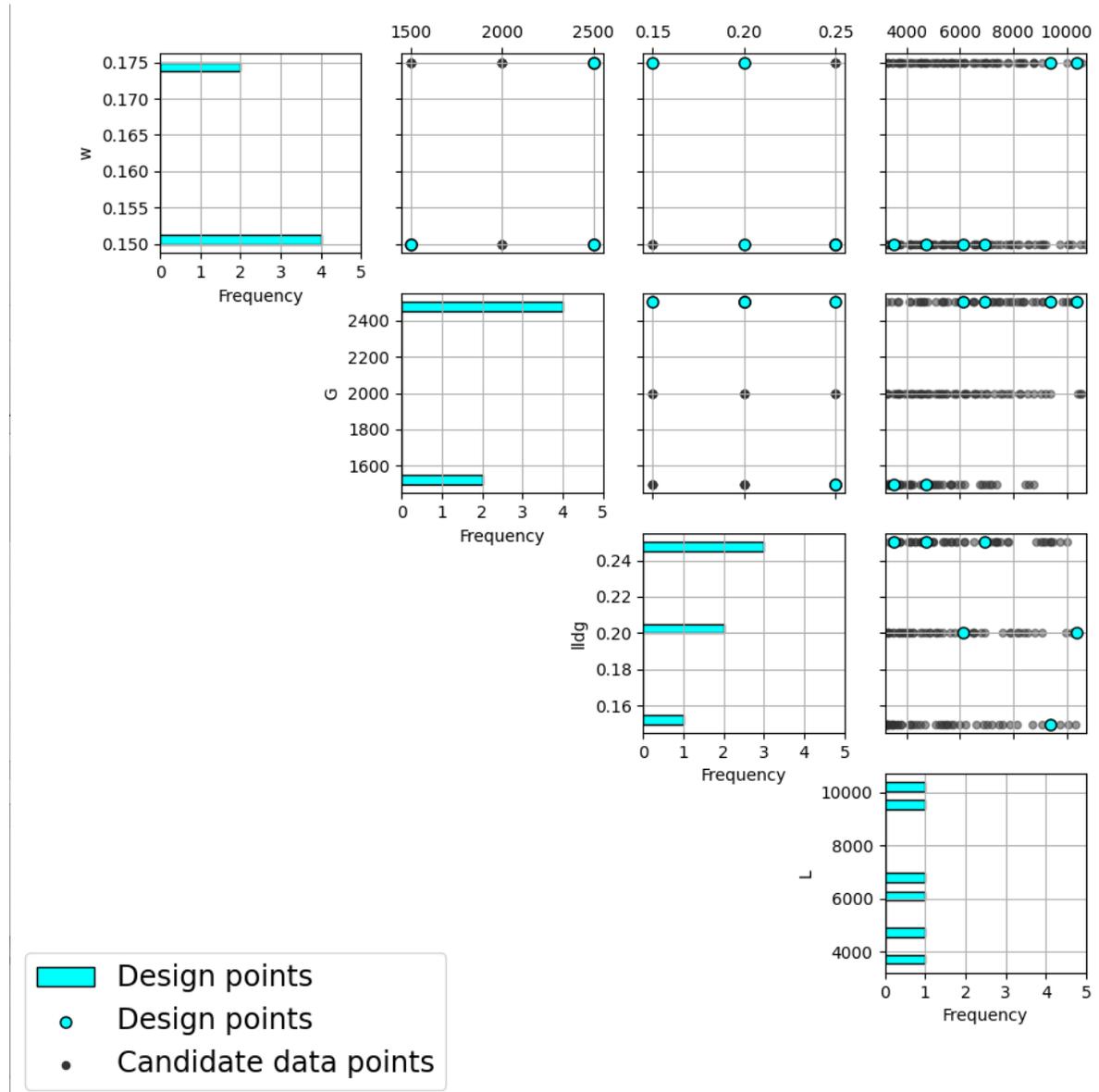


Fig. 17: SDOE pairwise plot of created design

created for each of the designs. In the example shown, 5 folders were created for the designs of size 4, 5, 6, 7 and 8, respectively. In each folder, there is a file containing the design, with a name that summarizes some of the key information about the design. For example, `candidates_d6_n10000_w+G+lldg+L` contains the design created using the candidate set called `candidates.csv`, with `d=6` runs, based on `n=10000` random starts, and based on the 4 inputs `W`, `G`, `lldg` and `L`.

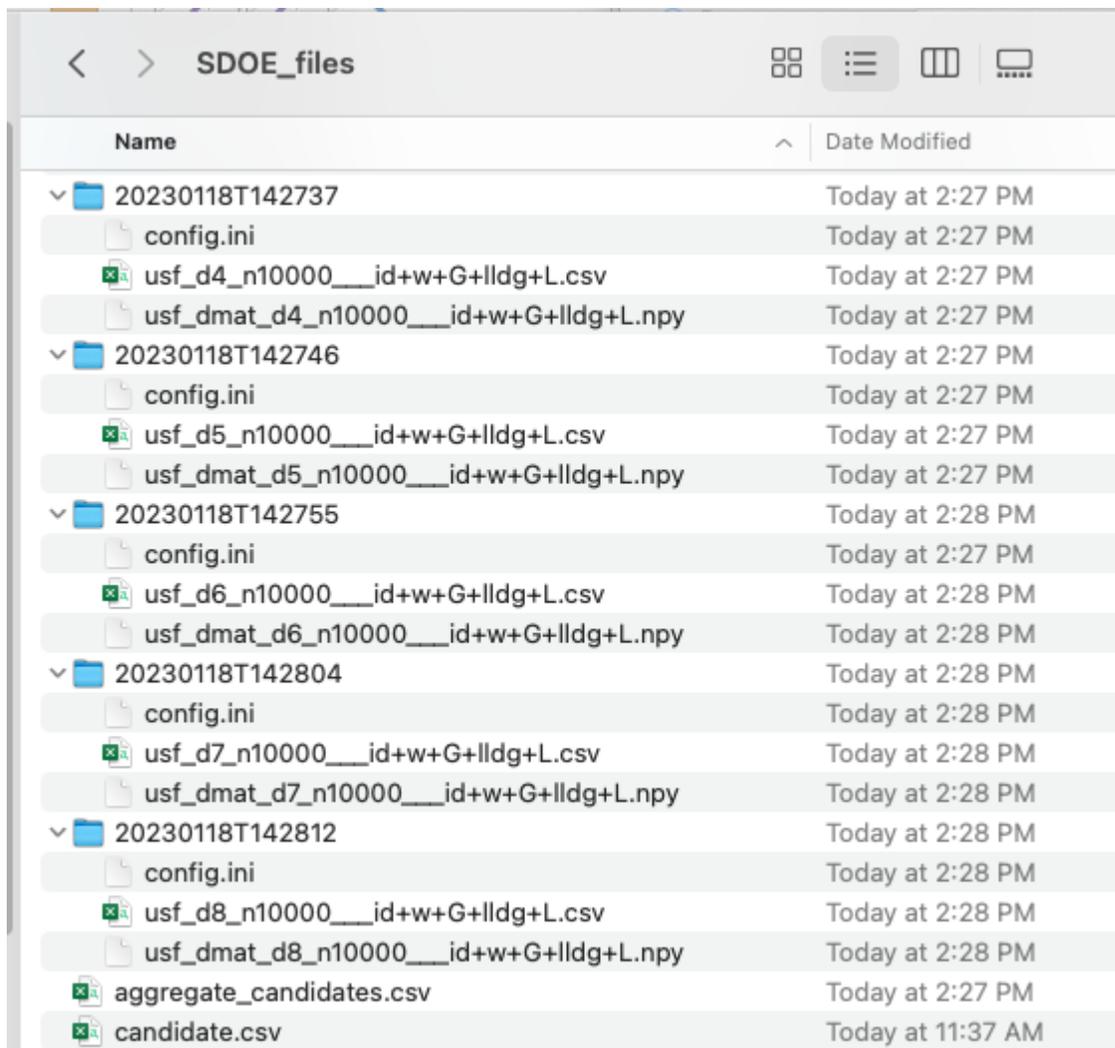


Fig. 18: SDOE directory

When one of the design files is opened it contains the details of each of the runs in the design, with the input factor levels that should be selected for that run. If an index column was included in the design, the index value will also be shown.

To evaluate the designs that have been created, it is helpful to look at a number of summaries, including the criteria values and visualizing the spread of the design points throughout the region. Recall that at the beginning of the design creation process we recommended constructing multiple designs, with different design sizes. By examining multiple designs, it is easier to determine which design is best suited to the requirements of the experiment.

In the **Created Designs** table, it is possible to see the criterion values for each of the designs. For **minimax** designs, the goal is to minimize how far away any point in the candidate set is away from a design point. Hence, smaller values of this criterion are better. It should be the case, that a larger design size will result in smaller values, as there are more design points to distribute throughout the input space, and hence any location should have a design point closer

The screenshot shows a Microsoft Excel spreadsheet with the following data:

	A	B	C	D	E	F
1	__id	w	G	lldg	L	
2	100	0.15	1500	0.25	4727	
3	99	0.15	1500	0.25	3534	
4	134	0.15	2500	0.2	6128	
5	213	0.175	2500	0.15	9386	
6	138	0.15	2500	0.25	6937	
7	217	0.175	2500	0.2	10389	
8						
9						
10						
11						
12						

Fig. 19: SDOE file containing a created design

to it. When evaluating between different sizes of design, it is helpful to think whether the improvement in the design criterion justifies the additional budget from a larger design.

For **maximin** designs, the goal is to maximize the distance between nearest neighbors for all design points. So for designs of the same size, we want the distance between neighboring points to be as large as possible, as this means that we have achieved near equal spacing of the design points. However, when we are comparing designs of different sizes, then the maximin criterion can be a bit confusing. Adding more runs to the design will mean that nearest neighbors will need to get closer together, and hence we would expect that on average the criterion value would get smaller for larger experiments. As with the minimax designs, we want to evaluate whether the closer packing of the design points from a larger experiment is worth the increase in cost for the additional runs.

Hint: Note that the criterion values for **minimax** and **maximin** should not be compared - one is comparing distances between design points and the candidate points, while the other is comparing distances between different design points.

For all of the designs, it is important to use the **View** option to look at scatterplots of the chosen design. When **Previous Data** points have been incorporated into the design, the plots will show how the overall collection of points fills the input space. When examining the scatterplots, it is important to assess (a) how close the design points have been placed to the edges of the region?, (b) are there holes in the design space that are unacceptably large?, and (c) does a larger design show a worthwhile improvement in the density of points to justify the additional expense?

Based on the comparison of the criterion values and the visualization of the spread of the points, the best design can be chosen that balances design performance with an appropriate use of the available budget. Recall that with sequential design of experiments, runs that are not used in the early stages might provide the opportunity for more runs at later stages. So the entire sequence of experimental runs should be considered when making choices about each stage.

Basic Steps for a Non-Uniform Space Design

We now consider some details for each of these steps for the second type of design, where we want to have different densities of design points throughout the chosen input region:

1. In the **Design Setup** box, click on the **Load Existing Set** button to select the file(s) to be used for the construction of the design. Several files can be selected and added to the box listing the chosen files.
2. For each of the files selected using the pull-down menu, identify them as either a **Candidate** file or a **Previous Data** file. **Candidate** .csv files are comprised of possible input combinations from which the design can be constructed. The columns of the file should contain the different input factors that define the dimensions of the input space, as well as a column that will be used to specify the weights associated with each of the design points. Note that there is a requirement for a column to be used to identify the prioritized regions of the input space. If this is not provided, then a non-uniform space filling design cannot be created.

Previous Data .csv files should have the same number of columns for the input space as the candidate file (with matching column names), and represent data that have already been collected. Note that a weight column is also required for the history file, as the calculation of how close each of the points are to each other requires this. The algorithm for creating the design aims to place points farther away from locations where data have already been obtained, while also filling the input space around those locations.

Both the **Candidate** and **Previous Data** files should be .csv files that have the first row as the Column headings. The Input and Weight columns should be numeric. Additional columns are allowed and can be identified as not necessary to the design creation algorithm at a later stage.

3. Click on the **View** button to open the **Preview Inputs** pop-up window, to see the list of columns contained in each file. The left hand side displays the first few rows of input combinations from the file. Select the columns that you wish to see graphically in the right hand box. We will select columns X1, X2, and Values (use the shift, control, or command key to select multiple columns), and then click on **Plot SDOE** to see a scatterplot matrix of the data.

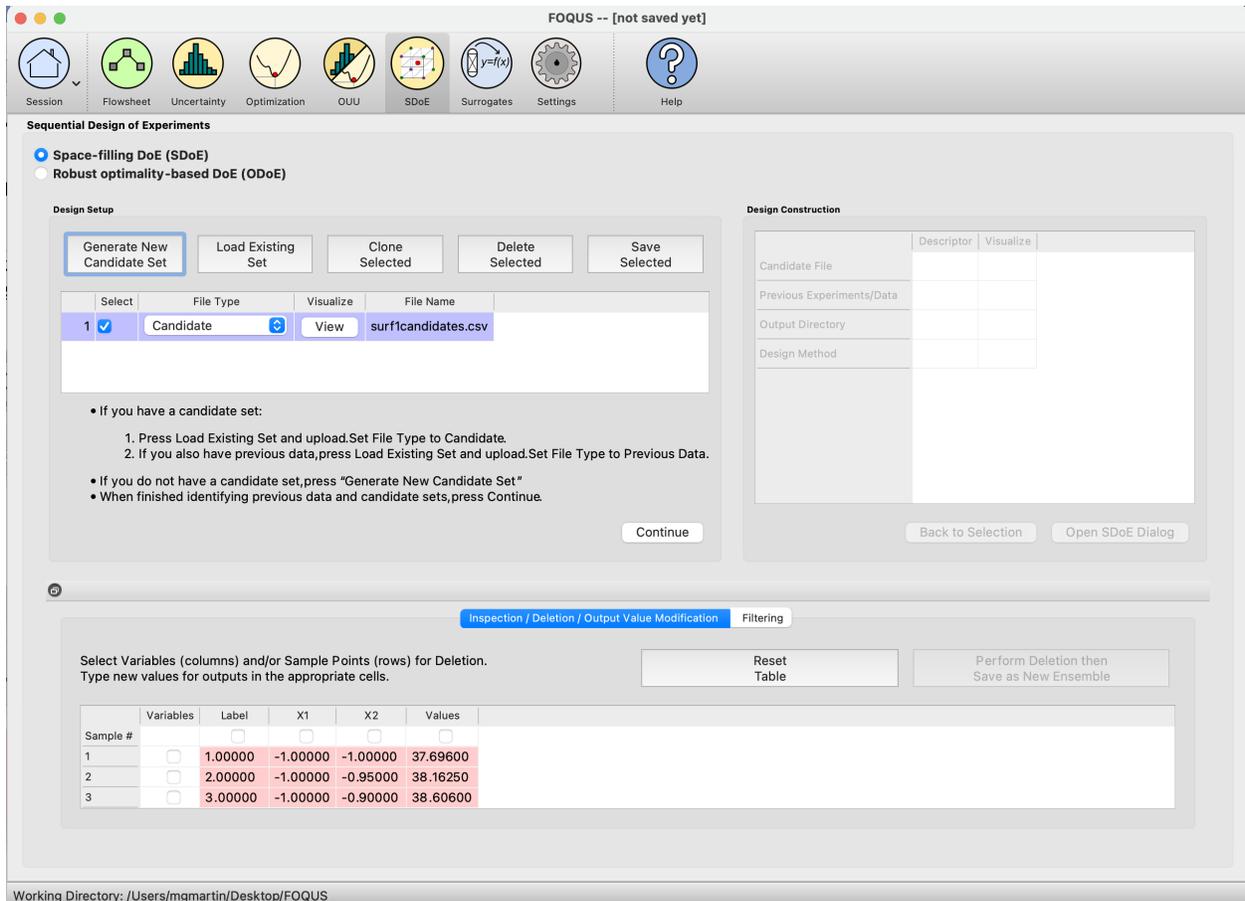


Fig. 20: SDOE Home Screen

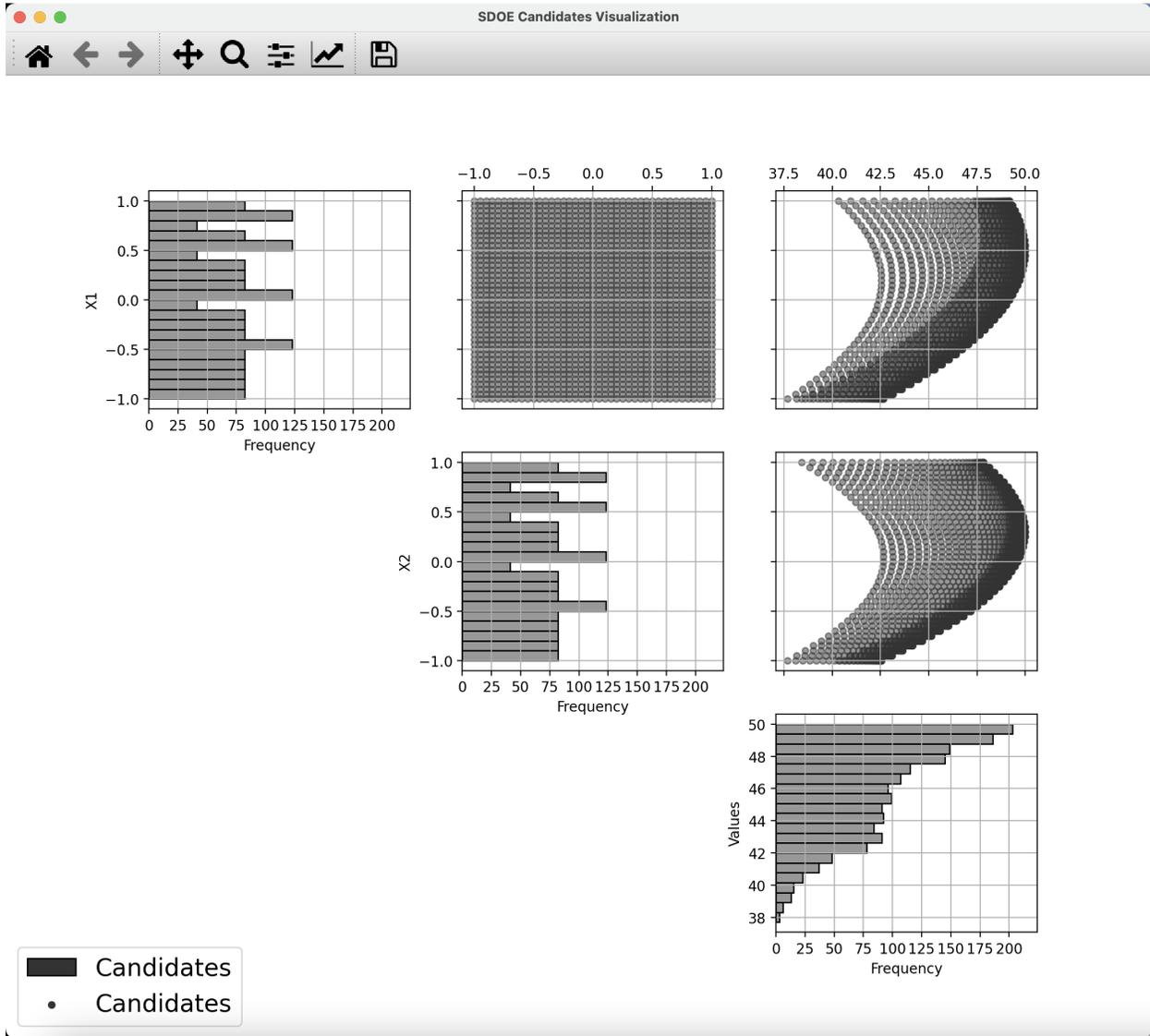


Fig. 21: SDOE plot of candidate set inputs

The plot shows histograms of each of the columns on the diagonals to provide a view of the distribution of values as well as the range of each input. The off-diagonals show pairwise scatterplots of each pair of columns selected. This should provide the experimenter with the ability to assess if the ranges specified and any constraints for the inputs have been appropriately captured for the specified candidate set. In addition, repeating this process for any historical data will provide verification that the already observed data have been suitably characterized.

Note: In this file, the “Values” column contains the numbers that will be used to define the weights. The numeric values contained in this column do not have any restrictions, except (a) there is a value provided for each row in the candidate set, and (b) that larger values correspond to points that the user wishes to emphasize with regions containing a higher density of points in the constructed design.

4. Once the data have been verified for both the **Candidate** and **Previous Data** files, click on the **Continue** button to make the **Design Construction** window active.
5. If more than one **Candidate** file was specified, then the **aggregate_candidates.csv** file that was created will have combined these files into a single file. Similarly if more than one **Previous Data** file was specified, then the **aggregate_previousData.csv** file has been created with all runs from these files. If only a single file was selected for either of the **Candidate** or **Previous Data** files, then their corresponding aggregated files will be the same as the original.

There are options to view the aggregated files for both the candidate and previous data files, simply scroll to the right of the Design Construction window, with a similar interface as was shown in step 3. In addition, a single plot of the combined candidate and previous data files can be viewed. In this plot the points representing the candidate locations and points of already collected data from the previous data file are shown in different colors.

6. Once the data have been verified as the desired set to be used for the design construction, click on the **Non-Uniform Space Filling** button at the bottom right corner of the **Design Construction** window, then select **Open SDOE Dialog**. This opens the second SDOE window, which allows for specific design choices to be made.

7. Unlike the Uniform Space Filling designs, the choice of the optimality criterion to be used is fixed at **maximin**. Recall that a **maximin** design looks to choose design points that are as far away from each other as possible. In this case, the design criterion is looking to maximize a weighted value of how close any two points are away from their nearest neighbor. Larger weights inflate the calculated distance function larger, thus making the apparent distance between the points seem closer than their standard non-weighted Euclidean distance.

8. The next choice to be made falls under **Scaling Method**, where the experimenter can select how the column specified in the **Weight** column will be scaled. The scaling translates the values in the column specified with the **Weight** label directly to the new range of [1, MWR], where MWR = Maximum Weight Ratio, which will be specified in the next step. The smallest value in the weight column (MinValue) gets mapped to the value 1, while the largest value in the column (MaxValue) gets mapped to the value MWR (which will be specified in the next step). For the **Direct MWR** option, the shape of the histogram of the values is preserved, through the formula:

$$\text{Scaled Weight} = 1 + ((\text{MWR} - 1) * (\text{Value} - \text{MinValue}) / (\text{MaxValue} - \text{MinValue}))$$

For the **Ranked MWR** option, the values are sorted from smallest to largest (ties allowed) and then assigned a rank. Rank = 1 corresponds to the smallest value, while the largest Rank is the number of rows in the candidate set (NumCand). Then the scaled weights are assigned through the formula:

$$\text{Scaled Weight} = 1 + ((\text{MWR} - 1) * (\text{Rank} - 1) / (\text{NumCand} - 1))$$

Note: The designs created are dependent on the choice of weights selected. The **Ranked MWR** choice creates a uniformly spaced order of points from “least important” to “most important” that results in a symmetric flat histogram for the weights, while the **Direct MWR** scaling preserves the shape of the original values. If the user is not sure which of the choices is better suited to their problem, we recommend generating designs for both choices and comparing the results to see which are a better match for desired spacing throughout the input space.

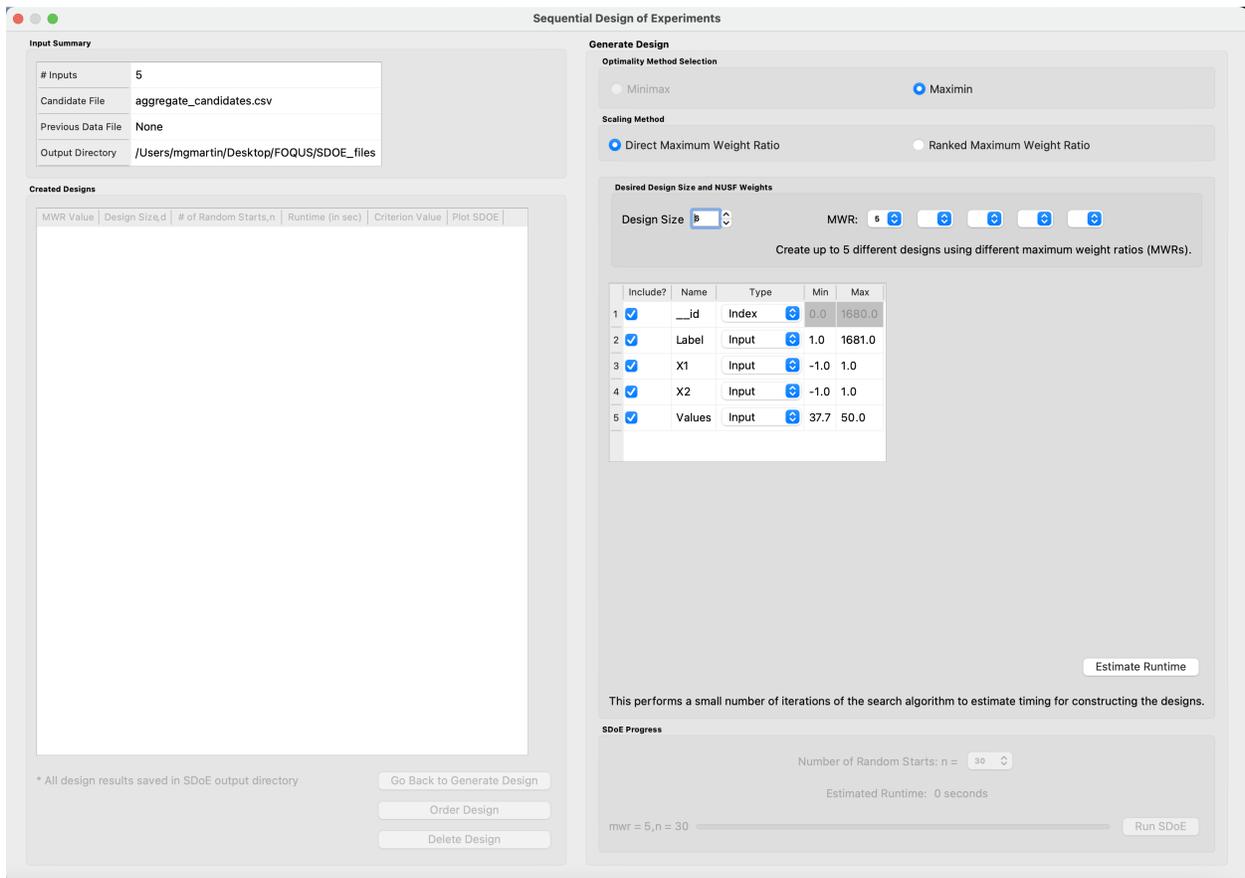


Fig. 22: SDOE second window

9. Next, there are options for the values of the Maximum Weight Ratio (**MWR**) to be used. This is an important step in the Non-Uniform Space Filling design process, as it gives the user control about how much difference there is in the density of points. Smaller values of MWR (close to 1), result in a nearly uniform design. Larger values result in a design that has a higher density of design points for the higher weighed regions, and more sparse for the lower weighted regions. Since how this value impacts the density of the design is also a function of the histogram of the values for the **Weight** column and the choice of the **Scaling Method**, we recommend constructing designs for several MWR values and comparing their results.

The user can specify up to 5 **MWR** values, where for each of the **MWR** boxes, there is a set of choices that range from 2 to 60. This range should provide considerably flexibility in choosing how unequal the spacing will be throughout the design space.

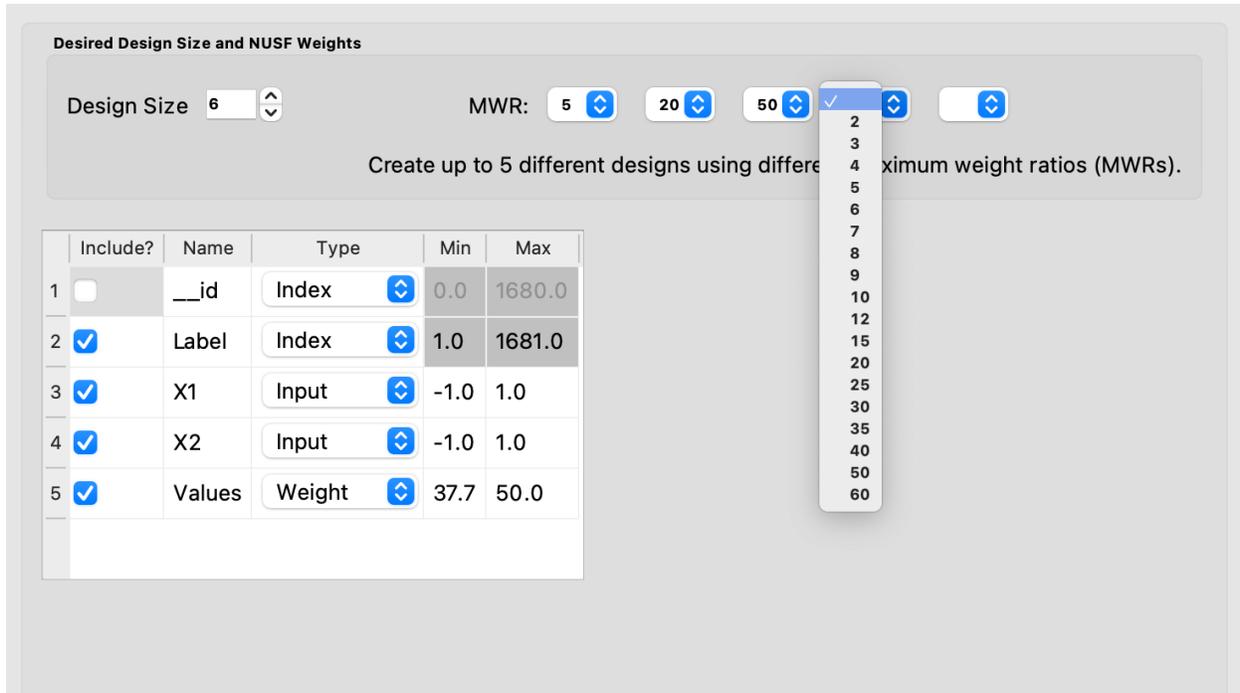


Fig. 23: Choice of MWR Value and Columns

Note: Here are some recommendations about the role of the MWR value and the choice of scaling:

- Think about changes to the MWR as multiplicative or exponential (e.g. 1 - 2 - 4 - 8 - 16), not linear (e.g. 1 - 2 - 3 - 4 - 5).
- If there are many candidate points that should be weighted approximately equally, the direct weight scaling might be more appropriate. The ranked weighting tends to spread out the final weights for similar values.
- If the original candidate set weight distribution is close to uniformly distributed, then the **Ranked MWR** and **Direct MWR** scalings will produce very similar designs.
- The ranked scaling for weights makes it easier to predict what the impact of a choice of MWR value will be (since the initial weight distribution is always approximately the same).
- As the skew of the direct weight distribution increases, the effective MWR becomes consistently smaller than the chosen value (only a small fraction of the candidates are using the edges of [1, MWR] range). Hence, for skewed distributions, a larger MWR might be needed for the Direct scaling to get a design that is similar to a given **MWR** value for the Ranked weight scaling.

Also in this step, the columns of the candidate set to be used for the construction of the design are identified. Under **Include?** in the box on the right hand side, the experimenter has the option of choosing whether particular columns should be included in the space-filling design search. Uncheck a box, if a particular column should not be included in the search.

Next select the **Type** for each column. Typically most of the columns will be designated as **Inputs**, which means that they will be used to define the input space and to find the best design. For the Non-Uniform Space Design, there is a required column for the **Weights**, which designates which rows in the candidate to emphasize (bigger weights) and which to de-emphasize (smaller weights). In addition, we recommend including one **Index** column which contains a unique identifier for each run of the candidate set. This makes tracking which runs are included in the constructed designs easier. If no **Index** column is specified, a warning appears later in the process, but this column, while recommended, is not strictly required.

Notice there is a new variable included in the first row of this box called **__id**. This column is an automatically-generated index of all rows of the candidate set, meaning the column counts up from 1, uniquely identifying each row. For example, if the candidate set contains 50 rows excluding the row of column names, the **__id** column would be 1, 2, 3, ..., 49, 50. The **Include** box next to **__id** can be unchecked if including this index column is not desired, but again, it is highly encouraged to have an index column to easily identify which candidate set rows are chosen in the design. The **__id** column **Type** is automatically set to **Index**. If using a different variable as the index column, make sure to uncheck the **Include** box next to **__id** and also change the **Type** of the desired index column to **Index**.

Finally, the **Min** and **Max** columns in the box allow the range of values for each input column to be specified. The default is to extract the smallest and largest values from the candidate and previous data files, and use these. This approach generally works well, as it scales the inputs to be in a uniform hypercube for comparing distances between the design points.

Note: The default values for **Min** and **Max** can generally be left at their defaults unless: (1) the range of some inputs represent very different amounts of change in the process. For example, if temperature is held nearly constant, while a flow rate changes substantially, then it may be desirable to extend the range of the temperature beyond its nominal values to make the amount of change in temperature more commensurate with the amount of change in the flow rate. This is a helpful strategy to make the calculated distance between any points a more accurate reflection of how much of an adjustment each input requires. (2) if changes are made in the candidate or previous data files. For example, if one set of designs are created from one candidate set, and then another set of designs are created from a different candidate set. These designs and the achieved criterion value will not be comparable unless the range of each input has been fixed at matching values.

10. Once the design choices have been made, click on the **Estimate Runtime** button. This generates a small number of iterations of the search algorithm to calibrate the timing for constructing and evaluating the designs. The time taken to generate a design is a function of the size of the candidate set, the size of the design, as well as the dimension of the input space.

Note: The number of random starts looks very different from what was done with the Uniform Space Filling Design. In that case, the number of random starts was offered in powers of 10. In this case, since a more sophisticated search algorithm is being used, each random start takes longer to run, but generally many fewer starts are needed. There is a set of choices for the number of random starts, which ranges from 10 to 1000. Producing a sample design for demonstration purposes with a small number of random starts (say 10 to 30) should work adequately, but recall that the choice of **Number of Random Starts** involves a trade-off between the quality of the design generated and the time to generate the design. The larger the chosen number of random starts, the better the design is likely to be. However, there are diminishing gains for increasingly large numbers of random starts. If running the actual experiment is expensive, it is generally recommended to choose as large a number of random starts as possible for the available time frame, to maximize the quality of the design generated.

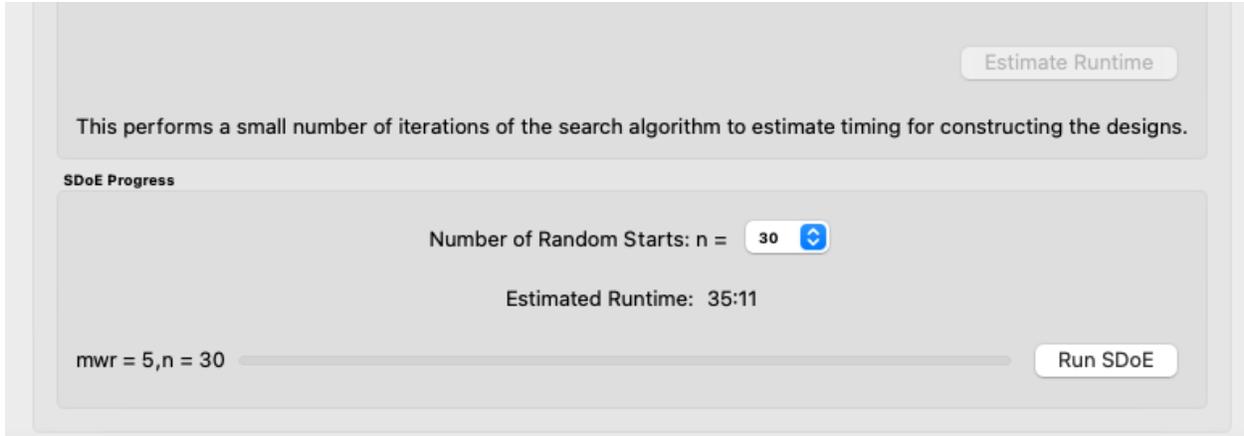


Fig. 24: Test SDOE timing

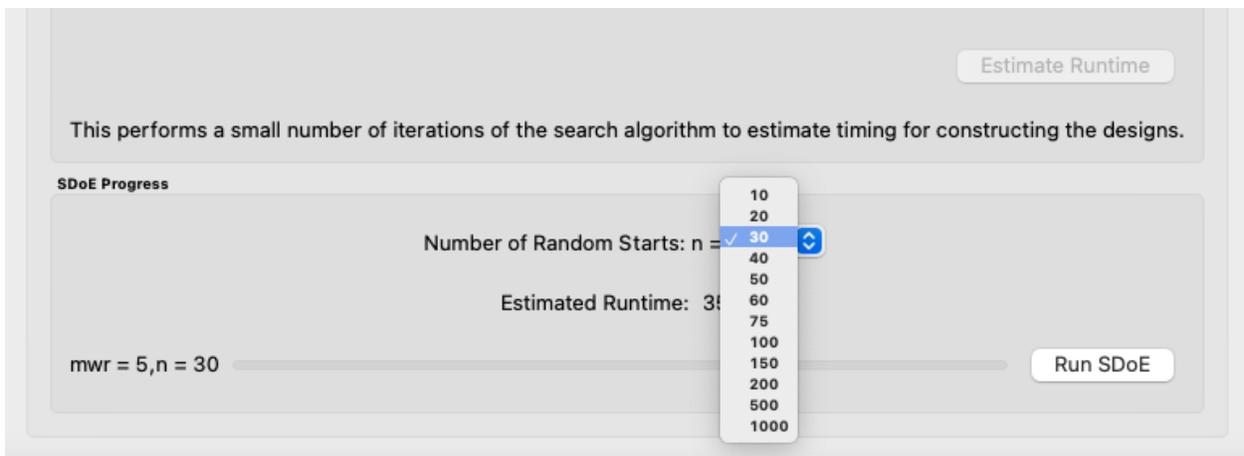


Fig. 25: Number of Random Start choices

11. Once the slider has been set to the desired **Number of Random Starts**, click on the **Run SDOE** button, and initiate the construction of the designs. The progress bar indicates how design construction is advancing through the chosen range of designs for each of the **MWR** values specified.

12. When the SDOE module has completed the design creation process, the left window **Created Designs** will be populated with files containing the results. The column entries summarize the key features of each of the designs, including **MWR**, **Design Size** (d, the number of runs in the created design), **# of Random Starts**, **n**, **Runtime** (number of seconds needed to create the design), **Criterion Value** (the value obtained for the maximin criterion for the saved design). Note that the criterion values are specific to the MWR value chosen, and hence should not be considered comparable across different values.

	MWR Value	Design Size, d	# of Random Starts, n	Runtime (in sec)	Criterion Value	Plot SDOE
1	5	6	30	737.23	4.06	View
2	20	6	30	690.91	55.48	View
3	50	6	30	679.25	345.41	View

Fig. 26: SDOE Created Designs

13. As with the Uniform Space Filling designs, to see details of the design, the **View** button at the right hand side of each design row can be selected to show a table of the design, as well as a pairwise scatterplot of the input and weight columns for the chosen design. The table and plot of the design are similar in characteristics to their counterparts for the candidate set. Candidate points and previous data are still shown in gray and pink, respectively, while the newly selected design points are shown in blue. If multiple designs were created with different **MWR** values (or using the different **Scaling Method** choices), it is helpful to examine the plots to compare their properties to those sought by the experimenter. A final choice should be made based on what is needed for the goals of the study.

14. Similar to the Uniform Space Filling designs, to access the file with the generated design, go to the **SDOE_files** folder, and a separate folder will have been created for each of the designs. The structure of the folder and files corresponds to what was done in the Uniform Space filling design instructions. The labeling of the files is a bit different to reflect the choices that the user made in creating the design. For example, the file **nusf_d10_n1000_m30_Label+w+G+lldg+L+Values.csv** contains the design of size 10 (d10), generated from 1000 random starts (n1000), with the maximum weight ratio (MWR) set to 30 (m30). The columns from the file that were used include “Label”, “w”, “G”, “lldg”, “L” and “Values”.

When one of the design files is opened it contains the details of each of the runs in the design, with the input factor levels that should be set for that run.

To evaluate and compare the designs that have been created, it is helpful to look at a number of summaries, including the criteria values and visualizing the spread of the design points throughout the region. Recall that at the beginning of the design creation process we recommended constructing multiple designs, with different MWR values, choosing between the Direct and Ranked weighting strategies, and potentially with different design sizes. By examining multiple designs, it is easier to determine which design is best suited to the requirements of the experiment.

In the **Created Designs** table, it is possible to see the criterion values for each of the designs. When comparing two designs of the same size with the same MWR value, the **maximin** criterion should be made as large as possible.

However, comparisons between designs with the same MWR value but of different sizes share the same issues that were present in the uniform space filling case. Adding more runs to the design will mean that nearest neighbors will need to get closer together, and hence we would expect that on average the criterion value would get smaller for larger experiments. Hence, we want to evaluate whether the closer packing of the design points from a larger experiment is worth the increase in cost for the additional runs.

Making comparisons for designs with different **MWR** values based on the design criterion is not recommended, because the distance metric that is embedded in the non-uniform space filling design approach adjusts based on the selected MWR value. Hence, it is not possible to make a direct comparison or easy interpretation of the values from the criterion for this approach.

Hence for the NUSF designs, it is critical to use the **View** option to look at graphical summaries of the designs. Two plots are produced: The first is the **Closest Distance by Weight (CDBW) plot**, and the second is the more familiar **pairwise scatterplot** of the created design.

First, we describe the information that is contained in the CDBW plot. There are two portions to the plot. The lower section shows a histogram of the weights in the candidate set. Note that the range of values goes from 1 to the MWR value selected. For the figure below, we are looking at a design created with a MWR value of 5. The shape of the histogram shows what values were available to be selected from the candidate set. The top portion of the plot, has a vertical line for each of the design points selected (in this case 15 vertical lines for 15 design points). The location of each vertical line shows the weight for the selected design point.

Second, a pairwise scatter plot of the design is provided to see how the design points fill the input space. Since the spread of the points throughout the design space is intentionally non-uniform, it is helpful to see how the distribution matches up with the specified weights provided in the candidate set. Recall that larger values of MWR lead to designs that are less evenly distributed, while MWR values that approach 1 will become closer to uniform.

When **Previous Data** points have been incorporated into the design, the plots will show how the overall collection of points fill the input space. When examining the scatterplots, it is important to assess (a) whether the increase in concentration of points is located in the desired region?, (b) is the degree of non-uniformity what was desired?, (c) how close the design points have been placed to the edges of the region?, (d) are there holes in the design space that are unacceptably large?, and (e) does a larger design show a worthwhile improvement in the density of points to justify the additional expense?

Recall that the effect of different MWR values depends on the size of the design, the spread of weights provided across the candidate points and the shape of the input region of interest. Hence, constructing several designs and comparing them can be an effective approach for obtaining the right design.

Based on the visualization of the spread of the points, the best design can be chosen that balances design performance with an appropriate use of the available budget. Recall that with sequential design of experiments, runs that are not used in the early stages might provide the opportunity for more runs at later stages. So the entire sequence of experimental runs should be considered when making choices about each stage.

Basic Steps for an Input-Response Space-Filling Design

We now consider some details for each of these steps for the third type of design, where we want a design that seeks to balance even spacing in the input space with even spacing in the response space. These designs can be used when information is known about the likely output values for input combinations of interest. When information about the likely output values is not available, a uniform space-filling design should be used.

We wish to simultaneously optimize space-filling in the input and the response spaces. However, it is often not possible to achieve maximum space-filling in both spaces simultaneously. Instead, we may seek a design that offers a compromise, performing relatively well in both the input and output spaces. Rather than offering a single “best” design, the Input Response Space-Filling Design algorithm constructs a Pareto front of designs, a collection of objectively best compromise designs that move across a spectrum of levels of input and response space-filling. At one

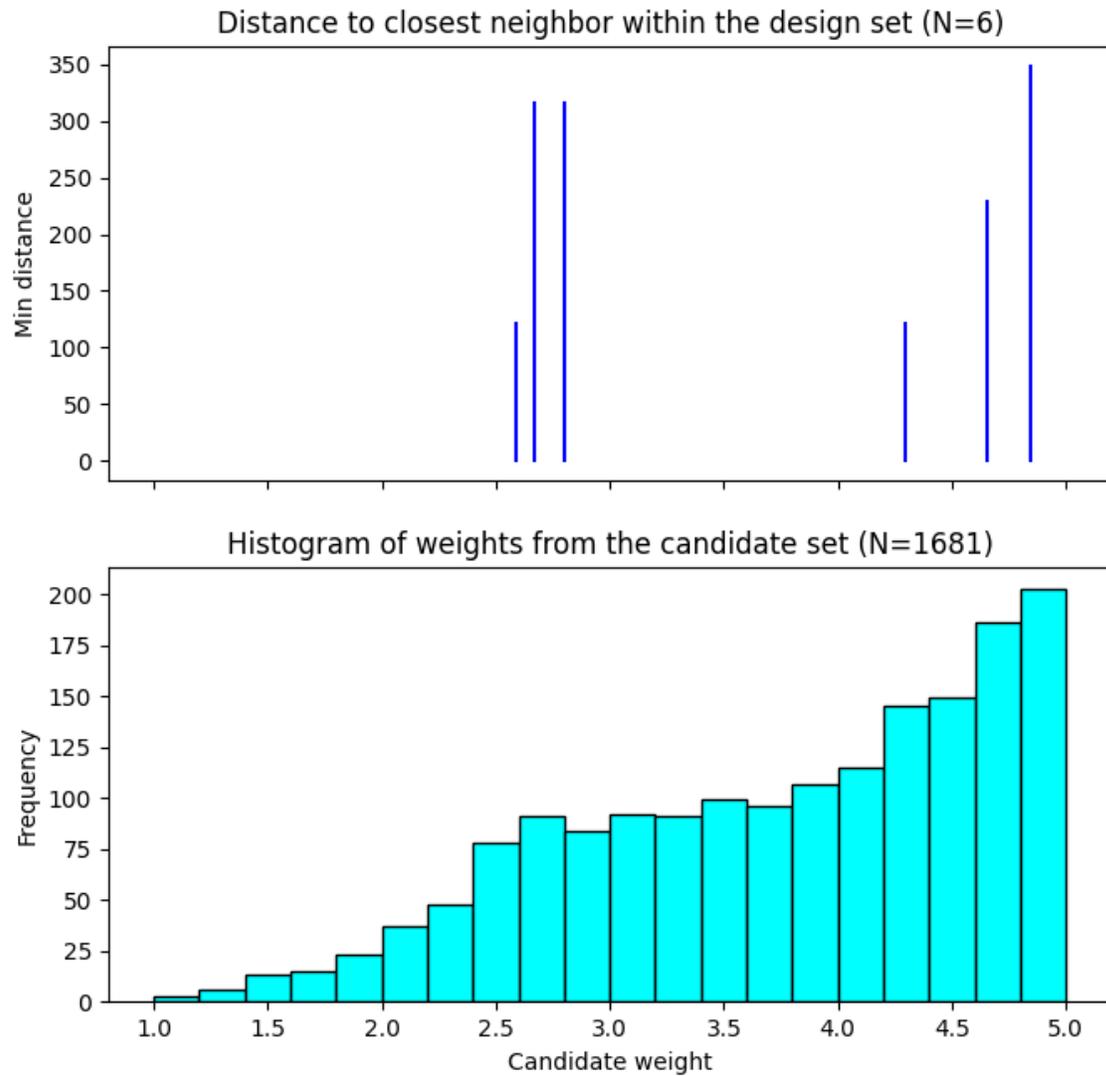


Fig. 27: A sample Closest Distance by Weight (CDBW) plot for a 6-run design with MWR value of 5

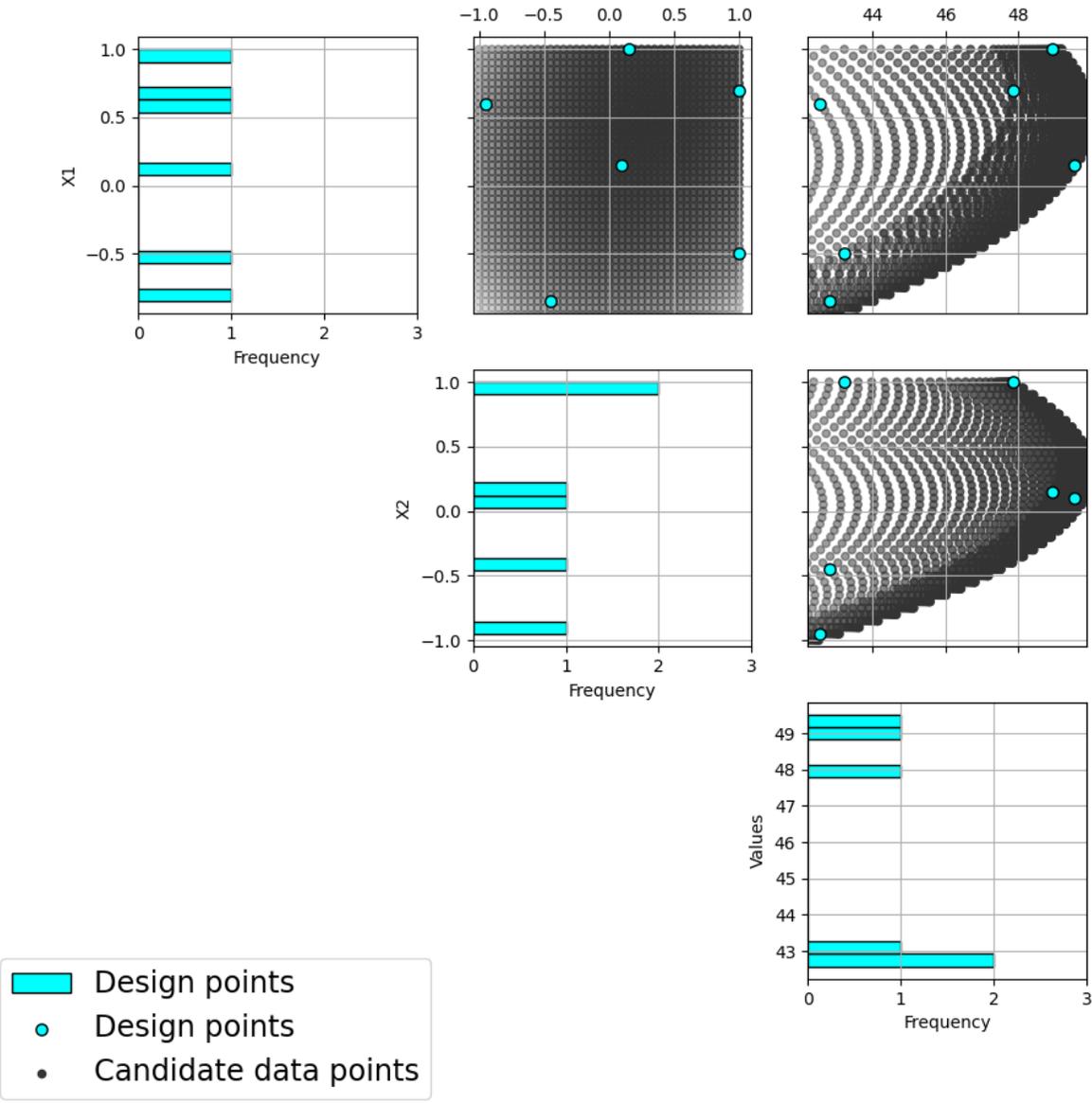


Fig. 28: A sample pairwise scatterplot for the constructed design with 6 runs and a MWR value of 5

end of the spectrum is the design that maximizes space-filling in the response space, while at the other end is the design that maximizes space-filling in the input space. Connecting these two extremes, there is a collection of compromise designs that balance space-filling in both spaces. Each design located on the Pareto front is the best compromise design for that spot along the spectrum. Experimenters should examine each design located on the Pareto front to find which of these compromise designs best suits their needs.

A step-by-step guide for using the SDOE module to create an Input-Response Space-Filling design is given below. For a set of worked examples, see the Examples section.

1. In the **Design Setup** box, click on the **Load Existing Set** button to select the file(s) to be used for the construction of the design. Several files can be selected and added to the box listing the chosen files.

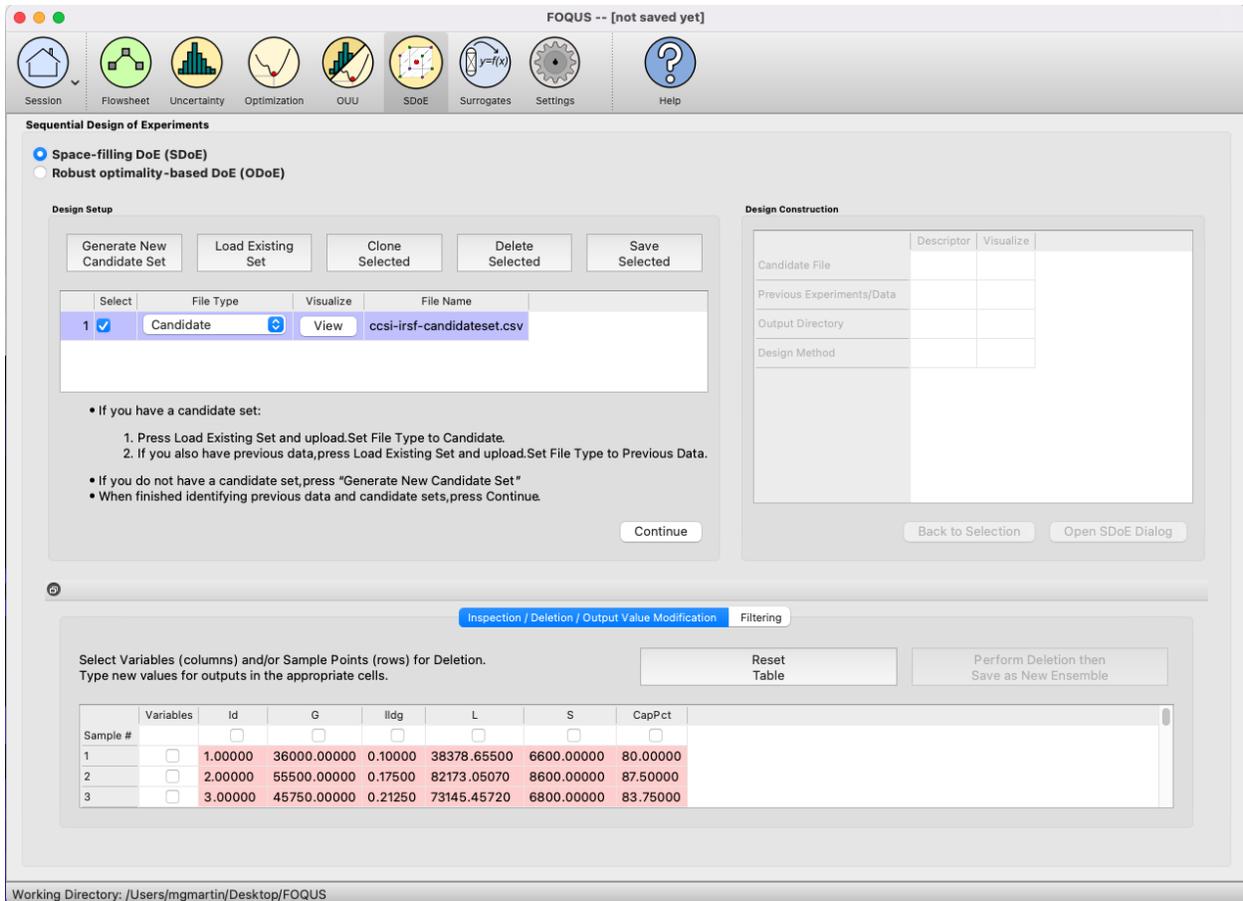


Fig. 29: SDOE Module Home Screen

2. For each of the files selected, using the pull-down menu, identify them as either a **Candidate file** or a **Previous Data** file. For Input-Response Space-Filling designs, **Candidate** .csv files are comprised of possible input and likely response combinations from which the design can be constructed. In this file, there should be one column corresponding to each of the different input factors that define the dimensions of the input space. Additionally, there should be at least one column of likely response values included for each input combination. Typically, these likely response values are determined from a previously-validated model of the underlying process. The provided likely response values will be used to identify designs with good space-filling properties in the input space. Therefore, the determination of even spacing in the response space is only as trustworthy as the model used to generate the likely response values.

Note: It is important to make sure the process model is reliable and provides consistent results before attempting an

input-response space-filling design. If not, it is recommended that the experimenter instead use a uniform space-filling design.

As previously stated, there is a requirement for at least one column to contain the response values. If this is not provided, then an input-response space-filling design cannot be created.

Previous Data .csv files should have the same number of columns for the input space as the candidate file (with matching column names), and represent data that have already been collected. Note that at least one response column is also required for the previous data file, as the determination of even spacing in the response space when taking into account previous data requires this. The algorithm for creating the design aims to fill the input and response spaces, while also not repeating input or response combinations that have already been run, as listed in the previous data.

Both the **Candidate** and **Previous Data** files should be .csv files that have the first row as the Column headings. The Input and Response columns should be numeric. Additional columns are allowed and can, if desired, be identified as not necessary to the design creation algorithm at a later stage.

	A	B	C	D	E	F	G
1	Id	G	lldg	L	S	CapPct	
2	1	36000	0.1	38378.655	6600	80	
3	2	55500	0.175	82173.051	8600	87.5	
4	3	45750	0.2125	73145.457	6800	83.75	
5	4	65250	0.1375	90543.204	11200	91.25	

Fig. 30: Columns in this Candidate Set

- Click on the **View** button to open the **Preview Inputs** pop-up window, to see the list of columns contained in each file. The left-hand side displays the first few rows of input combinations and responses from the file. Select the columns that you wish to see graphically in the right-hand box, and then click **Plot SDOE** to see a scatterplot matrix of the data.

Displayed on the diagonals of the scatterplot matrix are histograms of each of the columns. These plots provide a view of the distribution of values as well as the range of each input. The off-diagonals show pairwise scatterplots of each pair of columns selected. This should provide the experimenter with the ability to assess if the ranges specified and any constraints for the inputs have been appropriately captured for the specified candidate set. In addition, repeating this process for any previous data will provide verification that the already observed data have been suitably characterized.

- Once the data have been verified for both the **Candidate** and **Previous Data** files, click on the **Continue** button to make the **Design Construction** window active.
- If more than one **Candidate** file was specified, then the **aggregate_candidates.csv** file that was created will have combined these files into a single file. Similarly, if more than one **Previous Data** file was specified, then the **aggregate_previousData.csv** file has been created with all runs from these files. If only a single file was selected for either of the **Candidate** or **Previous Data** files, then its corresponding aggregated file will be the same as the original.

To view the aggregated files for both the candidate and previous data files (if provided), click **View**, which lies in the right-most column of the Output Directory row. Once selected, this has a similar interface as that shown in step 3. If both types of files have been provided, a single plot of the combined candidate and previous data files will be displayed. In this plot, the points representing the candidate locations and points of already collected data from the previous data file are shown in different colors.

Note: Make sure to include previous data if it exists. If it exists and is included, the design creation algorithm will

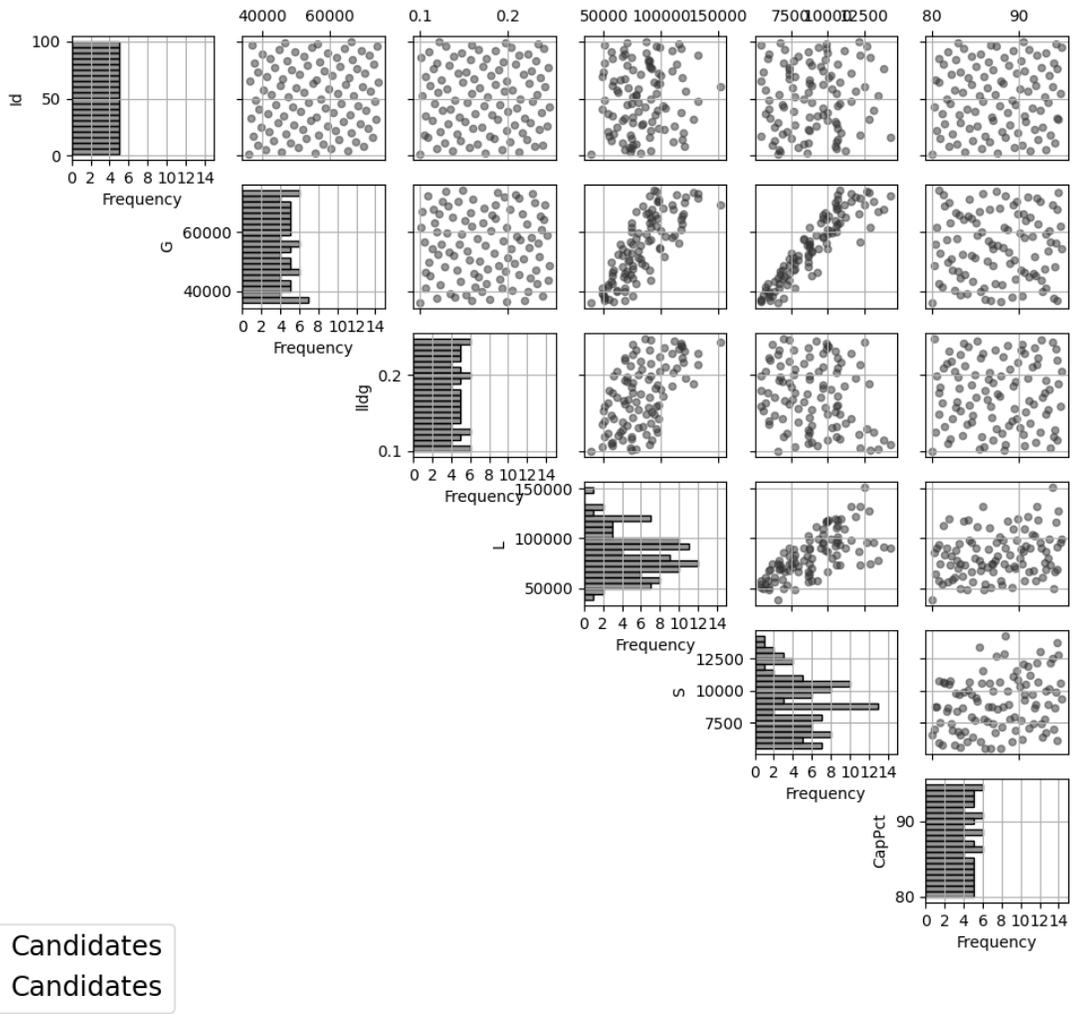


Fig. 31: Viewing Candidate Set

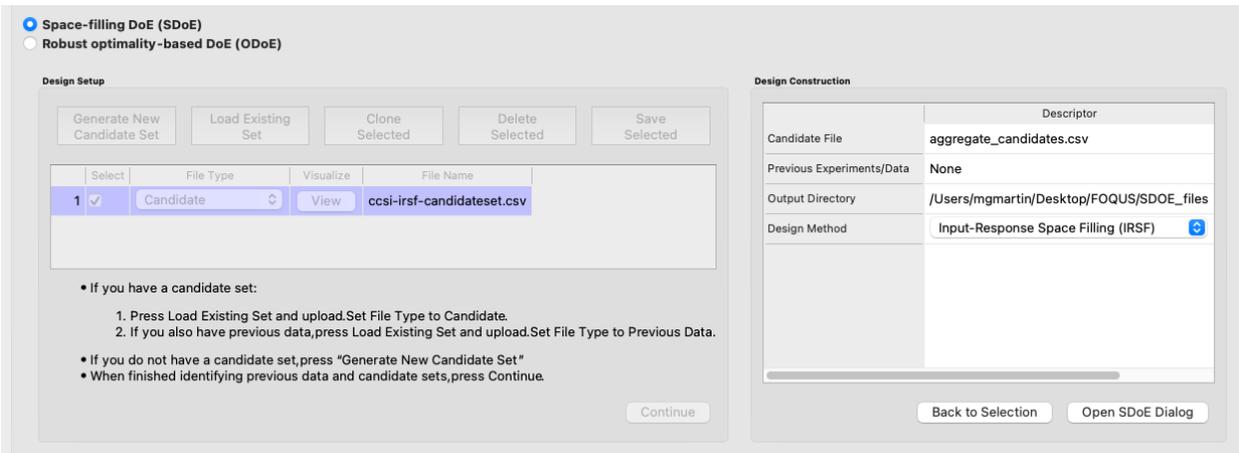


Fig. 32: SDOE Design Construction Window

spread new design points out so as not to rerun an input combination that has already been run.

- Once the data have been verified as the desired set to be used for the design construction, click on **Input-Response Space Filling** from the **Design Method** drop-down menu in the **Design Construction** window. This opens the second SDOE window, which allows for specific design choices to be made.

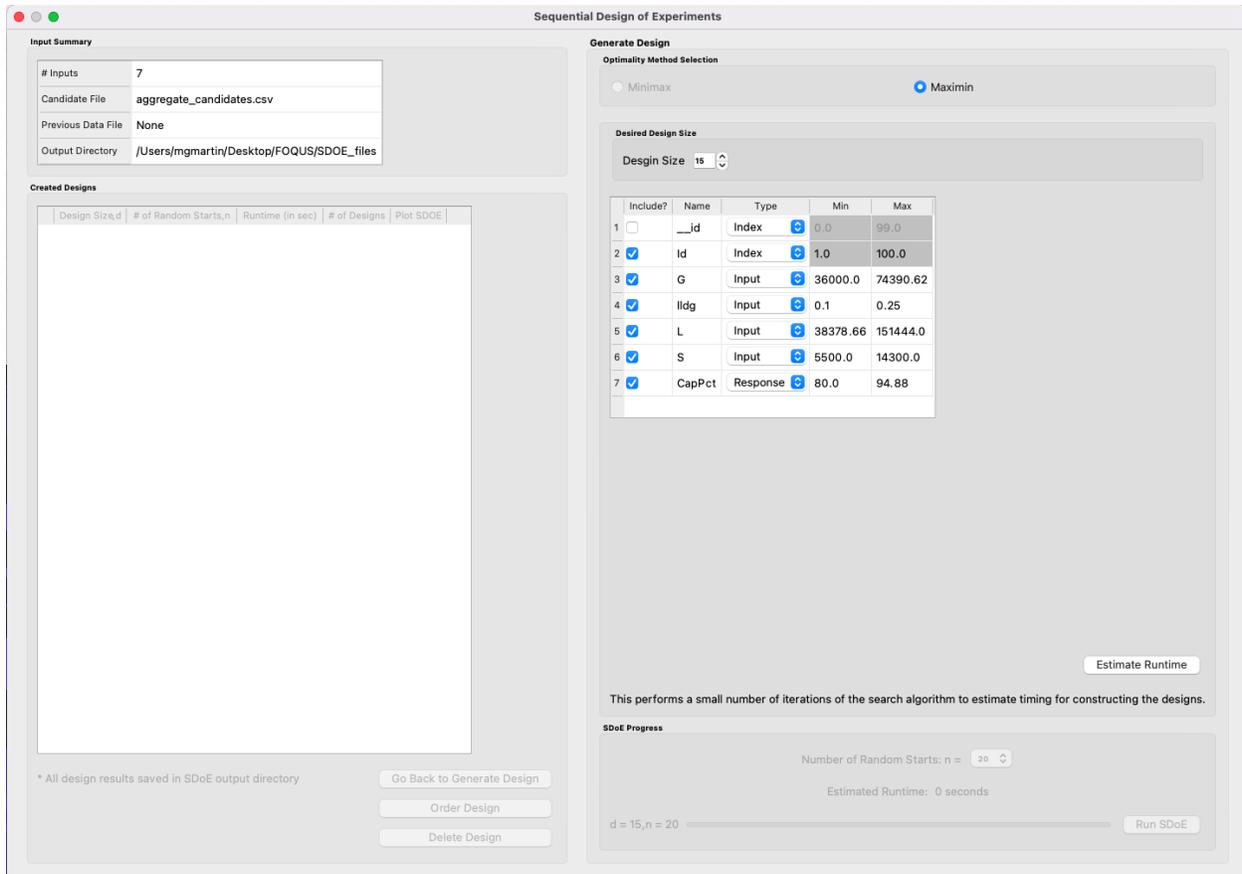


Fig. 33: SDOE Second Page

- Similar to Non-Uniform Space Filling designs, the choice of the optimality criterion to be used is fixed at **maximin**. Recall that a **maximin** design looks to choose design points that are as far away from each other as possible. In this case, the design criterion is looking to maximize how close any two points in the input space are away from their nearest neighbor and the same in the response space.
- Select the **Size** of the design desired. A larger design will give more information than a smaller design. This choice often comes down to the size of the budget for the experiment.
- Next select the **Type** for each column. In general, most of the columns should be designated as **Inputs**, which means they will be used to define the input space and to find the best design for the input space. For Input-Response Space-Filling designs in particular, there is a required column for the **Response**, which the experimenter will determine from the model. Multiple response columns can be given if desired. The algorithm will use the response(s) to find the best design for the response space. All of the Input and Response columns will be used in the determination of the Pareto front of best designs in both spaces.

In addition, there is a system-created **Index** column displayed amongst the other columns of the candidate set; it should be listed first. Using an index column makes tracking which runs are included in the constructed designs easier. It will have the name “_id” with a Min value of 1 and Max value that is the number of rows in the set. The

Type will be pre-set to “Index”. If the candidate set already included an index column, simply uncheck the **Include?** checkbox next to the column name that should be left out of design creation. Only one Index column can be included in design creation. If using a different index column than the one provided, remember to change the **Type** to **Index**.

Finally, the **Min** and **Max** columns in the box allow the range of values for each input column, except for “_id”, to be specified. The default is to extract the smallest and largest values from the candidate and previous data files, and use these. This approach generally works well, as it scales the inputs to be in a uniform hypercube for comparing distances between the design points.

Note: The default values for **Min** and **Max** can generally be left at their defaults unless: (1) The range of some inputs represent very different amounts of change in the process. For example, if temperature is held nearly constant, while a flow rate changes substantially, then it may be desirable to extend the range of the temperature beyond its nominal values to make the amount of change in temperature more commensurate with the amount of change in the flow rate. This is a helpful strategy to make the calculated distance between any points a more accurate reflection of how much of an adjustment each input requires. (2) If changes are made in the candidate or previous data files. For example, if one set of designs are created from one candidate set, and then another set of designs are created from a different candidate set. These designs and the achieved criterion value will not be comparable unless the range of each input has been fixed at matching values.

10. Once the design choices have been made, click on the **Estimate Runtime** button. This generates a small number of iterations of the search algorithm to calibrate the timing for constructing and evaluating the designs. The time taken to generate a design is a function of the size of the candidate set, the size of the design, as well as the dimensions of the input space and response space.

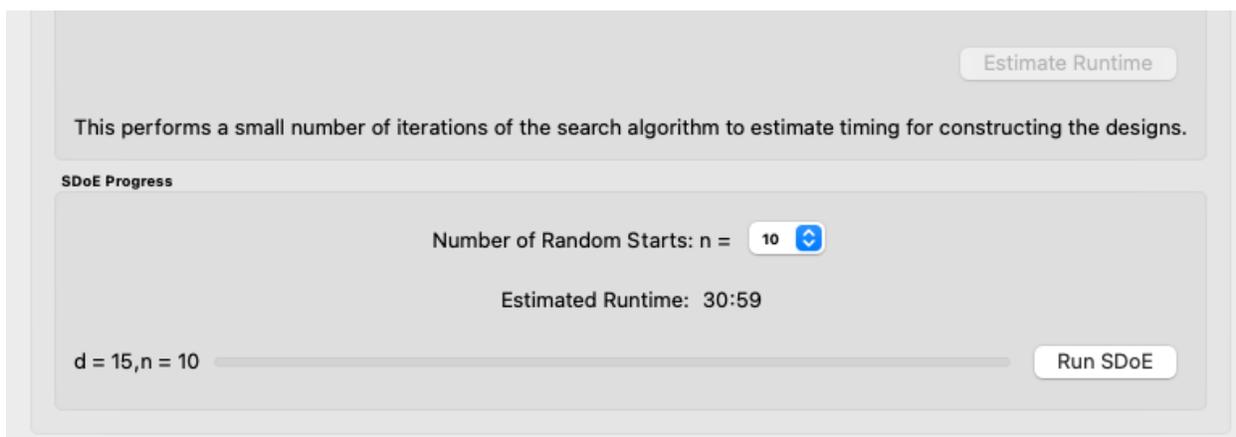


Fig. 34: Number of Random Starts

Note: The number of random starts looks very different from what was done with the Uniform Space Filling Design.

In that case, the number of random starts was offered in powers of 10. In this case, similar to Non-Uniform Space-Filling, since a more sophisticated search algorithm is being used, each random start takes longer to run, but generally many fewer starts are needed. There is a set of choices for the number of random starts, which ranges from 5 to 500. Producing a sample design for demonstration purposes with a small number of random starts (say 5 to 30) should work adequately, but recall that the choice of **Number of Random Starts** involves a trade-off between the quality of the design generated and the time to generate the design. The larger the chosen number of random starts, the better the design is likely to be. However, there are diminishing gains for increasingly large numbers of random starts. If running the actual experiment is expensive, it is generally recommended to choose as large a number of random starts as possible for the available time frame, to maximize the quality of the design generated.

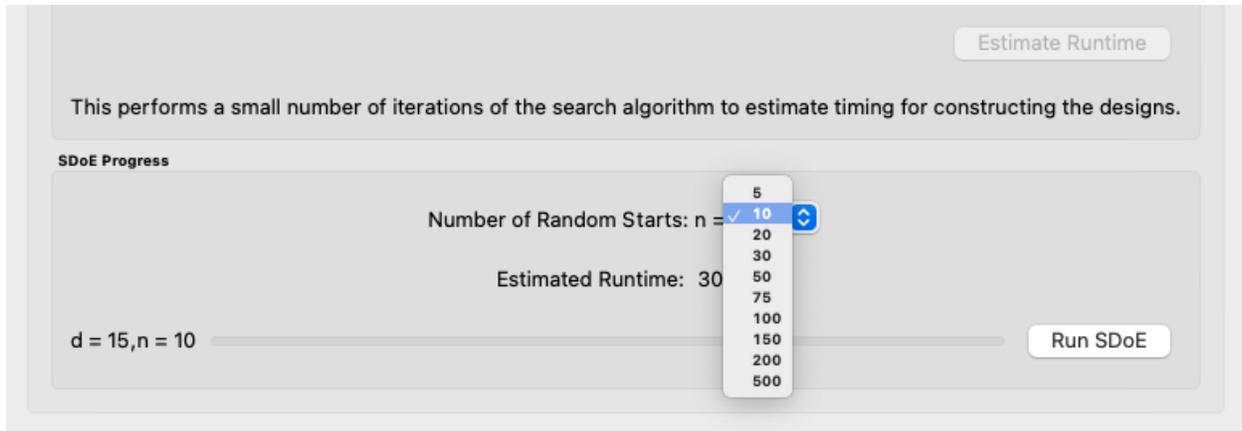


Fig. 35: Choosing the Number of Random Starts

11. Once the slider has been set to the desired **Number of Random Starts**, click on the **Run SDOE** button, and initiate the construction of the designs.
12. When the SDOE module has completed the design creation process, the left window **Created Designs** will be populated with a single file containing all results. The column entries summarize the key features of the collection of designs, including **Design Size** (d, the number of runs in each of the created designs), **# of Random Starts** (n), **Runtime** (number of seconds needed to create the designs), **# of Designs** (the number of designs found on the Pareto front). Clicking the **View** button in the **Plot SDOE** column gives a view of the Pareto front, with options to examine each of the created designs individually.

	Design Size, d	# of Random Starts, n	Runtime (in sec)	# of Designs	Plot SDOE
1	15	10	1857.97	5	View

Fig. 36: Created Designs Window

13. To view each of the designs on the Pareto front, click **View**. The plot given is of the Pareto front, with circles indicating the varying trade-offs of input and response space-filling criteria of each design on the Pareto front. By definition, these are all “best” designs along some spectrum of space-filling in the input and response spaces. There are a large number of other designs that would have been created, but when evaluated, would have been dominated (have worse space-filling) in at least one dimension by a design along the Pareto front.
Click on any circle in the plot to see a pairwise scatterplot of that individual design. These created-design pairwise scatterplots are similar in characteristics to their counterparts for the candidate set. It is helpful to examine the plots to compare their properties to those sought by the experimenter. A final choice should be made based on what is needed for the goals of the study.

14. To access the files with the generated designs, go to the **SDOE_files** folder, and a single folder will have been created for each Pareto front of designs created. This folder will have a name containing the date and time the designs were created. When opened, csv files of all created designs will be listed in the order they appear on the Pareto front, with the best-response design displayed first, and the best-input design second-to-last. The last file in the folder will

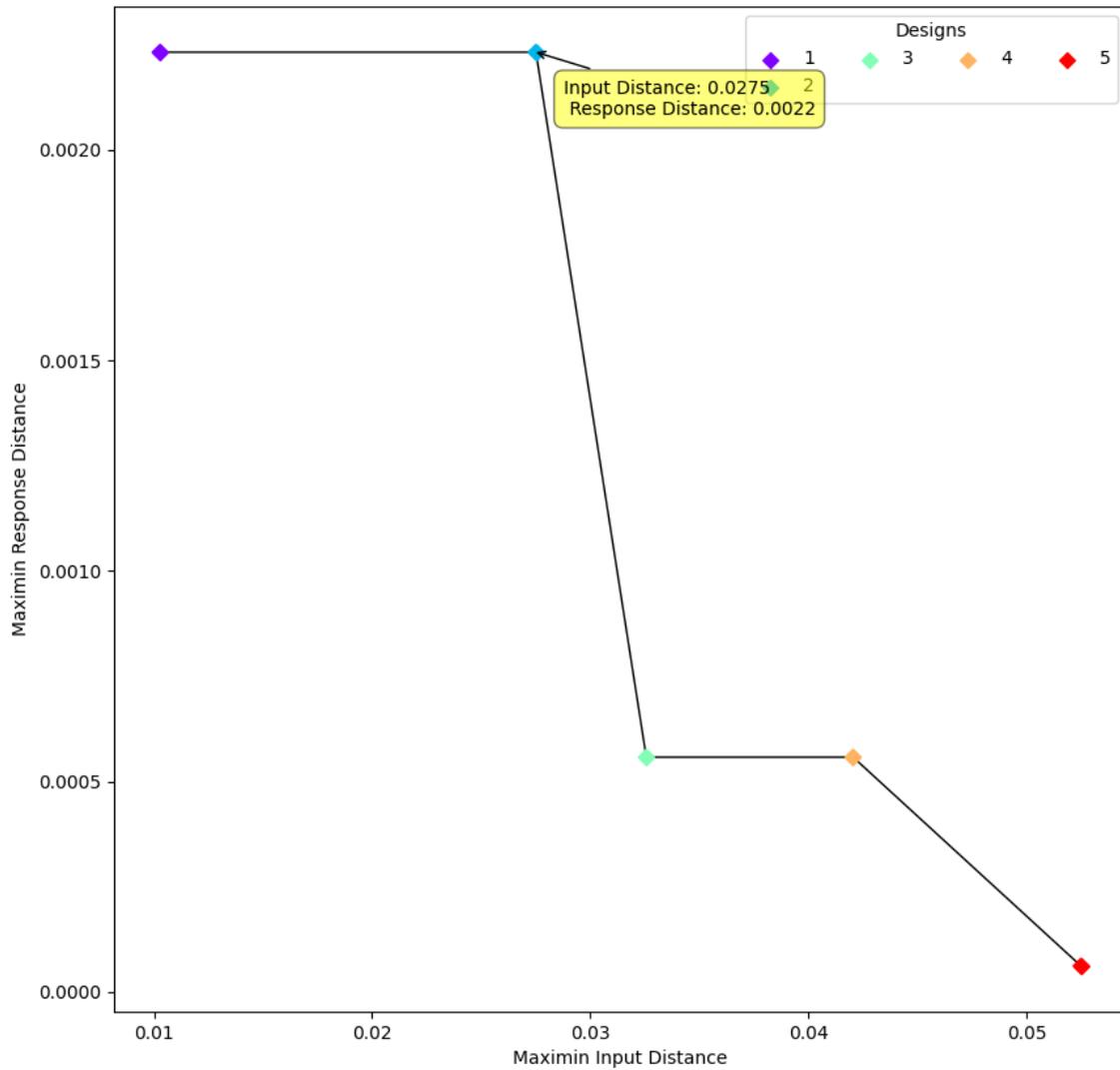


Fig. 37: Viewing a Pareto Front

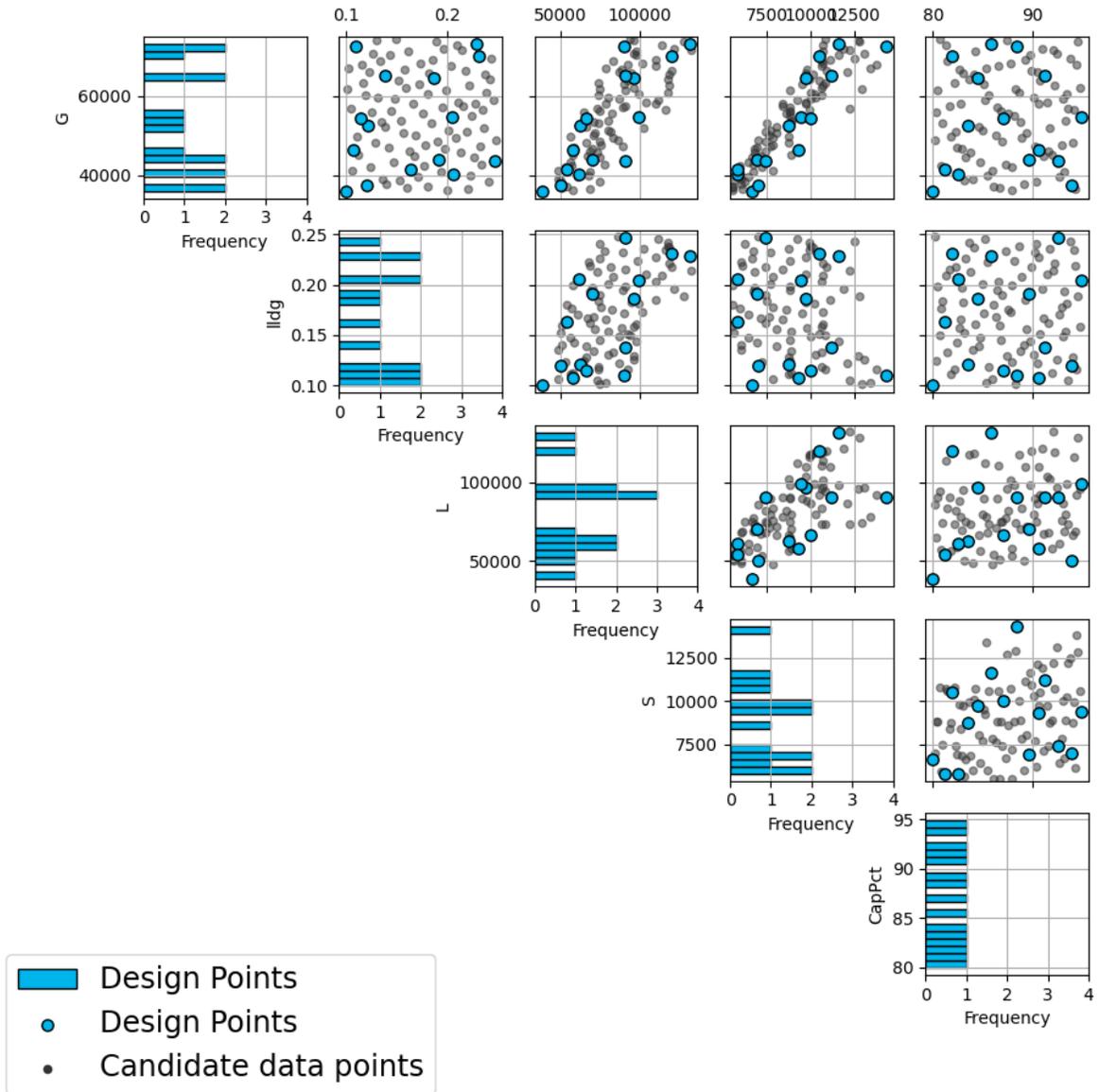


Fig. 38: Viewing the Pairwise Scatterplot of a Created Design

be a csv file of the Pareto front, containing the values of the input and response space-filling criteria for each design.

The created design files will have names similar to those given to files of Uniform and Non-Uniform Space-Filling designs mentioned in previous sections. The labeling reflects choices made by the experimenter in creating the designs. For example, the file `irsf_design2_d15_n30_id+G+lldg+L+S+CapPct.csv` contains the Input-Response Space-Filling design (irsf) of size 15 (d15) generated from 30 random starts (n30). This design is the second design on the Pareto front (design2), which means it has the second-highest value of the response space-filling criterion, and the second-lowest value of the input space-filling criterion. The columns from the file that were used include “_id” (system-generated ID column), “G”, “lldg”, “L”, “S”, and “CapPct”.

When one of these design files is opened it contains the details of each of the runs in the design, with the input factor levels that should be set for that run.

To evaluate and compare the designs that have been created, it is helpful to look at a number of summaries, including the criteria values of input and response space filling, and visualizing the spread of the design points throughout the region by studying the pairwise scatterplots. Recall that at the beginning of the design creation process we recommended constructing multiple sets of designs. By examining many designs, it is easier to determine which design is best suited to the requirements of the experiment.

Efficient Implementation of Experimental Run Order

Once designs have been created, it is often important to optimize the run order to efficiently reach equilibrium and allow for the maximum number of runs to be implemented within a constrained budget or time period. While statisticians generally recommend using a randomized order for the experimental runs, it can sometimes mean the difference of a small randomized experiment versus a larger non-randomized experiment.

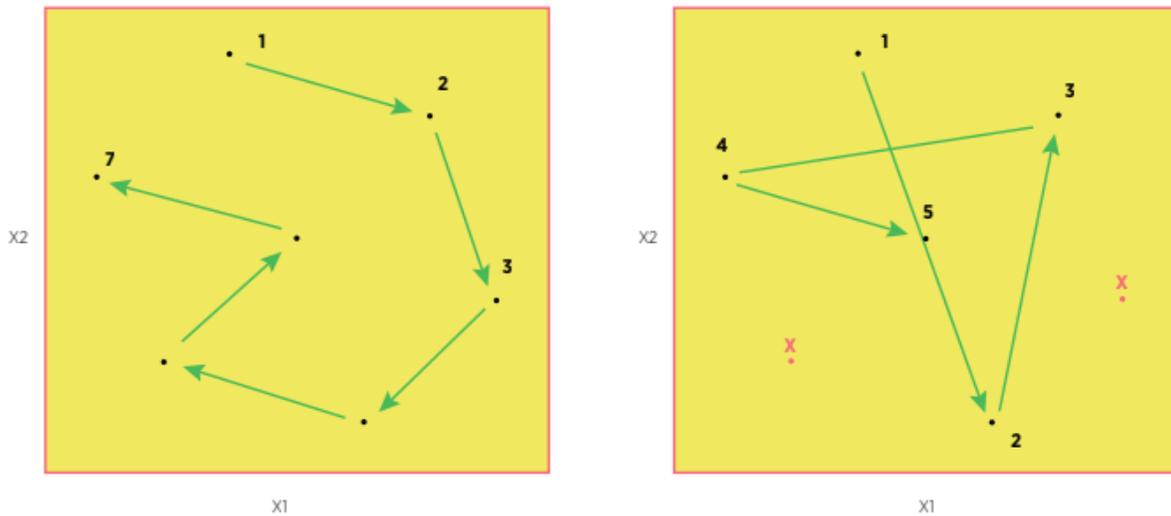


Fig. 39: Comparison of the number of runs possible with an optimized run order (left) versus an inefficient randomized run order (right)

In this section we describe how to generate an efficient run order for a design created using the Uniform Space Filling or Non-Uniform Space Filling design options.

Once we created a design (USF or NUSF), it appears on the left panel in the **Created Designs** table. Click on the design that we want to order (it is highlighted in blue as shown below). Then click on the button below named **Order Design**, to order the design points in an efficient run order that sequences the runs to favor having nearby points adjacent to each other in the run order.

A pop up window confirms the location of the newly ordered file (see below). Click ‘Yes’ to continue.

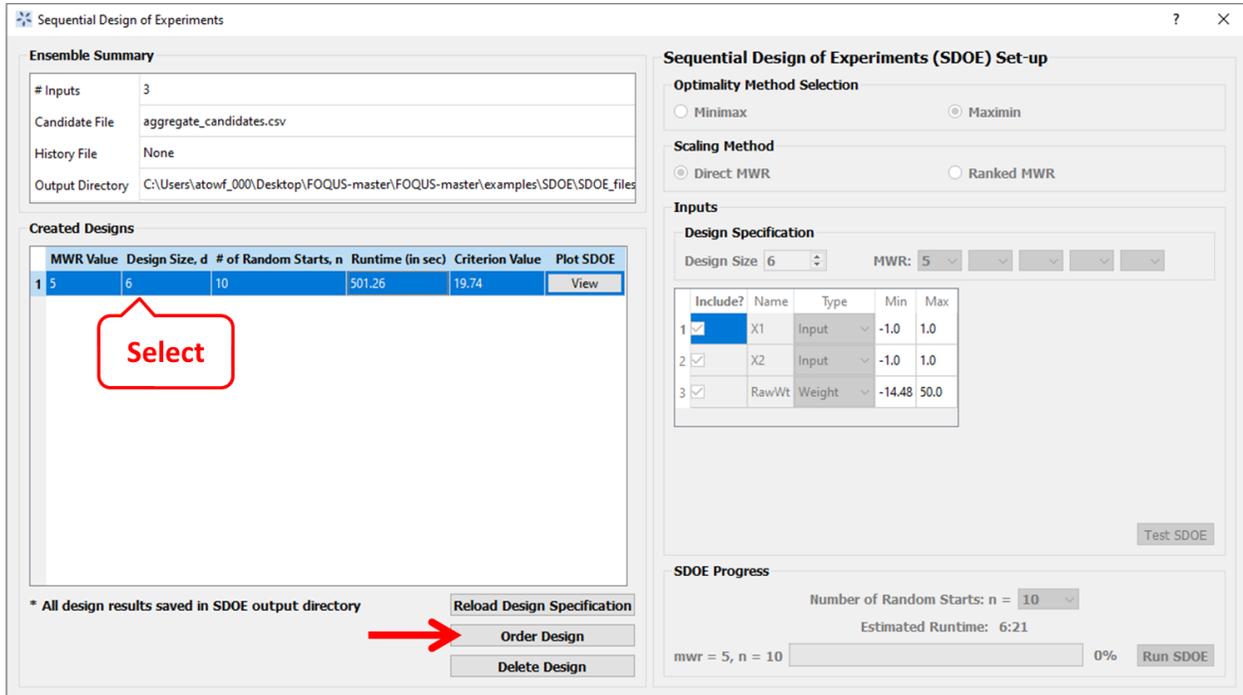


Fig. 40: How to create an ordered design

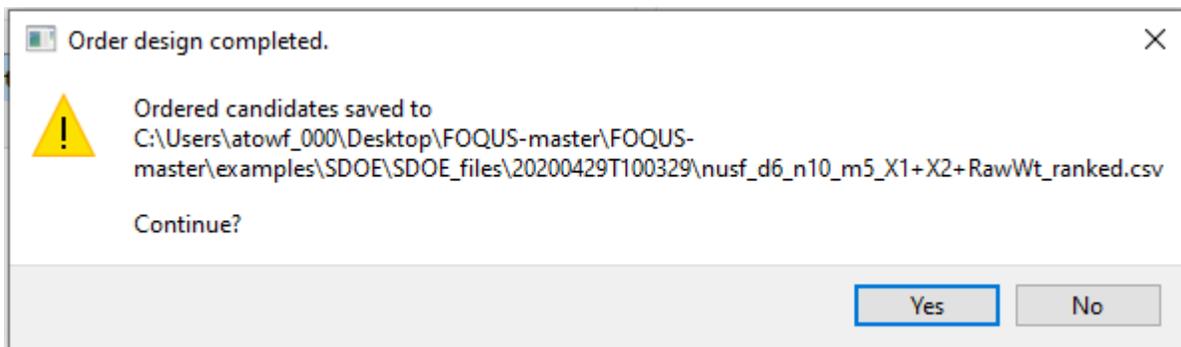


Fig. 41: Message window for new design created

Both design files (located in the designated folder) are saved in the csv format, which can be opened with your preferred application (e.g. Microsoft Excel). You can produce a scatterplot of the ordered design file either using FOQUS or any other external application.

The ordering scheme provides a method for the user to design the experimental run order that follows the minimal path distance to traverse from one design point to another, i.e., minimal changes the the experimental processes. This standardizes the range of each input factor t to be between 0 and 1, and then minimizes the sum of the Euclidean distances between all of the points. Often this would be a preferred operational implementation to increase the efficiency of the experiment, by reducing the time for the process to reach equilibrium. The implementation provided uses the TSP (travelling sales person) algorithm as implemented in the 'python-tsp' library package for ordering/ranking the design points.

An alternative to this approach is a simple sequential ordering (ascending or descending) of the most expensive input factor. This is easily implemented by the user, and can be efficient for the running of the experiment, but should be used cautiously because the run order might confound other changes in the system during the implementation of the experiment.

8.1.3 Examples

Next, we illustrate the use of the SDOE capability for several different scenarios.

Example USF-1 constructs several uniform space filling designs of size 8 to 10 runs for a 2-dimensional input space based on a regular square region with a candidate set that is a regularly spaced grid. Both minimax and maximin designs are constructed to illustrate the difference in the criteria. Example USF-2 takes one of the designs created in Example 1, and considers how it might be used for sequential updating with additional experimentation. In this case the Example 1 design is considered as historical data, and the goal is to augment the design with several additional runs. Example USF-3 considers a 5-dimensional input space based on a CCSI example, and demonstrates what the process of Sequential Design of Experiments might look like with several iterations of constructing uniform space filling designs.

Example NUSF-1 constructs several non-uniform space filing designs of size 15 in a 2-dimensional regular input space. Several designs are generated using the same weights, but with different Maximum Weight Ratios (MWRs), to illustrate how the concentration of points can be altered to match the experimenter's preferences. Example NUSF-2 considers a CCSI example, with a non-regular region, and the weights that were derived from the width of the confidence interval for prediction based on an existing model. The goal is to concentrate more of the new runs in regions where there is greater uncertainty, and hence the widths of the confidence intervals are larger. Again multiple designs are created to show how the MWR influences the concentration of the points in different regions.

Example IRSF-1 constructs a set of "best" designs (a Pareto front) along a spectrum of input and response space-filling. The designs are based on a 2-dimensional input space and a 1-dimensional response. Different designs along the Pareto front are compared to illustrate (a) what a Pareto front is and (b) how to choose a design from those on the Pareto front.

The files for these tutorials are located in: `examples/tutorial_files/SDOE`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

Example USF-1: Constructing Uniform Space Filling minimax and maximin designs for a 2-D input space

For this first example, the goal is to construct a simple space-filling design with between 8 and 10 runs in a 2-dimensional space based on a regular unconstrained square region populated with a grid of candidate points.

1. From the FOQUS main screen, click the **SDOE** button. On the top left side, select **Load Existing Set**, and select the SDOE_Ex1_Candidates.csv file from examples folder. This identifies the possible input combinations from which the design will be constructed. The more possible candidates that can be provided to the search algorithm used to construct the design, the better the design might be for the specified criterion.

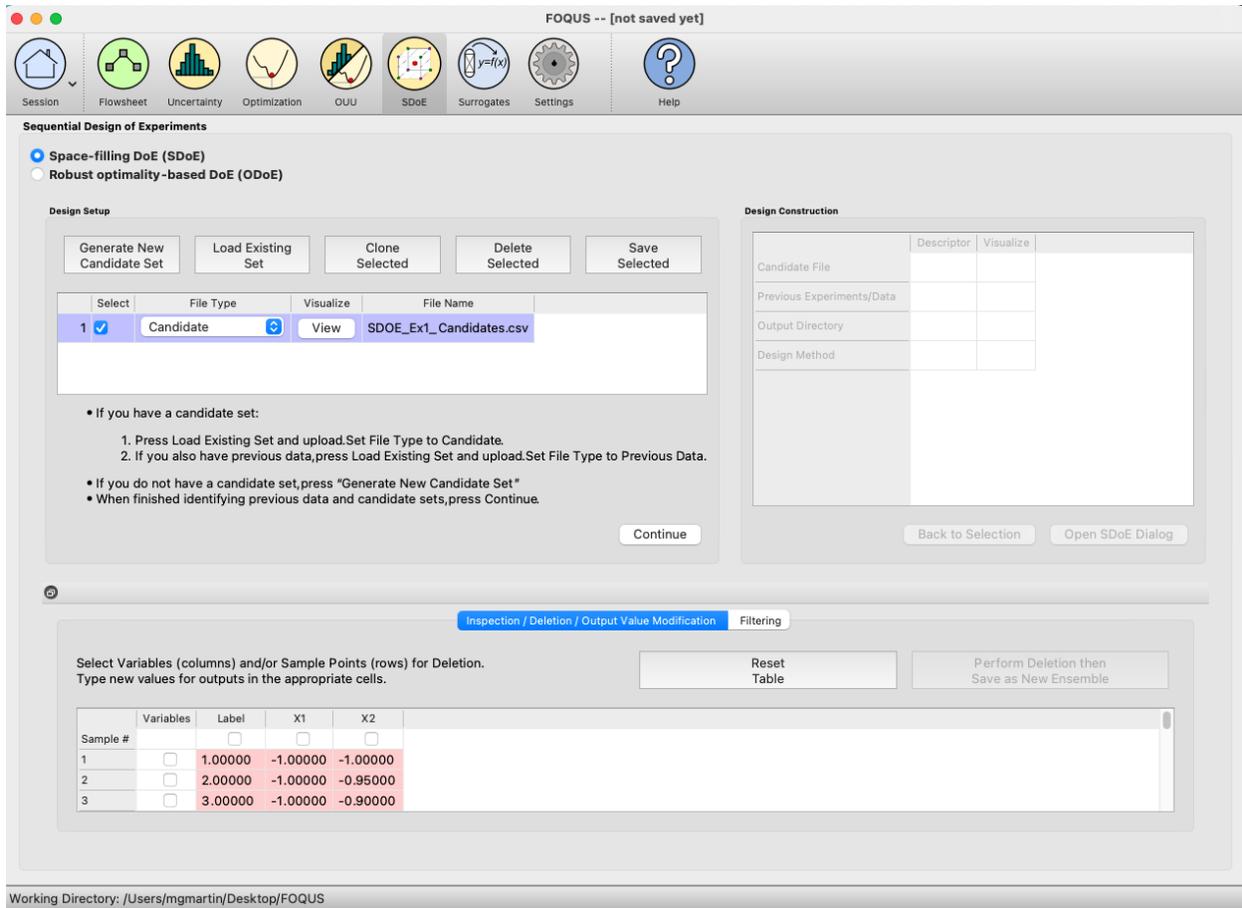


Fig. 42: Ex 1 Design Setup

2. Next, by selecting **View** and then **Plot** it is possible to see the grid of points that will be used as the candidate points. In this case, the range for each of the inputs, X1 and X2, has been chosen to be between -1 and 1.
3. Next, click on **Continue** to advance to the **Design Construction** Window, and then select **Uniform Space Filling** and click on **Open SDOE Dialog** to advance to the second SDOE screen, where particular choices about the design can be made. On the second screen, select **minimax** for the **Optimality Method Selection**. Change the **Min Design Size** and **Max Design Size** to 8 and 10, respectively. This will construct 3 minimax designs of size 8, 9 and 10. Next, uncheck the column called **Label**. This will mean that the design is not constructed using this as an input. There should be an **_id** column automatically created containing unique identifiers to identify which runs from the candidate set were chosen for the final designs. Since the ranges of each of X1 and X2 are the bounds that we want to use for creating this design, we do not need to change the entries in **Min** and **Max**.
4. Once the choices for the design have been specified, click on the **Estimate Runtime** button to estimate the time

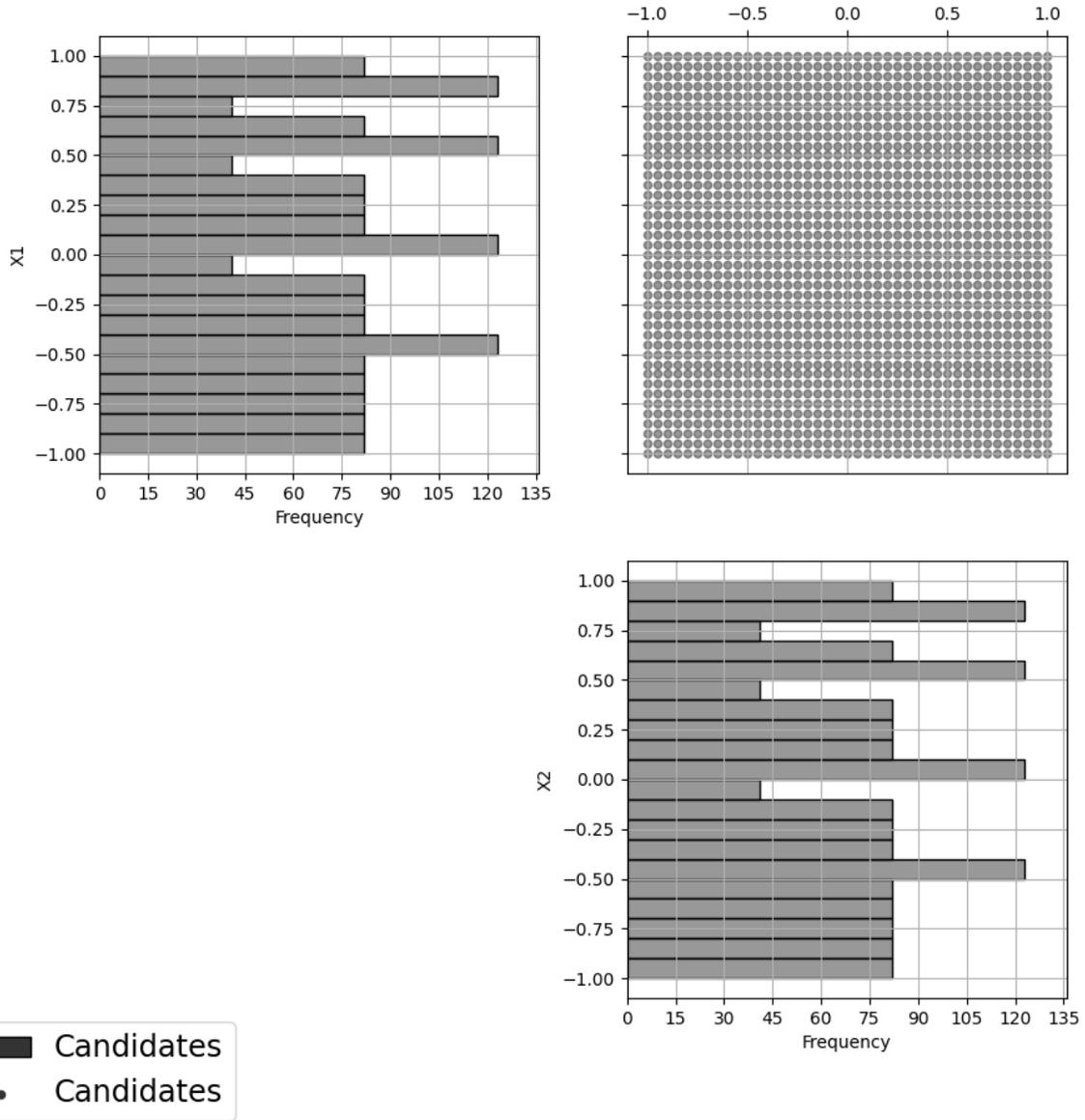


Fig. 43: Ex 1 Candidate Grid

Generate Design

Optimality Method Selection

Minimax Maximin

Desired Design Sizes

Min Design Size

Max Design Size

	Include?	Name	Type	Min	Max
1	<input checked="" type="checkbox"/>	__id	Index	0.0	1680.0
2	<input type="checkbox"/>	Label	Input	1.0	1681.0
3	<input checked="" type="checkbox"/>	X1	Input	-1.0	1.0
4	<input checked="" type="checkbox"/>	X2	Input	-1.0	1.0

This performs a small number of iterations of the search algorithm to estimate timing for constructing the designs.

SDoE Progress

Number of Random Starts: $n = 10^{\wedge} 5$

Estimated Runtime: 4:01

d = 8, n = 100000

Fig. 44: Ex 1 Minimax design choices

taken for creating the designs. For the computer on which this example was developed, if we ran the minimum number of random starts ($10^3=1000$), it is estimated that the code would take 2 seconds to create the three designs (of size 8, 9 and 10). If we chose $10^4=10000$ runs, then the code is estimated to take 24 seconds. It is estimated that $10^5=100000$ random starts would take 4 minutes and 1 second, while $10^6=1$ million random starts would take approximately 41 minutes and 29 seconds. In this case, we selected to create designs based on 100000 random starts, since this was a suitable balance between timeliness and giving the algorithm a chance to find the best possible designs. Hence, select 10^5 for the **Number of Random Starts**, and then click **Run SDOE**.

5. Since we are also interested in examining maximin designs for the same scenario, we click on the **Go Back to Generate Design** button in the **Created Designs** window to repopulate the right window with the same choices that we made for all of the design options.
6. After changing the **Optimality Method Selection** to **maximin**, click on **Estimate Runtime**, select 10^5 for the **Number of Random Starts**, and then click **Run SDOE**. After waiting for the prescribed time, the **Created Designs** window will have 6 created designs - three that are minimax designs and three that are maximin designs.
7. We now consider the choices between the designs to determine which is the best match for our experimental goals. We can see a list of the selected design points by clicking **View** for any of the created designs, and **Plot** allows us to see the spread of the design points throughout the input region.

Clearly, there is a trade-off between the cost of the experiment (larger number of runs involve more time, effort and expense) and how well the designs fill the space. When choosing which of the designs is most appropriate for the experiment, it is important to remember that resources spent early in the process cannot be used later, so it is helpful to balance early learning about the process, with the ability to identify and focus on the desired optimal location later.

There is also a small difference in priorities between the minimax and the maximin criteria. Minimax seeks to minimize how far any candidate point (which defines our region of interest) is from a design point. Maximin seeks to spread out the design points and maximize how close the nearest points are to each other. As noted previously, minimax designs tend to avoid putting too many points on the edge of the region, while maximin designs often place a number of points right on the edges of the input space.

By looking at the placement of the points, how well they fill the desired space, and the points proximity to the edge of the region, the user can find a good match for their experimental goals. After considering all of the trade-offs between the alternatives, select the design that best matches the goals of the experiment.

8. The file for the selected design can be found in the **SDOE_files** folder. The design can then be used to guide the implementation of the experiment with the input factor levels for each run.

Example USF-2: Augmenting the Example USF-1 design in a 2-D input space with a Uniform Space Filling Design

In this example, we consider the sequential aspect of design, by building on the first example results. Consider the scenario where based on the results of Example 1, the experimenter selected to actually implement and run the 8 run minimax design.

1. In the **Design Setup** box, click on **Load Existing Set** to select the candidate set that you would like to use for the construction of the design. This may be the same candidate set that was used in Example 1, or it might have been updated based on what was learned from the first data collection. For example, if it was learned that one corner of the design space might not be desirable, then the candidate set can be updated to remove candidate points that are now considered undesirable. For the **File Type** leave the designation as **Candidate**.

To load in the experimental runs that were already collected, click on **Load Existing Set** again, and select the design file that was created in the **SDOE_files** folder. This time, change the **File Type** to **Previous Data**. If you wish to view either of the candidate or previous data files, click on **View** to see either a table or plot.

2. Click on the **Continue** button at the bottom right of the **Design Setup** box. This will activate the **Design Construction** box.

Created Designs

	Optimality Method	Design Size,d	# of Random Starts,n	Runtime (in sec)	Criterion Value	Plot SDoE
1	minimax	8	100000	76.1	0.01	View
2	minimax	9	100000	78.28	0.01	View
3	minimax	10	100000	78.94	0.01	View

* All design results saved in SDoE output directory

[Go Back to Generate Design](#)

[Order Design](#)

[Delete Design](#)

Fig. 45: Ex 1 Minimax created designs

Input Summary

# Inputs	4
Candidate File	aggregate_candidates.csv
Previous Data File	None
Output Directory	/Users/mgmartin/Desktop/FOQUS/SDOE_files

Created Designs

	Optimality Method	Design Size,d	# of Random Starts,n	Runtime (in sec)	Criterion Value	Plot SDoE
1	minimax	8	100000	76.1	0.01	View
2	minimax	9	100000	78.28	0.01	View
3	minimax	10	100000	78.94	0.01	View
4	maximin	8	100000	76.88	0.19	View
5	maximin	9	100000	77.53	0.16	View
6	maximin	10	100000	81.78	0.15	View

* All design results saved in SDoE output directory

[Go Back to Generate Design](#)
[Order Design](#)
[Delete Design](#)

Generate Design

Optimality Method Selection

Minimax
 Maximin

Desired Design Sizes

Min Design Size:

Max Design Size:

Include?	Name	Type	Min	Max
<input checked="" type="checkbox"/>	_id	Index	0.0	1680.0
<input type="checkbox"/>	Label	Input	1.0	1681.0
<input checked="" type="checkbox"/>	X1	Input	-1.0	1.0
<input checked="" type="checkbox"/>	X2	Input	-1.0	1.0

This performs a small number of iterations of the search algorithm to e

SDoE Progress

Number of Random Starts: n = 10⁵

Estimated Runtime: 4:07

d = 10, n = 100000

Fig. 46: Ex 1 Created designs

3. After examining that the desired files have been selected, click on the **Uniform Space Filling** button at the bottom right corner of the **Design Construction** window. This will open the second SDOE window that shows the **Sequential Design of Experiments Set-Up** window on the right hand side.
4. Select **Minimax** or **Maximin** for the type of design to create.
5. Select the **Min Design Size** and **Max Design Size** to match what is desired. If you wish to just generate a single design of the desired size, make **Min Design Size = Max Design Size**. Recall that this will be the number of additional points that will be added to the existing design, not the total design size.
6. Next, select the options desired in the box: a) Should any of the columns be excluded from the design creation? If yes, then unclick the **Include?** box. b) For input factors to be used in the construction of the uniform space filling design, make sure that the **Type** is designated as **Input**. The automatically generated index column **__id** will be already listed as **Index**. If there is a label column for the candidates, then uncheck its **Include?** box to make sure only one index column is used. c) Finally, you can optionally change the **Min** and **Max** ranges for the inputs to adjust the relative emphasis that distances in each input range are designated.
7. Once the set-up choices have been made, click **Estimate Runtime** to find out what the anticipated time is for generating designs based on different numbers of random starts.
8. Select the number of random starts to use, based on available time. Recall that using more random starts is likely to produce a design that is closer to the overall best optimum.
9. After the SDOE module has created the design(s), the left window **Created Designs** is populated with the new design(s). These can be viewed with the **View** option, where the plot now shows the **Previous Data** in pink, and the newly added possible design in blue. This allows better assessment of the appropriateness of the new design subject to the data that have already been collected.
10. To access the file that contains the created designs, go to the **SDOE_files** folder. As before, a separate folder will have been created for each design.
11. If there is a desire to do another set in the sequential design, then the procedure outlined above for Example 2 can be followed again. The only change will be that this time there will be 3 files that need to be imported: A **Candidate** file from which new runs can be selected, and two **Previous Data** files. The first of these files will be the selected design from Example USF-1, and the second the newly created design that was run as a result of Example USF-2. When the user clicks on **Continue** in the **Design Setup** window, the two **Previous Data** files will be aggregated into a single **Aggregated Previous Data** file.

Example USF-3: A Uniform Space Filling Design for a Carbon Capture example in a 5-D input space

In this example, we consider a more realistic scenario of a sequential design of experiment. Here we explore a 5-dimensional input space with **G**, **lIdg**, **CapturePerc**, **L** and **SteamFlow** denoting the space that we wish to explore with a space-filling design. The candidate set, **Candidate Points 8perc**, contains 93 combinations of inputs that have been validated using an ASPEN model as possible combinations for this scenario. The goal is to collect 18 runs in two stages that fill the input space. There are some constraints on the inputs, that make the viable region irregular, and hence the candidate set is useful to avoid regions where it would be problematic to collect data.

1. After selecting the **SDOE** tab in FOQUS, click on **Load Existing Set** and select the candidate file, **Candidate Points 8perc**.
2. To see the range of each input and how the viable region of interest is captured with the candidate set, select **View** and then plot. In this case we have chosen to just show the 5 input factors in the pairwise scatterplot.
3. After clicking **Continue** in the **Design Setup** box, and then **Uniform Space Filling** from the **Design Construction** box, the **Generate Design** box will appear on the right side of the second window. Here, select the options desired for the experiment to be run. For the illustrated figure, we selected a **Minimax** design with 3 potential sizes: 10, 11, 12. We specified that the column **__id** will be used as the Index, **G**, **lIdg**, **CapturePerc**, **L**, **SteamFlow** will define the 5 factors to be used as inputs. We unclicked the **Include?** box for **CO2 captured** since we

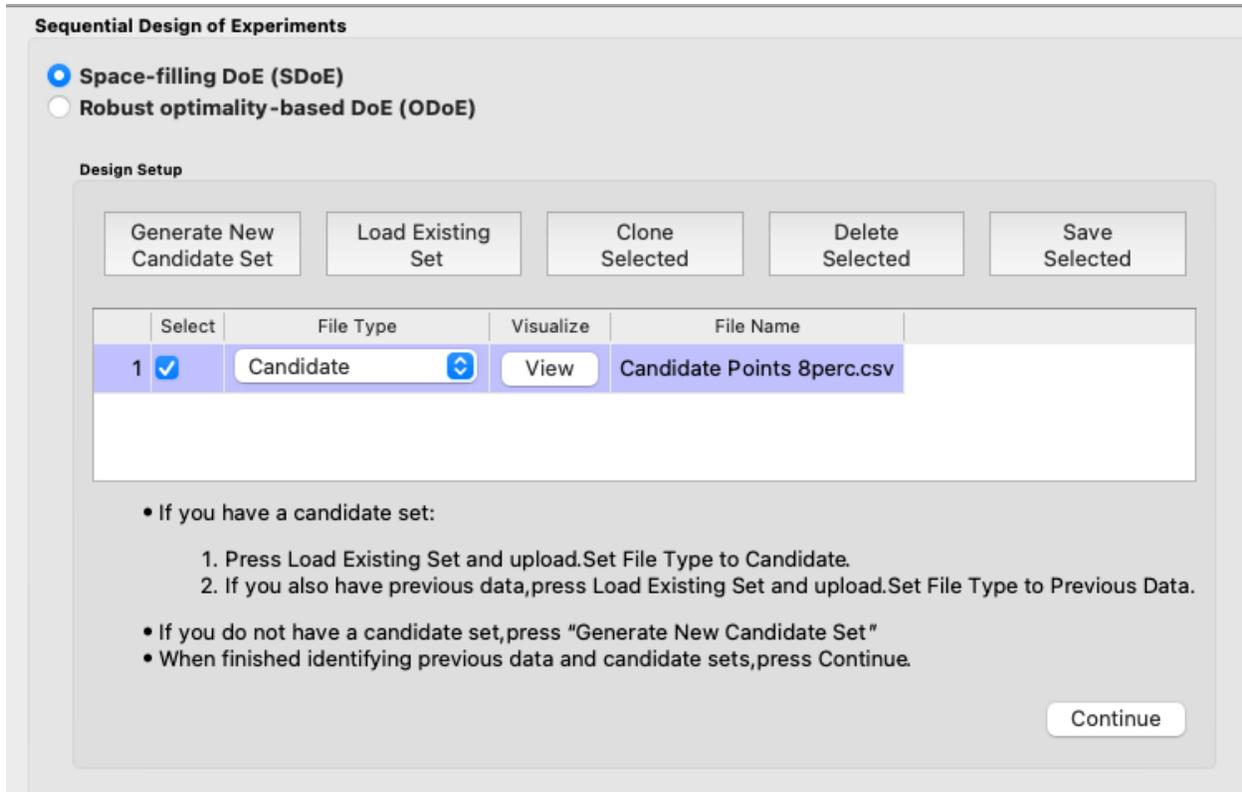


Fig. 47: Ex 3 Design Setup window

do not want to use it in the design construction, and also unclicked the **Include?** box for **Test No.** as we already have an index column with **__id**.

4. After clicking **Estimate Runtime** and selecting the number of random starts to be used, click **Run SDOE**. After the module has created the requested designs, they can be viewed and compared.
5. By clicking **View** and then **Plot**, the designs can be viewed. Suppose that the experimenter decides to use the 12 run design in the initial phase, then this would be the design that would be implemented and data collected for these 12 input combinations.
6. After these runs have been collected, the experimenter wants to collect additional runs. In this case, return to the first SDOE module window, and load in the candidate set (which can be changed to reflect any knowledge gained during the first phase, such as undesirable regions or new combinations to include). The completed experiment should also be included as a **Previous Data** file, by going to the **SDOE_files** folder and selecting the file containing the appropriate design. Note that all candidate file(s) and previous data file(s) used together must contain all the same column names, with the exception of the **__id** column since it is automatically created later on.
7. After clicking **Continue** in the **Design Setup** box, and then **Uniform Space Filling** from the **Design Construction** box, the **Generate Design** box will appear on the right side of the second window. Here, select the options desired for the experiment to be run. For the illustrated figure, we selected a **Minimax** design with a design size of 6 (to use the remaining available budget). We again specified that the column **__id** will be used as the Index, **G, Ildg, CapturePerc, L, SteamFlow** will define the same 5 factors to be used as inputs, and we uncheck the unneeded columns **Test No.** and **CO2 captured**.
8. After clicking **Estimate Runtime** and selecting the number of random starts to be used, click **Run SDOE**. After the module has created the requested design, it can be viewed. After selecting **View** and then **Plot**, the experimenter can see the new design with the previous data runs included. This provides a good plot to allow

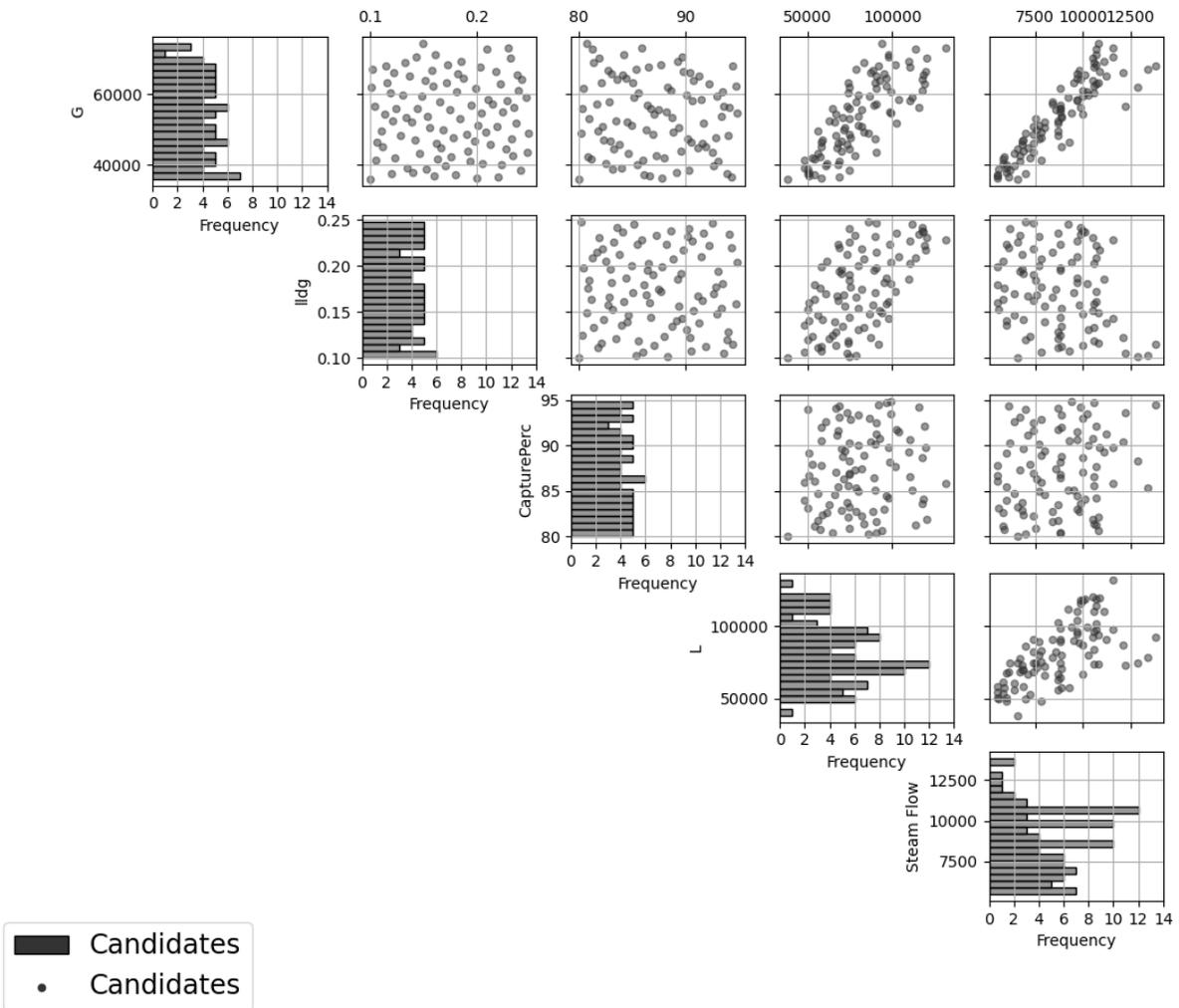


Fig. 48: Ex 3 plot of viable input space as defined by candidate set

Generate Design

Optimality Method Selection

Minimax Maximin

Desired Design Sizes

Min Design Size

Max Design Size

	Include?	Name	Type	Min	Max
1	<input checked="" type="checkbox"/>	__id	Index <input type="button" value="↓"/>	0.0	92.0
2	<input type="checkbox"/>	Test No.	Input <input type="button" value="↓"/>	1.0	100.0
3	<input checked="" type="checkbox"/>	G	Input <input type="button" value="↓"/>	36000.0	74390.62
4	<input checked="" type="checkbox"/>	lldg	Input <input type="button" value="↓"/>	0.1	0.25
5	<input checked="" type="checkbox"/>	CapturePerc	Input <input type="button" value="↓"/>	80.0	94.88
6	<input checked="" type="checkbox"/>	L	Input <input type="button" value="↓"/>	38378.66	131752.23
7	<input checked="" type="checkbox"/>	Steam Flow	Input <input type="button" value="↓"/>	5500.0	13800.0
8	<input type="checkbox"/>	CO2 captured	Input <input type="button" value="↓"/>	3574.0	7966.0

Fig. 49: Ex 3 generate design window for first stage

Created Designs

	Optimality Method	Design Size,d	# of Random Starts,n	Runtime (in sec)	Criterion Value	Plot SDoE
1	minimax	10	100000	83.4	0.06	View
2	minimax	11	100000	86.14	0.06	View
3	minimax	12	100000	85.42	0.07	View

* All design results saved in SDoE output directory

[Go Back to Generate Design](#)

[Order Design](#)

[Delete Design](#)

Fig. 50: Ex 3 10,11,12 run designs created for first stage

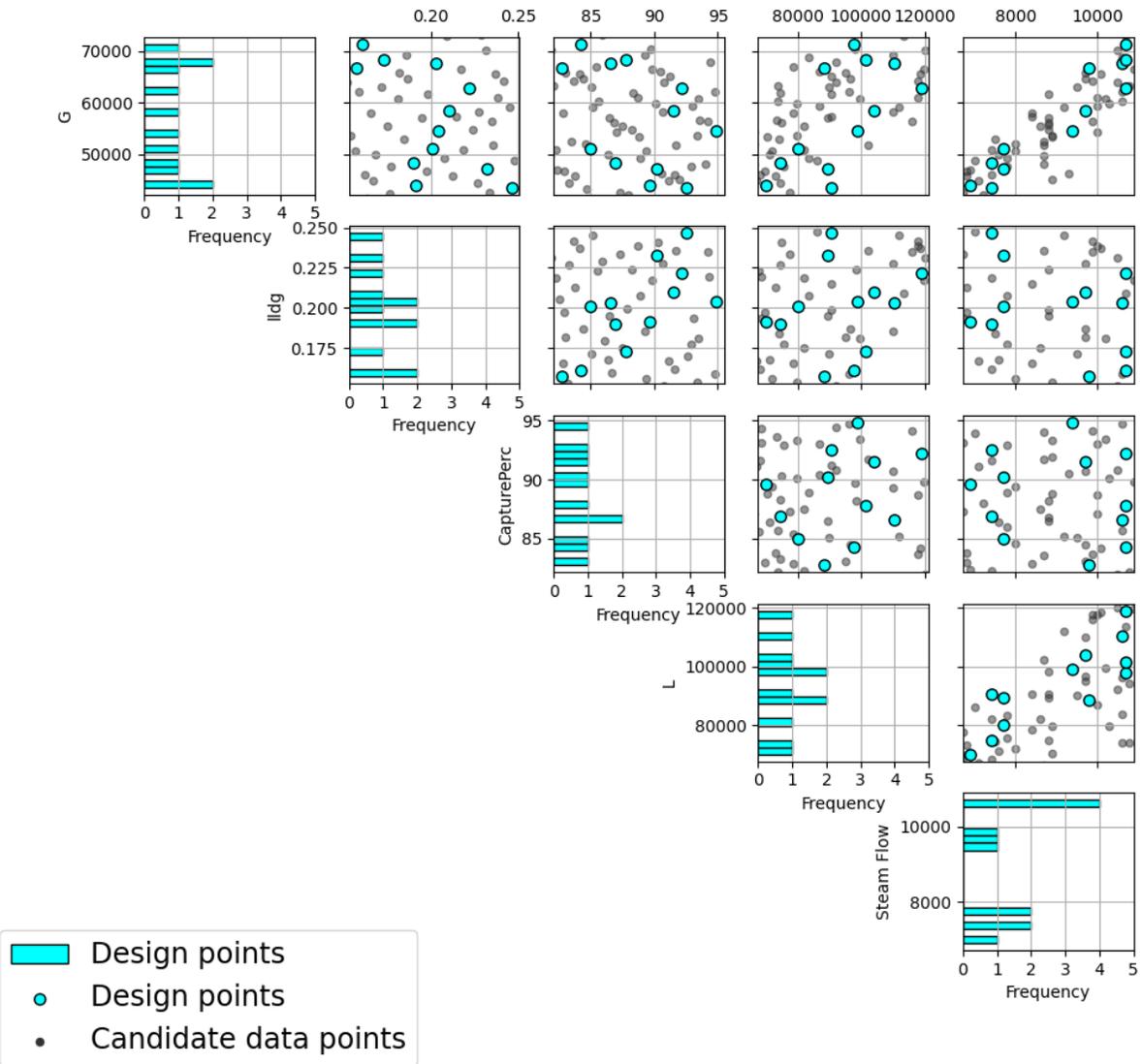


Fig. 51: Ex 3 chosen experiment for first stage

Sequential Design of Experiments

Space-filling DoE (SDoE)
 Robust optimality-based DoE (ODoE)

Design Setup

	Select	File Type	Visualize	File Name
1	<input checked="" type="checkbox"/>	Candidate	<input type="button" value="View"/>	Candidate Points 8perc.csv
2	<input checked="" type="checkbox"/>	Previous Data	<input type="button" value="View"/>	usf_d12_n100000___id+G+Ildg+CapturePerc+L+Steam Flow.csv

- If you have a candidate set:
 1. Press Load Existing Set and upload.Set File Type to Candidate.
 2. If you also have previous data,press Load Existing Set and upload.Set File Type to Previous Data.
- If you do not have a candidate set,press "Generate New Candidate Set"
- When finished identifying previous data and candidate sets,press Continue.

Fig. 52: Ex 3 design setup box for second stage

the complete sequence of two experiments to be examined as a combined set of runs. Note that the first and second stages are shown in different colors. The first stage is shown in pink while the second is shown in blue. Candidate points not part of either stage are in gray.

Example NUSF-1: Constructing Non-Uniform Space Filling maximin designs for a 2-D input space

For this first Non-Uniform Space Filling design example, the goal is to construct a non-uniform space-filling design with 20 runs in a 2-dimensional space based on a regular unconstrained square region populated with a grid of candidate points. The choice of how to construct the candidate set should be based on: a) what is the precision with which each of the inputs can be set in the experiment, and b) timing for generating the designs. Note that the finer the grid that is provided in the candidate set, the longer the search algorithm will take to run for a given number of random starts. In general a finer grid will give better options for the best design, but with diminishing returns after a large number of candidates have already been provided

As noted previously in the Basics section, in addition to specifying the candidate point input combinations, it is also required to supply an additional column of weights. This column will provide the necessary information about which regions of the input space should be emphasized more, and which should be emphasized less. The figure below shows some of the characteristics of the candidate set.

The candidates are laid out in a regular grid with equal spacing between levels of each of X1 and X2. A contour plot of the weight function that was used to generate the weights is shown on the left side of the plot. The weights range from -14.48 to 50, with the largest values of the weights near the top left corner of the input space. The smallest values lie in the bottom right corner. On the right hand side, we can see a plot where the relative size of the points is proportionate to the size of the weight assigned to that candidate point. This second representation is helpful when the candidate points do not fall on a regular grid, or if the relationship for determining the weights is not smooth.

Here is the process for generating NUSF designs for this problem:

Generate Design

Optimality Method Selection

Minimax Maximin

Desired Design Sizes

Min Design Size

Max Design Size

	Include?	Name	Type	Min	Max
1	<input checked="" type="checkbox"/>	__id	Index <input type="button" value="↓"/>	0.0	92.0
2	<input type="checkbox"/>	Test No.	Input <input type="button" value="↓"/>	1.0	100.0
3	<input checked="" type="checkbox"/>	G	Input <input type="button" value="↓"/>	36000.0	74390.62
4	<input checked="" type="checkbox"/>	lldg	Input <input type="button" value="↓"/>	0.1	0.25
5	<input checked="" type="checkbox"/>	CapturePerc	Input <input type="button" value="↓"/>	80.0	94.88
6	<input checked="" type="checkbox"/>	L	Input <input type="button" value="↓"/>	38378.66	131752.23
7	<input checked="" type="checkbox"/>	Steam Flow	Input <input type="button" value="↓"/>	5500.0	13800.0
8	<input type="checkbox"/>	CO2 captured	Input <input type="button" value="↓"/>	3574.0	7966.0

Fig. 53: Ex 3 generate design box for second stage

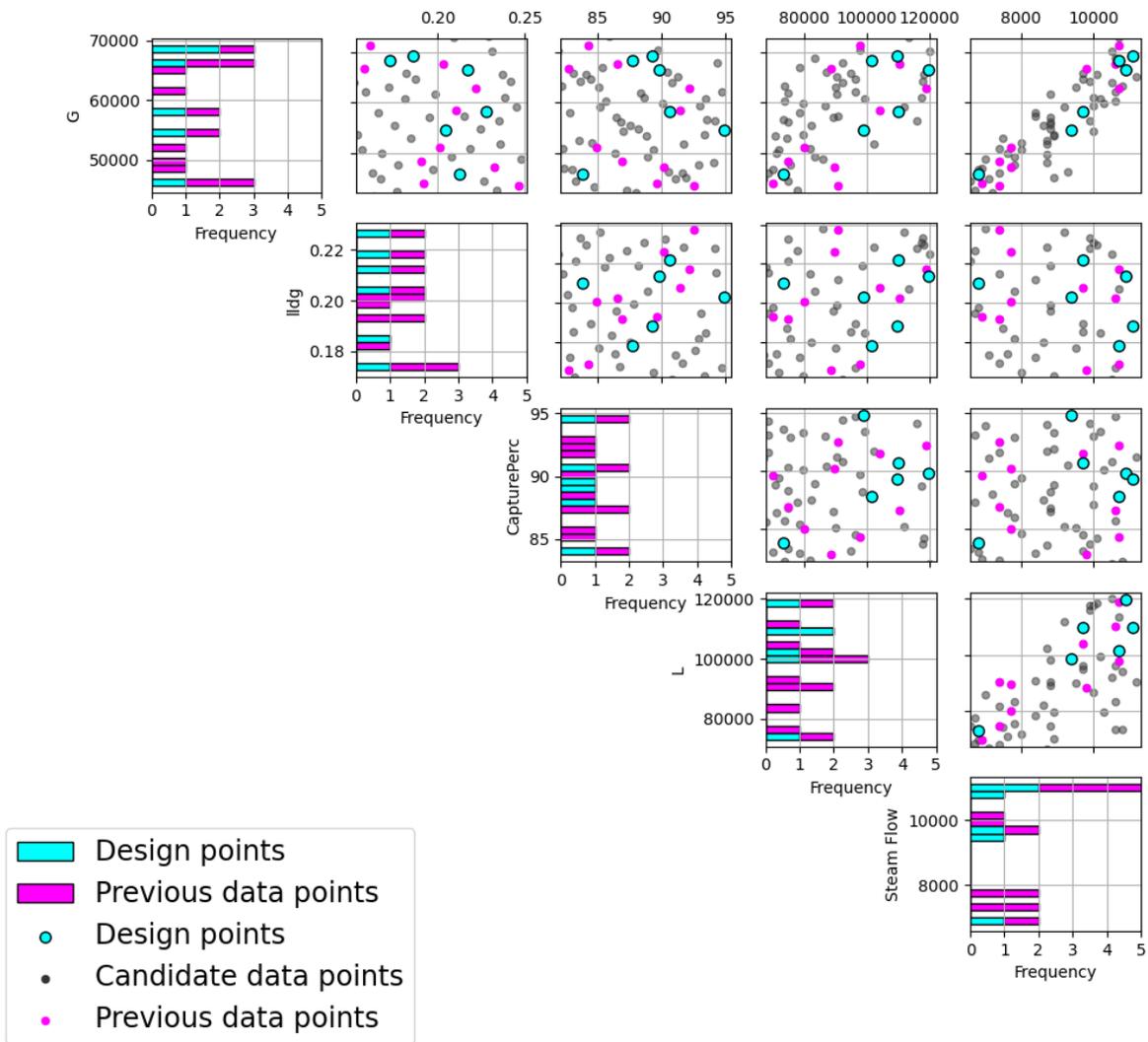


Fig. 54: Ex 3 chosen experiment for second stage

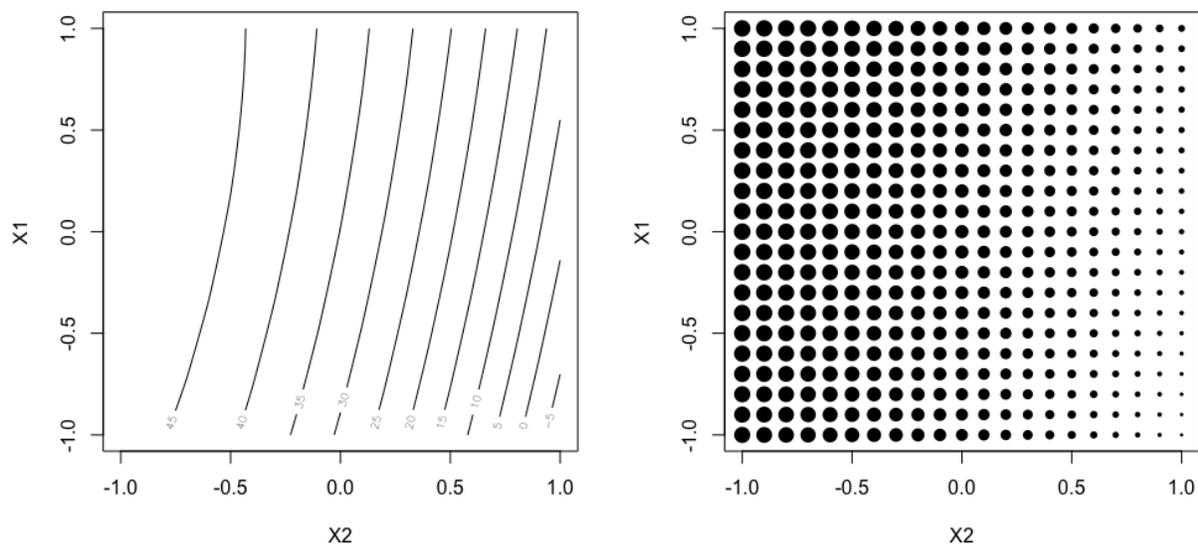


Fig. 55: Ex NUSF1 Candidate set of points with their associated weights. Left shows the underlying relationship used to generate the design, and right shows the candidates with the size of the point proportional to the assigned weight.

1. From the FOQUS main screen, click the **SDOE** button. On the top left side, select **Load Existing Set**, and select the “NUSFex1.csv” file from examples folder.
2. Next, by selecting **View** and then **Plot** it is possible to see the grid of points that will be used as the candidate points. In this case, the range for each of the inputs, X1 and X2, has been chosen to be between -1 and 1.
3. Next, click on **Continue** to advance to the **Design Construction** Window, and the click on **Non-Uniform Space Filling** to advance to the second SDOE screen, where particular choices about the design can be made. On the second screen, the first choice for **Optimality Method Selection** is automatic, since the non-uniform space filling designs only use the **Maximin** criterion. The next choice is to choose the **Scaling Method**, where the choices are **Direct** and **Ranked**. The default is to use the Direct scaling which translates the weights provided with a linear transformation so that they lie in the range 1 to whatever **MWR** value is selected below. For this example, we choose the option for Direct scaling.

Next select the **Design size**, where here we have decided to construct a design with 20 runs. The choice of the **Maximum Weight Ratio** or **MWR** is one of the more difficult choices that the experimenter will need to make, since it is often one that they do not have much experience with. It is for this reason that we recommend constructing several designs with different MWR values and then comparing the results to see which value is best suited for the experiment to be run. Recall that a value of 1 corresponds to a uniform space filling design, while larger values will place increasing concentration of points near the regions with larger weight values. In this case, we select to generate 3 designs, with **MWR** values of 5, 10 and 30. This should give a good variety of designs to choose from after they have been constructed.

There are also choices for which columns to include in the analysis. Here we use all 4 columns for creating the design, so all **Include?** boxes remain checked. The **__id** column is automatically created and we will use it as the index column here. In addition, it is possible to see the range of values for each of the columns in the spreadsheet. Here the two input columns range from -1 to 1, while the “RawWt” column ranges from -8 to 50. The user can change these values if they wish to rescale the ranges to widen or narrow them, but in general these values can be left as is.

4. Once the choices for the design have been specified, click on the **Estimate Runtime** button to estimate the time taken for creating the designs. For the computer on which this example was developed, if we ran 30 random

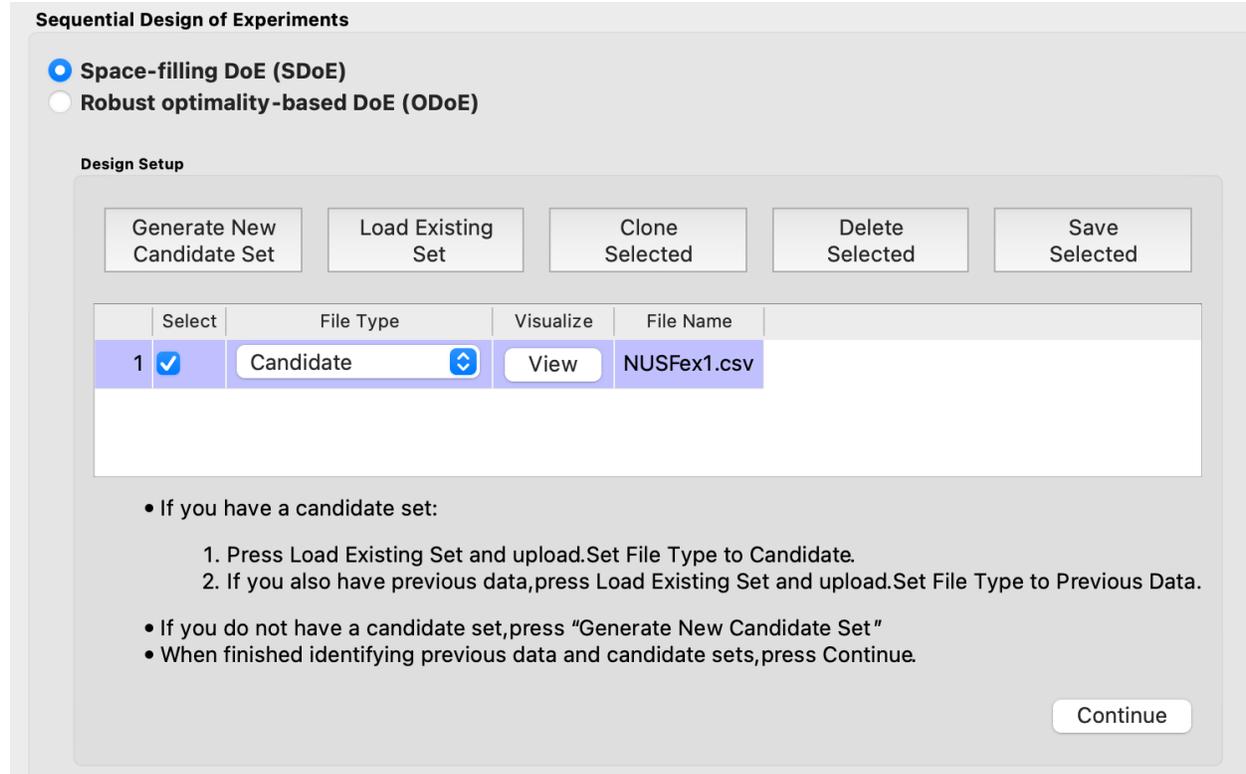


Fig. 56: Ex NUSF1 choice of file for candidate set

starts, it is estimated that the algorithm would take 10:08 minutes to generate the 3 designs with MWR values of 5, 10, 30. Note that the timing changes linearly, so using 20 random starts would take twice as long as using 10 random starts. Recall that the choice of the number of random starts involves a trade-off between getting the designs created quickly and the quality of the designs. For many applications, we would expect that using at least 30 random starts would produce designs that are of good quality.

5. Once the algorithm has generated the designs, the left box called **Created Designs** populates with the 3 designs that we have created. Some of the key choices of the designs are summarized in the columns. The size of the design, the MWR value and the number of random starts are all noted. In addition, the time to create the design is also included. The criterion value is provided. Recall from the discussion in the Basics section, that the criterion value can be compared for designs of the same size and with the same MWR value, but should not be compared across design sizes or across different MWR values.
6. To examine each of the created designs, select **View** and choose the columns to be included, and click **Plot**. For this example we included all of the columns except the index column `__id`. Note that two plots are created for each design. The first is the **Closest Distance by Weight (CDBW) plot**, and the second is the more familiar **pairwise scatterplot** of the created design.

First, we describe the information that is contained in the CDBW plot. There are two portions to the plot. The lower section shows a histogram of the weights in the candidate set. Note that the range of values goes from 1 to the MWR value selected. For the figure below, we are looking at a design created with a MWR value of 5. The shape of the histogram shows what values were available to be selected from. The top portion of the plot, has a vertical line for each of the design points selected (in this case 20 vertical lines for 20 design points). The location of each vertical line shows the weight for the selected design point. In this case, the smallest weight selected had weight around a value of 2, while there are several design points chosen that have weight close to the maximum possible (the MWR value). This allows the user to see how much emphasis was placed on getting the larger weight values into the design.

The second plot is the more familiar scatterplot of the design points. It is clear that the non-uniform space filling

Sequential Design of Experiments

Generate Design

Optimality Method Selection

Minimax Maximin

Scaling Method

Direct Maximum Weight Ratio Ranked Maximum Weight Ratio

Desired Design Size and NUSF Weights

Design Size: 20 MWR: 5 10 30

Create up to 5 different designs using different maximum weight ratios (MWRs).

	Include?	Name	Type	Min	Max
1	<input checked="" type="checkbox"/>	__id	Index	0.0	440.0
2	<input checked="" type="checkbox"/>	X1	Input	-1.0	1.0
3	<input checked="" type="checkbox"/>	X2	Input	-1.0	1.0
4	<input checked="" type="checkbox"/>	RawWT	Weight	-8.01	49.54

This performs a small number of iterations of the search algorithm to estimate timing for constructing the designs.

Fig. 57: Ex NUSF1 Choice of settings for generating NUSF designs

SDoE Progress

Number of Random Starts: n = 30

Estimated Runtime: 10:08

mwr = 5, n = 30

Fig. 58: Ex NUSF1 specification of timing to generate the requested designs.

Input Summary

# Inputs	4
Candidate File	aggregate_candidates.csv
Previous Data File	None
Output Directory	/Users/mgmartin/Desktop/FOQUS/SDOE_files

Created Designs

	MWR Value	Design Size, d	# of Random Starts, n	Runtime (in sec)	Criterion Value	Plot SDOE
1	5	20	30	199.99	0.91	View
2	10	20	30	194.82	3.24	View
3	30	20	30	192.6	26.43	View

* All design results saved in SDOE output directory

[Go Back to Generate Design](#)

[Order Design](#)

[Delete Design](#)

Fig. 59: Ex NUSF1 created designs for three MWR values of 5, 10 and 30

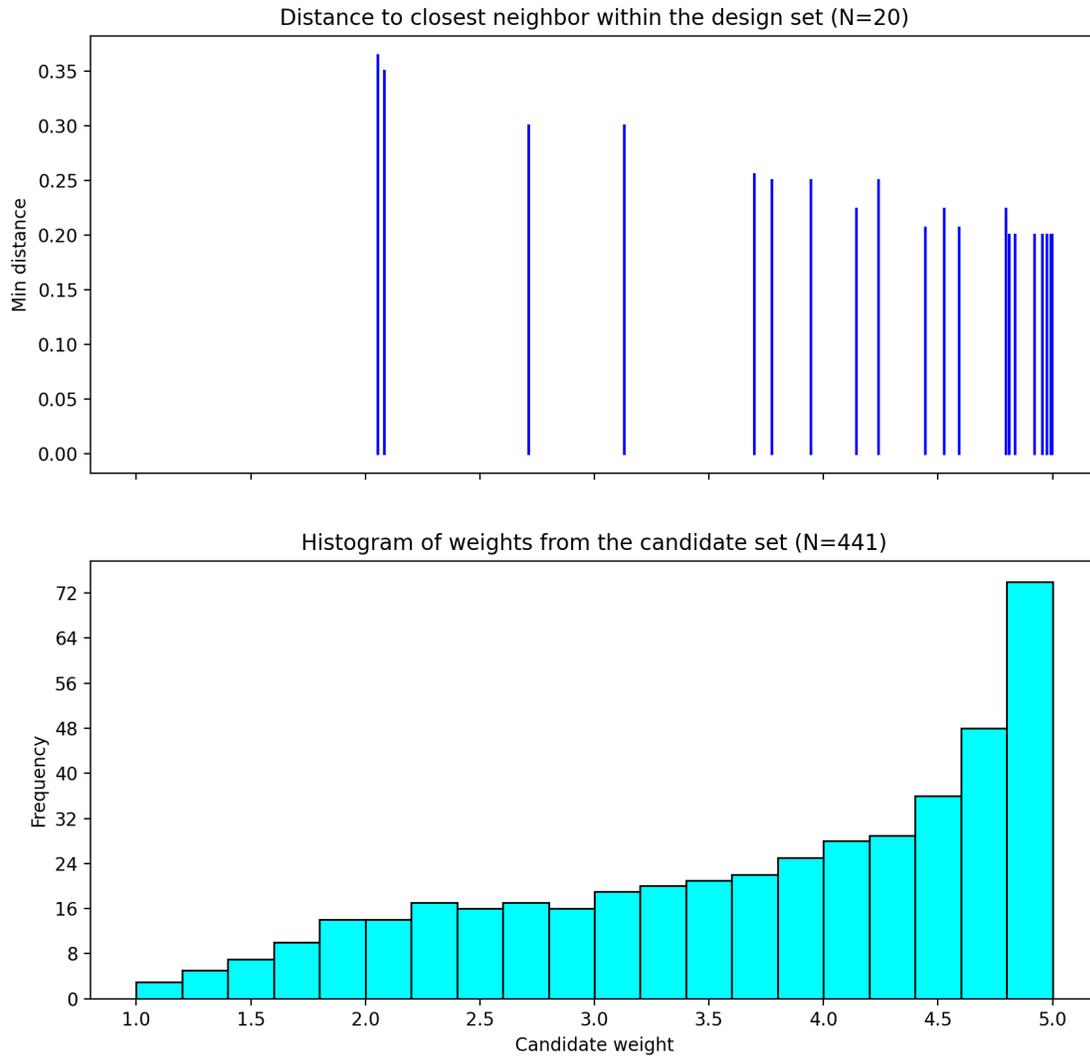


Fig. 60: Ex NUSF1 Closest Distance by Weight (CDBW) plot for the constructed design with MWR values of 5

approach has lived up to its name and has generated a design that has a greater emphasis of points for the larger weights. The design still provides space filling throughout the region, but with very different densities of points for the various regions.

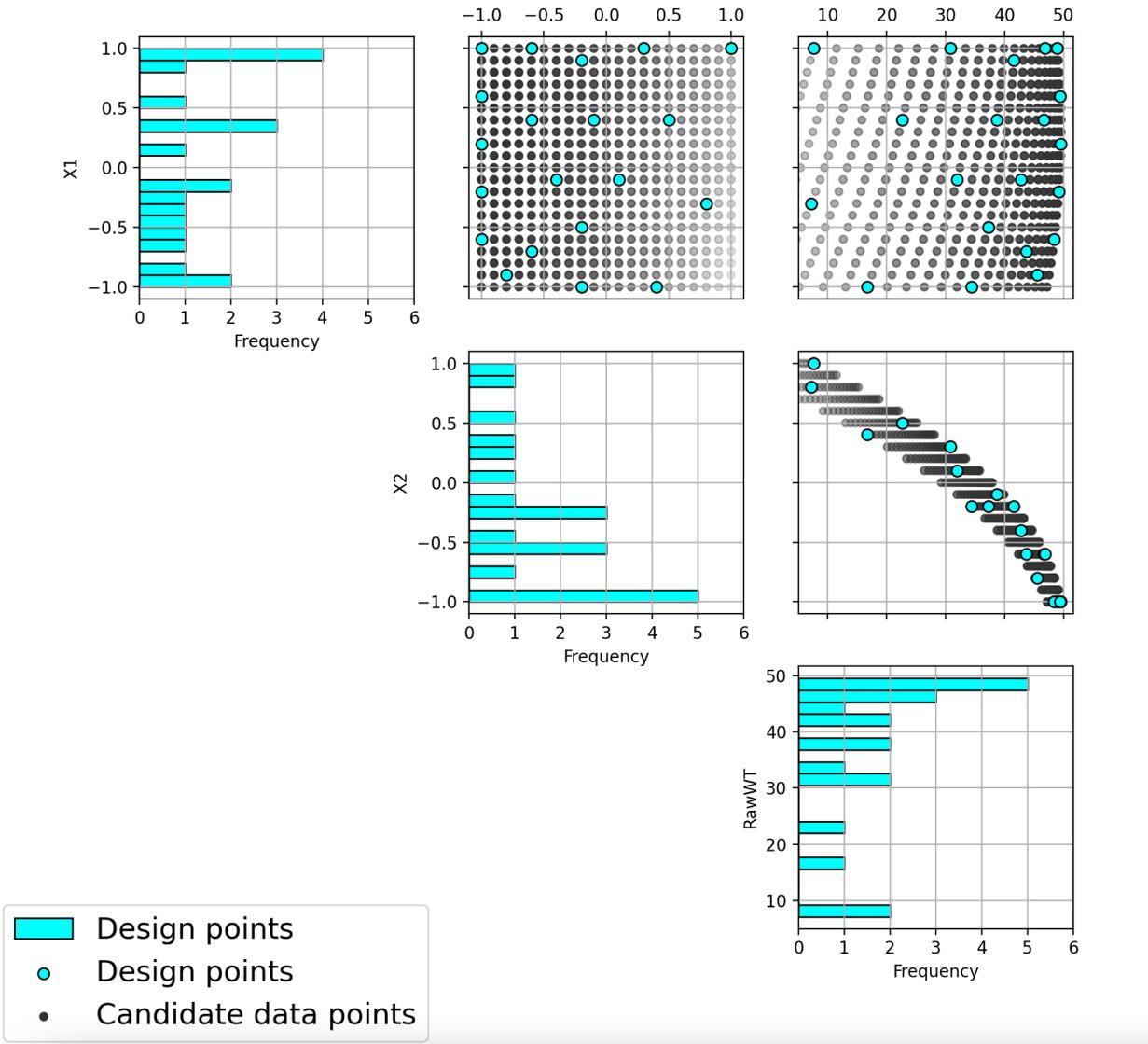


Fig. 61: Ex NUSF1 pairwise scatterplot for the constructed design with MWR values of 5

- The next step is to repeat the process for the other two designs created. In this case we can see that the NUSF designs for MWR values of 10 and 30 create even more concentrated designs in the region with higher weights. The figure below shows the collection of the CDBW plot for MWR values of 10 and 30.

When we compare the three CDBW plots for the designs with MWR of 5, 10 and 30, we see that more of the points are shifted to the right closer to the maximum weight value as we increase the MWR value. This gives control to the user to adjust the relative density of points for different weights.

When we compare the three designs, we can see that increasing the **MWR** produces a design that moves more of the points closer to the higher weight regions of the input space. This gives the user the control that is needed to create a customized design that matches the desired concentration of points in the regions where they are desired. After

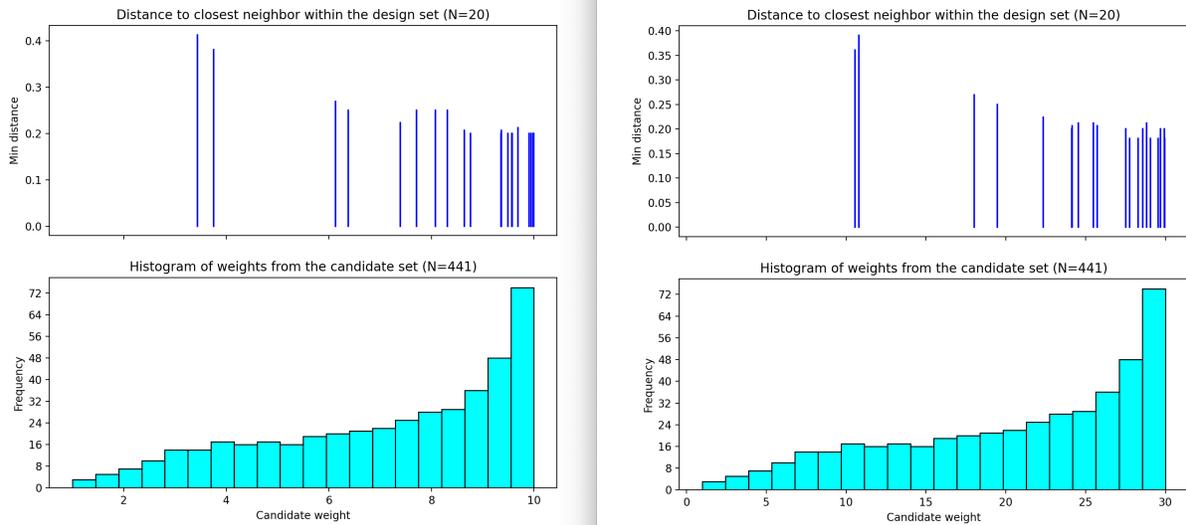


Fig. 62: Ex NUSF1 Closest Distance by Weight (CDBW) plot for the constructed designs with MWR values of 10 and 30

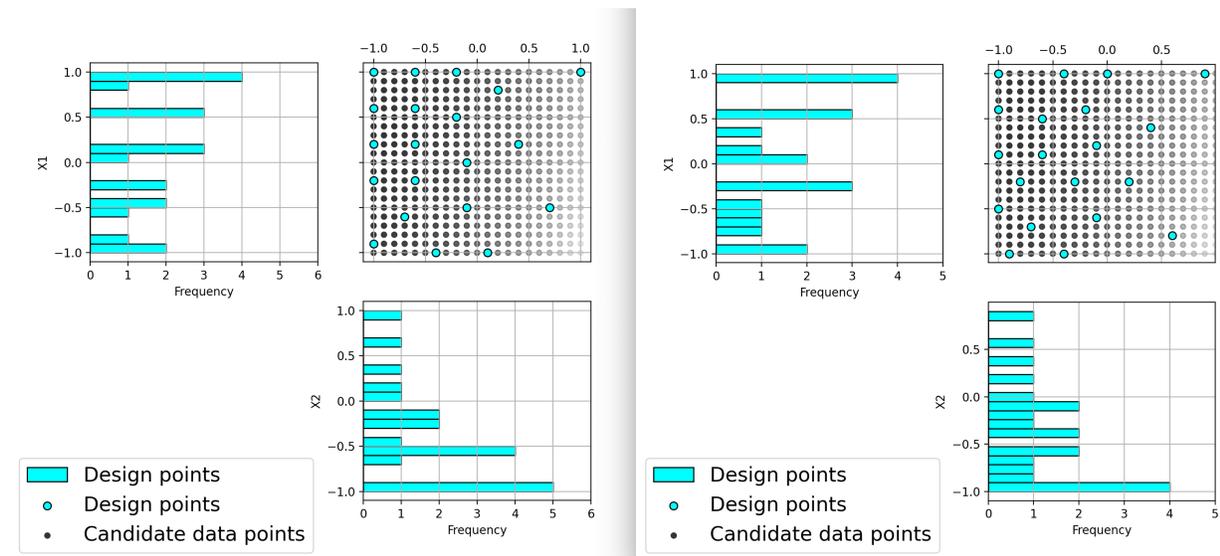


Fig. 63: Ex NUSF1 pairwise scatterplot for the constructed designs with MWR values of 10 and 30

examining the different summary plots for the three designs, the user can choose the plot that is the best match to their experimental needs.

Example NUSF-2: Constructing Non-Uniform Space Filling for a 4-Input Carbon Capture example

For this second Non-Uniform Space Filling design example, we consider a carbon capture example with 4 inputs (G, lldg, w, L). In this case the experimenter is interested in constructing a 10 run design that is space filling, but also places a slightly higher emphasis in the region that is expected to contain the optimum of the process. The experimenter's team of experts identify that the most likely location for that optimum is located a $G=2200$, $lldg=0.2$, $w=0.15$ and $L=8000$. As such they construct a set of weights that are highest at this location in the input space, and then taper away the further the inputs are from that optimum. The figure below shows the set of 526 candidate points that take into account the constraints in the region, where running an experiment at those locations would not yield a desirable outcome or perhaps would not even generate any response. The red triangle indicates the identified likely optimum for all pairwise scatterplots above the diagonal. The size of the symbols is scaled to be proportional to the weights at each location, with largest points near the optimum and tapering away as we move to the extremes of the input space.

Here is the process for generating NUSF designs for this problem:

1. From the FOQUS main screen, click the **SDOE** button. On the top left side, select **Load from File**, and select the "CCSIex.csv" file from examples folder.
2. Next, by selecting **View** and then **Plot** it is possible to see the pairwise scatterplot of all of candidate points. Note that in this file there are 6 columns - the Label column will be used to identify which of the candidates are selected in the constructed designs. The Weights column summarizes how desirable a candidate point is by its proximity to the anticipated optimum location.
3. Next, click on **Confirm** to advance to the **Ensemble Aggregation** Window, and then click on **Non-Uniform Space Filling** to advance to the second SDOE screen, where particular choices about the design can be made. On the second screen, the first choice for **Optimality Method Selection** is automatic, since the non-uniform space filling designs only use the **Maximin** criterion.

The next choice is to choose the **Scaling Method**, where the choices are **Direct** and **Ranked**. The default is to use the Direct scaling which translates the weights provided with a linear transformation so that they lie in the range 1 to whatever **MWR** value is selected below. For this example, we will explore what difference the choice of the scaling method makes on the resulting designs, but begin by choosing the option for Direct scaling.

Next select the **Design size**, where here we have decided to construct a design with 10 runs. The choice of the **Maximum Weight Ratio** or **MWR** for this example reflects that we wish to have a design that is still space filling throughout the input region, but with a slightly emphasized concentration near the anticipated optimum. Hence we will select small values that are not too far away from 1 (which represents a uniform space filling design). Because, it is not always easy to judge the impact of the choice of MWR value, we recommend constructing several designs with different MWR values and then comparing the results to see which value is best suited for the experiment to be run.

In this case, we select to generate 2 designs, with **MWR** values of 2 and 5. This should provide some variety of designs to choose from after they have been constructed.

4. Once the choices for the design have been specified, click on the **Test SDOE** button to estimate the time taken for creating the designs. For the computer on which this example was developed, if we ran 30 random starts, it is estimated that the algorithm would take 15:08 minutes to generate the 2 designs with MWR values of 2 and 5. Note that the timing changes linearly, so using 40 random starts would take twice as long as using 20 random starts. Recall that the choice of the number of random starts involves a trade-off between getting the designs created quickly and the quality of the designs. For many applications, we would expect that using at least 30 random starts would produce designs that are of sufficient quality.
5. Once the algorithm has generated the designs, the left box called **Created Designs** populates with the 2 designs that we have created. Some of the key choices made by the experimenter for the designs are summarized in the columns. The size of the design, the MWR value and the number of random starts are all noted. In addition,

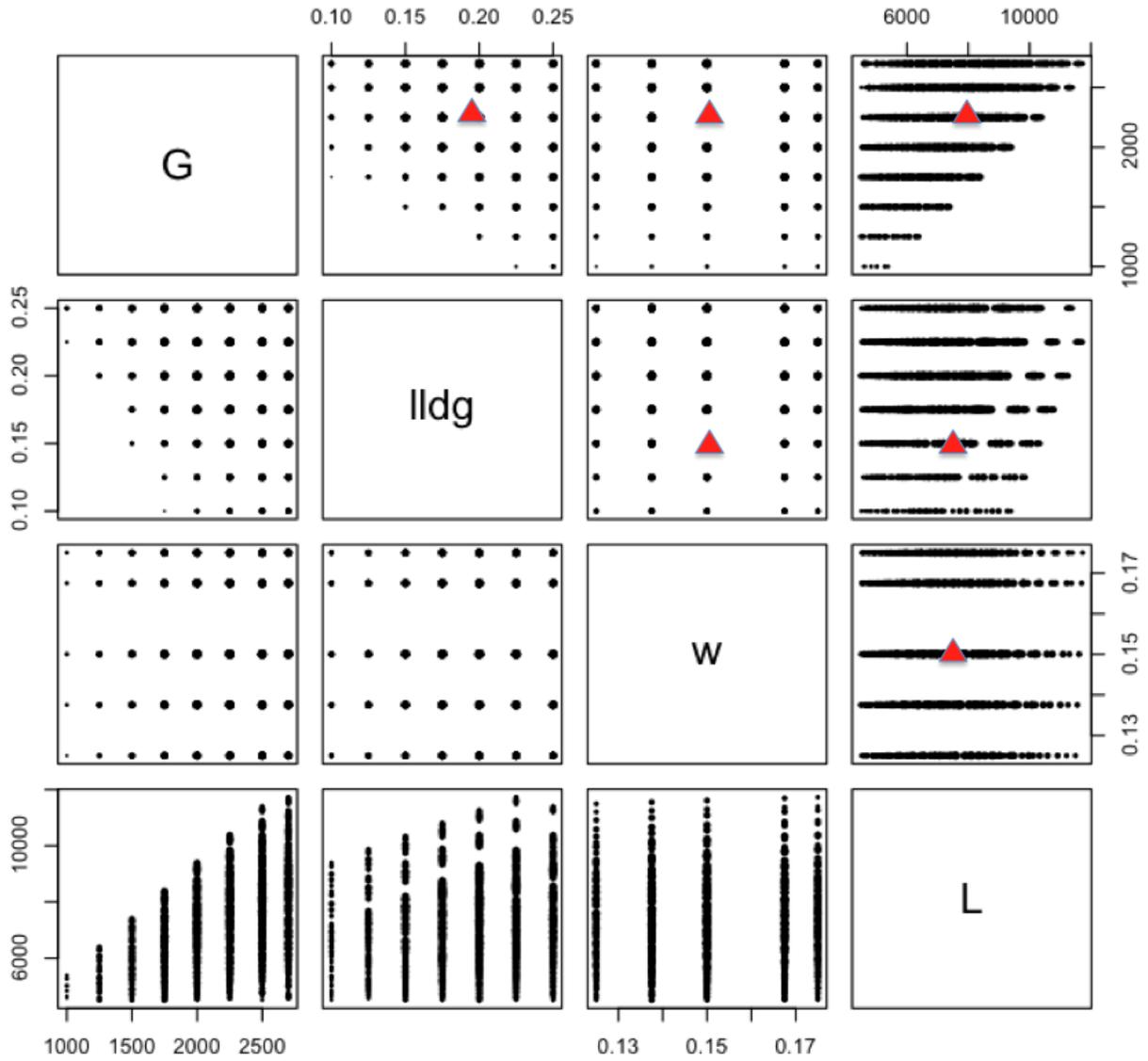


Fig. 64: Example NUSF2 pairwise scatterplot of the candidate set with the anticipated optimum location shown with red triangles

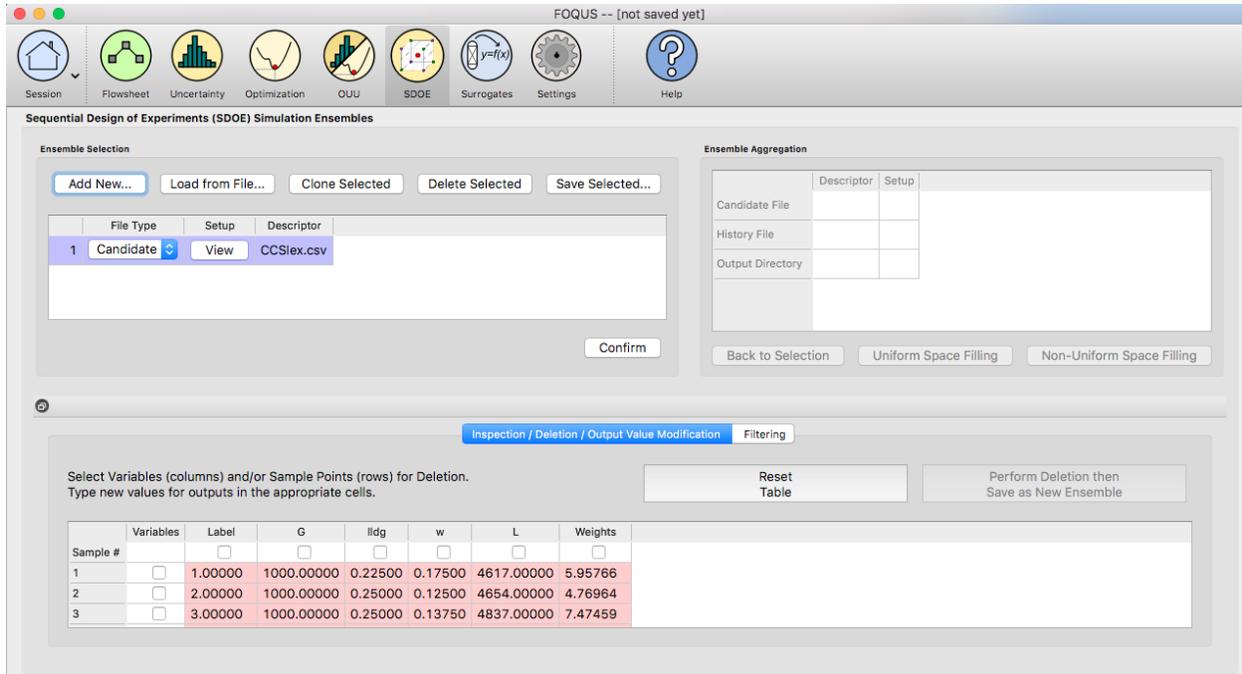


Fig. 65: Example NUSF2 choice of file for candidate set

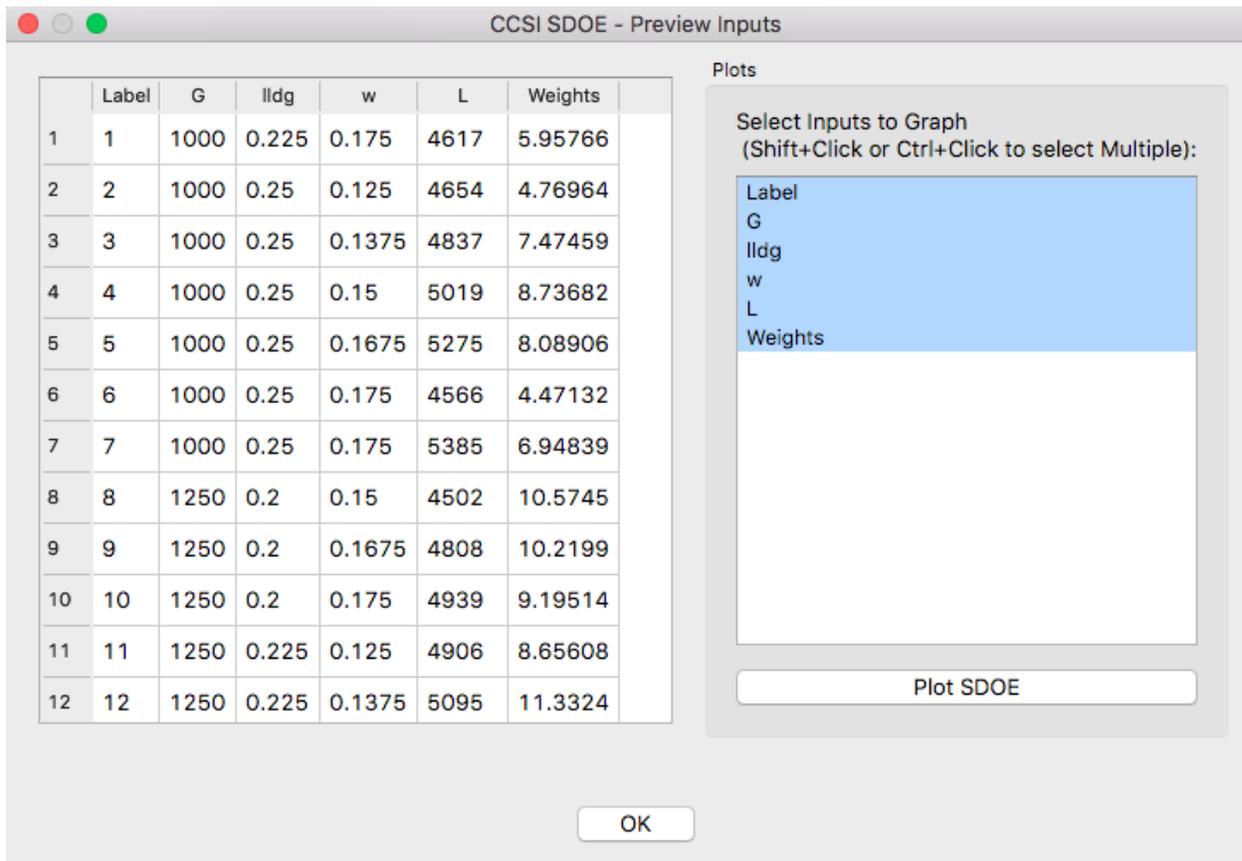


Fig. 66: Example NUSF2 top of file with candidate points

Sequential Design of Experiments (SDOE) Set-up

Optimality Method Selection

Minimax Maximin

Scaling Method

Direct MWR Ranked MWR

Inputs

Design Specification

Design Size MWR:

	Include?	Name	Type	Min	Max
1	<input checked="" type="checkbox"/>	Label	Index	1.0	546.0
2	<input checked="" type="checkbox"/>	G	Input	1000.0	2700.0
3	<input checked="" type="checkbox"/>	lldg	Input	0.1	0.25
4	<input checked="" type="checkbox"/>	w	Input	0.12	0.18
5	<input checked="" type="checkbox"/>	L	Input	4502.0	11727.0
6	<input checked="" type="checkbox"/>	Weights	Weight	3.43	19.87

Fig. 67: Ex NUSF2 Choice of settings for generating NUSF designs

SDOE Progress

Number of Random Starts: n =

Estimated Runtime: 15:08

mwr = 2, n = 30

Fig. 68: Ex NUSF2 specification of timing to generate the requested designs.

the time to create the design is also included. The criterion value is provided. Recall from the discussion in the Basics section, that the criterion value can be compared for designs of the same size and with the same MWR value, but should not be compared across design sizes or across different MWR values.

- To examine each of the created designs, select **View** and choose the columns to be included, and click **Plot**. For this example we included only the 4 input columns to keep each plot to a moderate size. Note that two plots are created for each design. The first is the **Closest Distance by Weight (CDBW) plot**, and the second is the more familiar **pairwise scatterplot** of the created design.

Recall that there are two portions to the CDBW plot. The lower section shows a histogram of the weights in the candidate set. Note that the range of values goes from 1 to the MWR value selected. For the figure below, we are looking at a design created with a MWR value of 2. The shape of the histogram shows what values were available to be selected from. The top portion of the plot, has a vertical line for each of the design points selected (in this case 10 vertical lines for 10 design points). The location of each vertical line shows the weight for the selected design point.

This allows the user to see how much emphasis was placed on getting the larger weight values into the design.

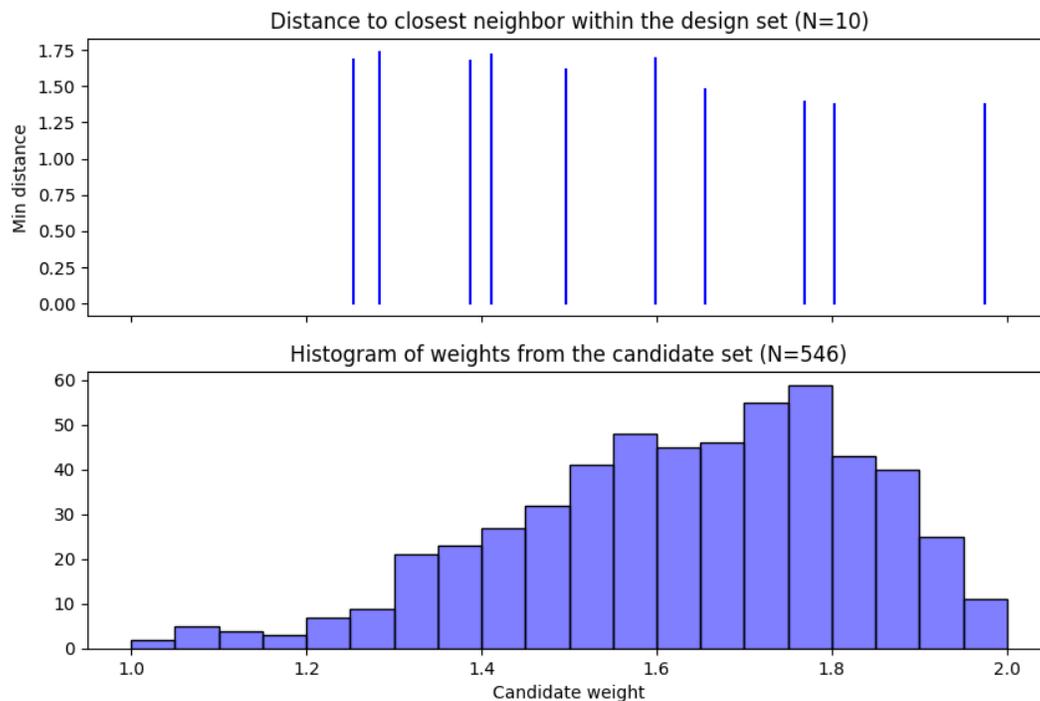


Fig. 69: Ex NUSF2 Closest Distance by Weight (CDBW) plot for the constructed 10 run design with MWR values of 2

In looking at the location of the vertical lines in the top of the CDBW plot, we see that some locations in the input space have been chosen across the majority of the range of the weight values. This reflects the relatively small MWR of 2 value that was selected.

The second plot is the more familiar scatterplot of the design points. This shows the location of the 10 selected design points in the 4 dimensional input space. The points look to cover much the same region as the overall candidate points, but with a slight concentration of points closer to the anticipated optimum.

- Next we consider, reproducing the same designs, but now selecting the **Ranked** scaling option to see how this changes the results of the constructed design. We repeat the early steps for the SDoE module with the same file for the candidate set, “CCSIex.csv”, and all of the choices for the design the same, except this time we choose

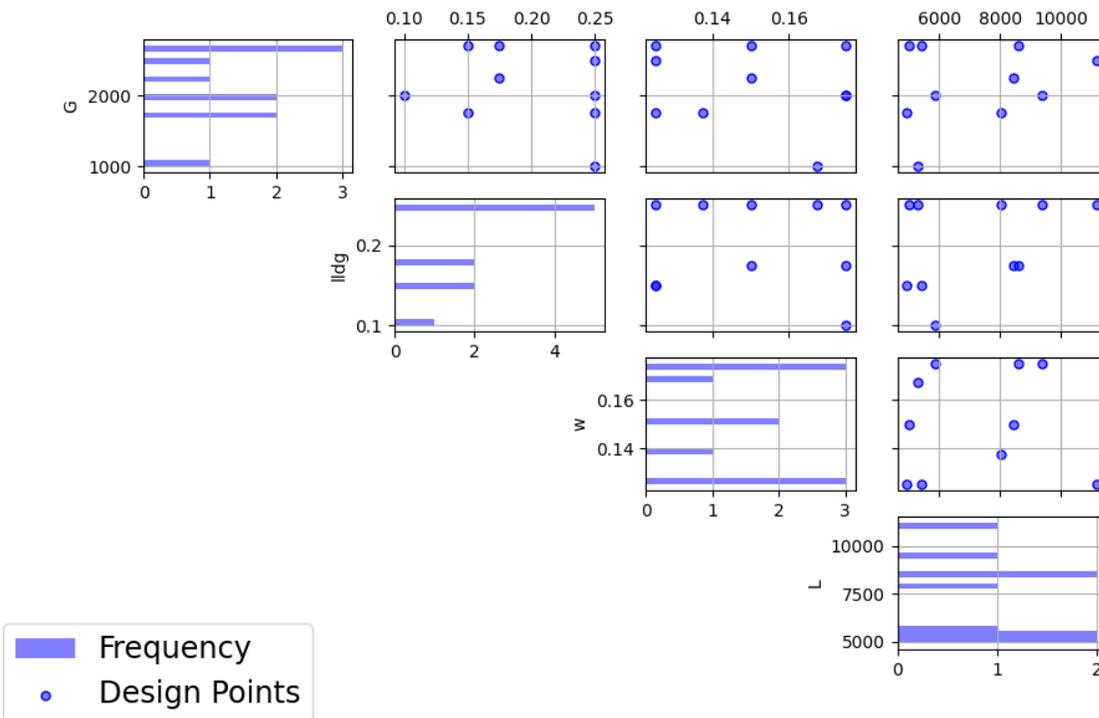


Fig. 70: Ex NUSF2 pairwise scatterplot for the constructed 10 run design with MWR values of 2

the **Scaling Method**, as **Ranked**.

We again construct designs with MWR values of 2 and 5. The time required to generate these designs will be approximately the same as for the other choice of scaling method.

To compare the designs, we can examine the CDBW plots for all 4 of the constructed designs. The figure below shows the CDBW plots for all 4 designs.

To understand the differences between the choices, we note the following points. (a) Not that for the top row of CDBW plots for those associated with the Direct scaling, the shape of the histograms for the candidate set are the same as for the original unscaled weights provided in the candidate set. In this case, we have a skewed distribution with very few small weights. (b) In contrast, the bottom row of CDBW plots are for the Ranked scaling, and the shape of the histogram is quite different from what was obtained with the Direct weighting. As is typical of the the Ranked scaling, we obtain an even histogram with nearly the same count in each bar. (c) Next when we compare the left (MWR=2) and right (MWR=5) plots, we see that the left plots have a more evenly spread set of weights selected across the entire range of values. For the MWR=5 plots, we see that there is a greater concentration of larger weights that have been selected. (d) To select the design that is best suited for the goal of the experiment, it is helpful to think about how non-uniform the spread of points should be, and how big are the gaps where no runs will be collected. The 4 sets of pairwise scatterplots can help to see where the gaps exist. The scatterplots are slightly harder to interpret as the number of factors increases, but the histograms for each input can give a good idea of how the runs are spread across the range of each input.

Sequential Design of Experiments (SDOE) Set-up

Optimality Method Selection

Minimax Maximin

Scaling Method

Direct MWR Ranked MWR

Inputs

Design Specification

Design Size MWR:

	Include?	Name	Type	Min	Max
1	<input checked="" type="checkbox"/>	Label	Index	1.0	546.0
2	<input checked="" type="checkbox"/>	G	Input	1000.0	2700.0
3	<input checked="" type="checkbox"/>	lldg	Input	0.1	0.25
4	<input checked="" type="checkbox"/>	w	Input	0.12	0.18
5	<input checked="" type="checkbox"/>	L	Input	4502.0	11727.0
6	<input checked="" type="checkbox"/>	Weights	Weight	3.43	19.87

Fig. 71: Ex NUSF2 Choice of settings for generating NUSF designs with Ranked for the Scaline Method

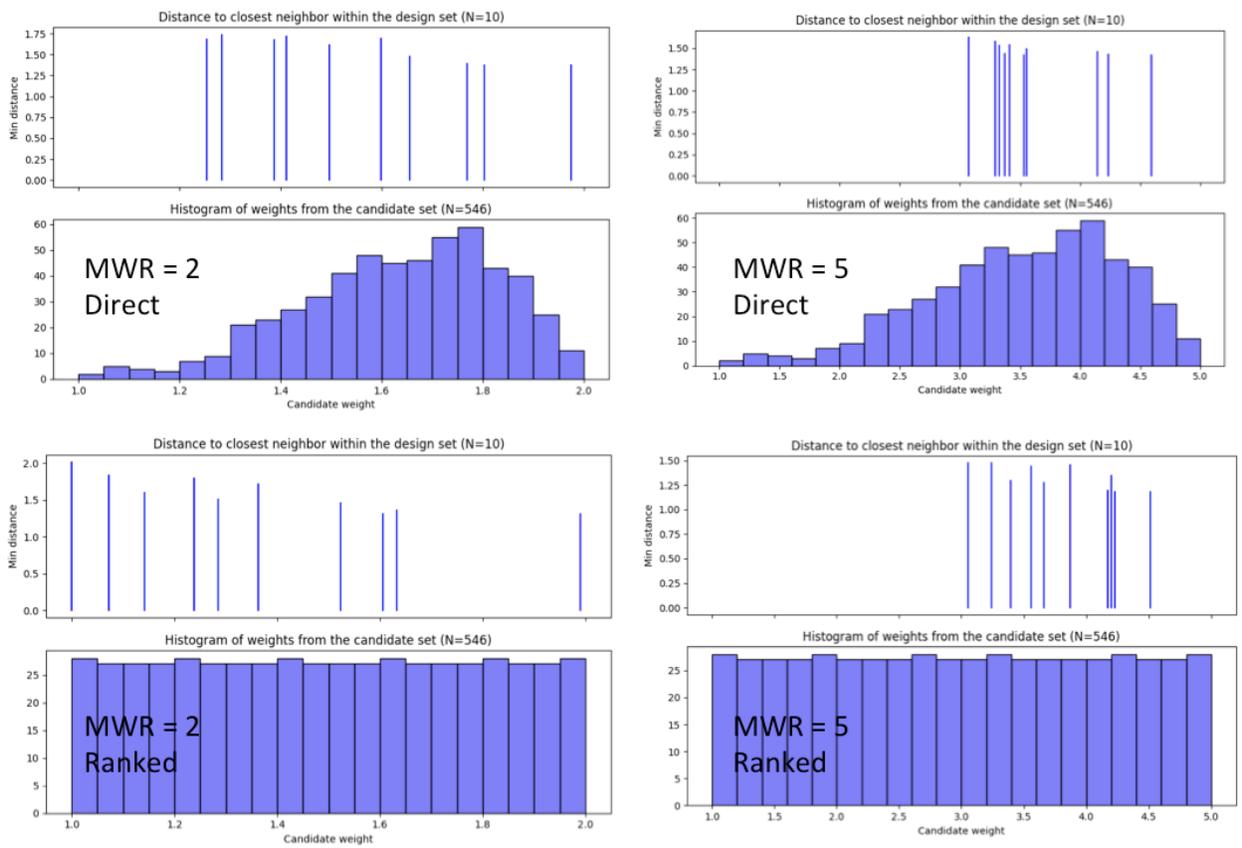


Fig. 72: Ex NUSF2 Comparison of the CDBW plots for designs with MWR values of 2 and 5 with both Direct and Ranked scaling

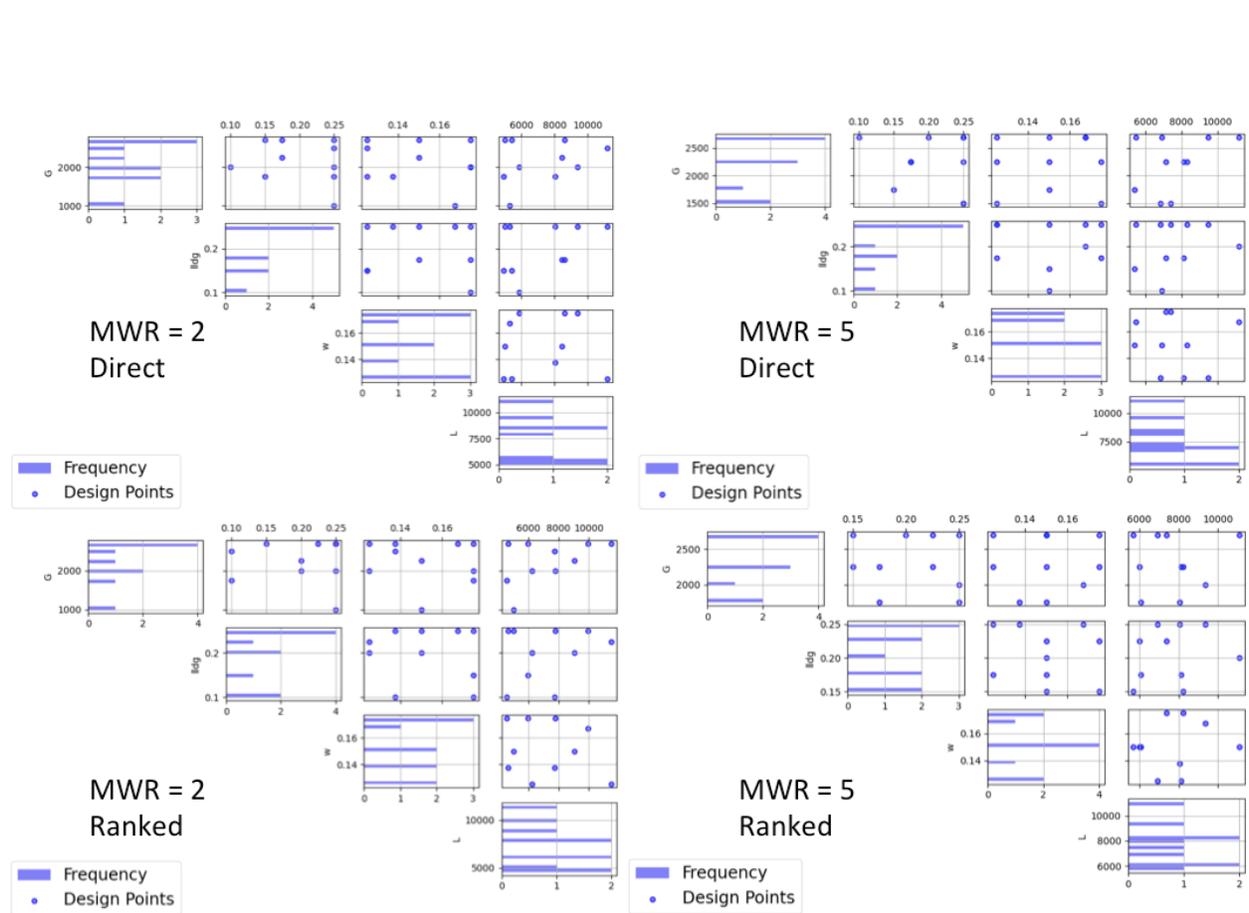


Fig. 73: Ex NUSF2 Comparison of the pairwise scatterplots for designs with MWR values of 2 and 5 with both Direct and Ranked scaling

Example IRSF-1: Constructing Input-Response Space Filling maximin designs for a 2-D input space with 1-D response

For this first Input-Response Space Filling design example, the goal is to construct a Pareto front of input-response space-filling designs, all with 20 runs in a 2-dimensional input space based on a regular unconstrained square region populated with a grid of candidate points, along with a single response variable. All designs on the Pareto front have a unique balance of space filling in the input space and response space, giving the experimenter latitude to choose which design is best for a particular situation.

The choice of how to construct the candidate set should be based on: (a) what is the precision with which each of the inputs can be set in the experiment (the candidate set should not contain increments finer than what can be set in the experiment), and (b) timing for generating the designs. The finer the grid that is provided in the candidate set, the longer the search algorithm will take to run for a given number of random starts. In general, a finer grid will give better options for the best design, but with diminishing returns after a large number of candidates have already been provided.

As noted previously in the Basics section, in addition to specifying the candidate point input combinations, it is also required to supply an additional column of the likely response variable values (or multiple columns for multiple responses of interest). This column will be used to make sure that space filling is being accomplished in the response space as well as the input space.

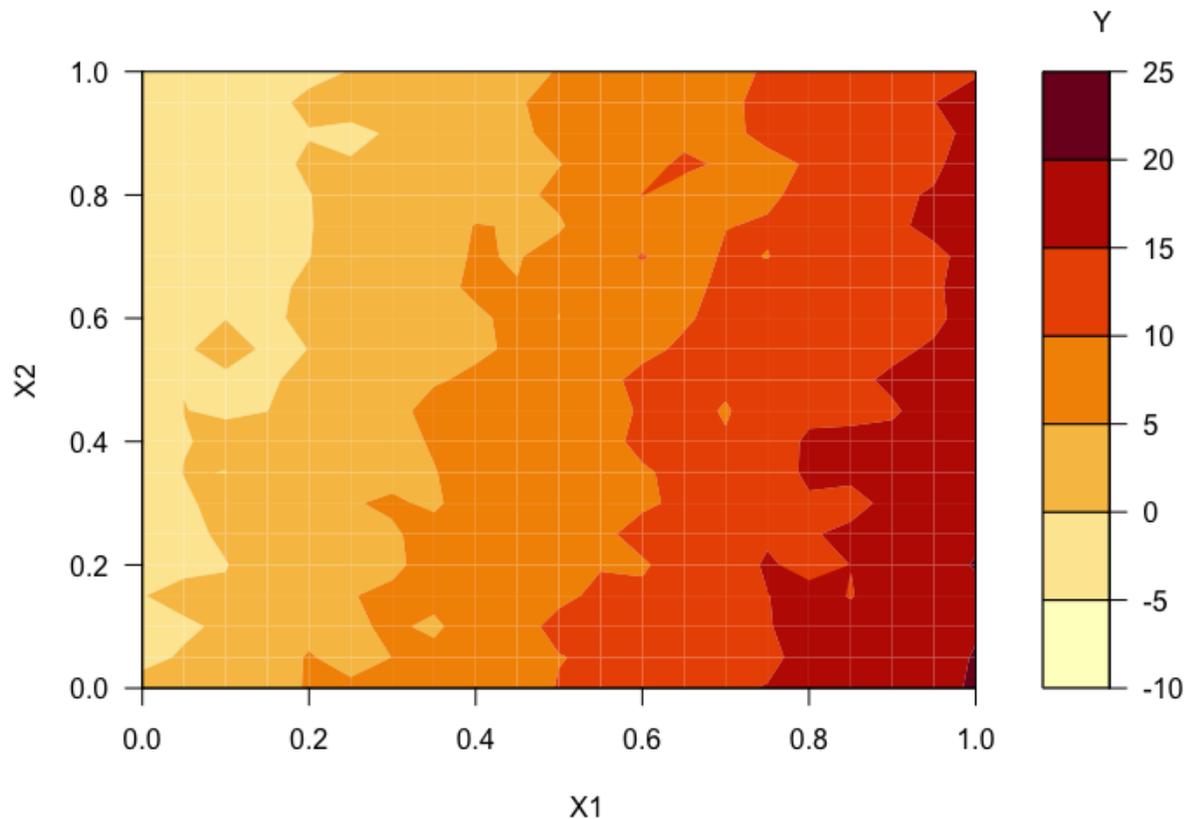


Fig. 74: Figure 1: Contour plot of the candidate set, with the two inputs X1 and X2 on the axes and the response Y in color

The candidates are laid out in a regular grid with equal spacing between levels of each of X1 and X2. There is a

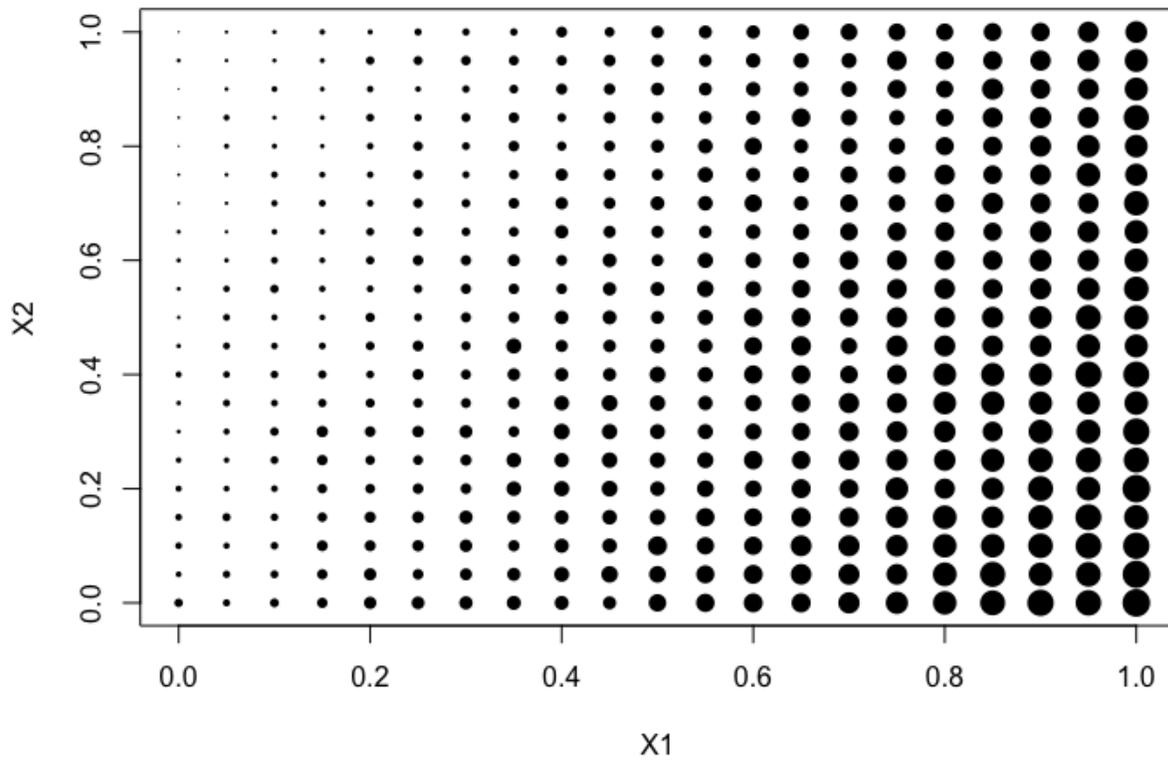


Fig. 75: Figure 2: Plot of the candidate set, here with Y represented in the point size

single response variable of interest, Y . The relationship between the inputs X_1 and X_2 and the response of interest Y can be characterized by a linear model.

A contour plot of the function that was used to generate the response values is shown in Figure 1 above. In Figure 2, we can see a plot where the relative size of the points is proportional to the size of the response associated with that candidate point. This second representation is helpful when the candidate points do not fall on a regular grid, or if the relationship of the response and the inputs is not smooth.

Here is the process for generating IRSF designs for this problem:

1. From the FOQUS main screen, click the **SDOE** button. On the top left side, select **Load Existing Set**, and select the “irsf-example1-candset.csv” file from the examples folder.

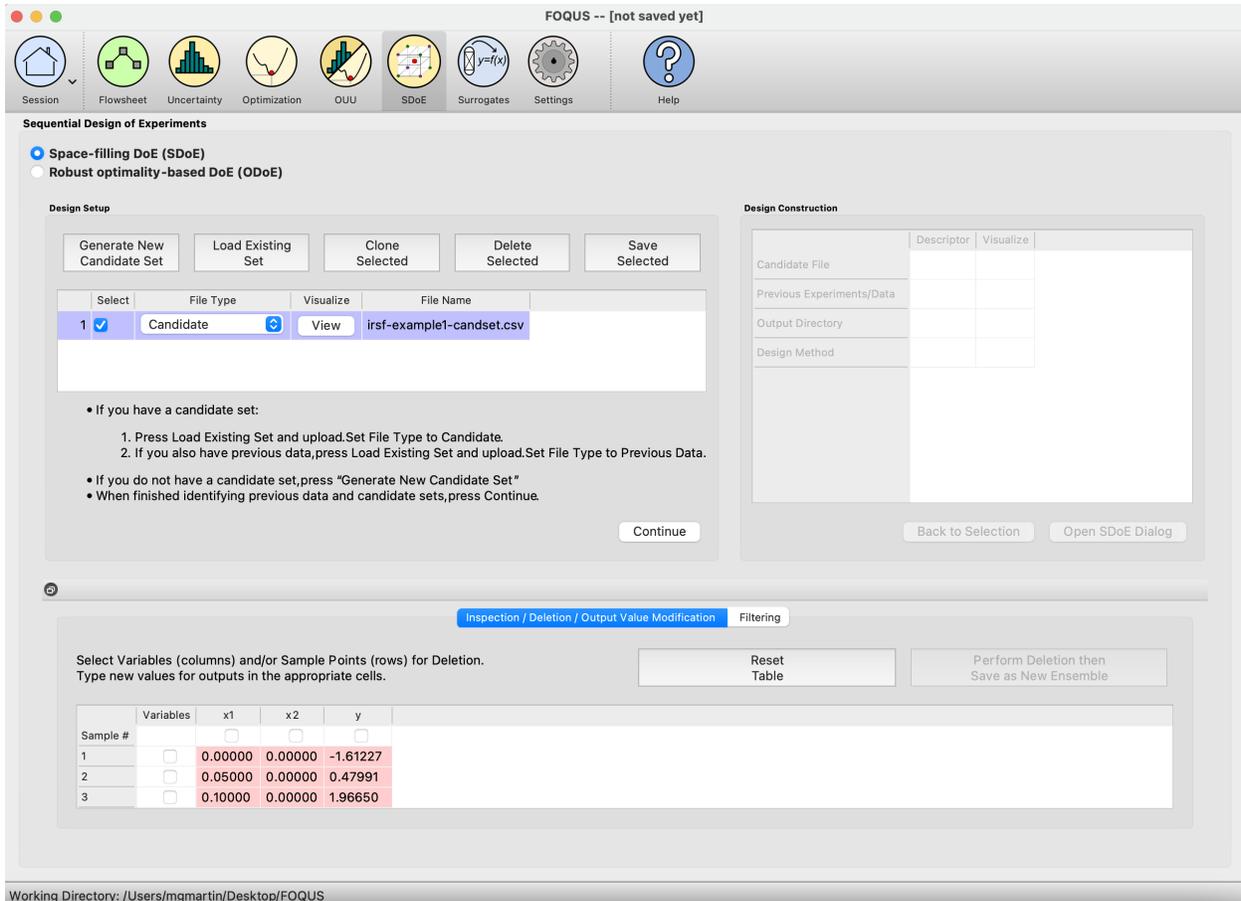


Fig. 76: Figure 3: SDOE Module First Page

2. Next, by selecting **View** and then **Plot**, it is possible to see the grid of points that will be used as the candidate points. In this case, the range for each of the inputs, X_1 and X_2 , has been chosen to be between 0 and 1. The range of the response, Y , is from -5 to 21.

3. Next, click on **Continue** to advance to the **Design Construction** Window, and then click on **Input-Response Space Filling (IRSF)** to advance to the second SDOE screen, where particular choices about the design can be made. On the second screen, the first choice for **Optimality Method Selection** is automatic, since the input-response space filling designs only use the **Maximin** criterion. We next select the **Design Size**, where here we have decided to construct a design with 10 runs. The choice of design size is typically based on the budget of the experiment.

There are also choices for which columns to include in the design construction. Here we use all 3 columns for creating the design, so the three **Include?** boxes for our X_1 , X_2 , and Y columns remain checked. In addition, it is

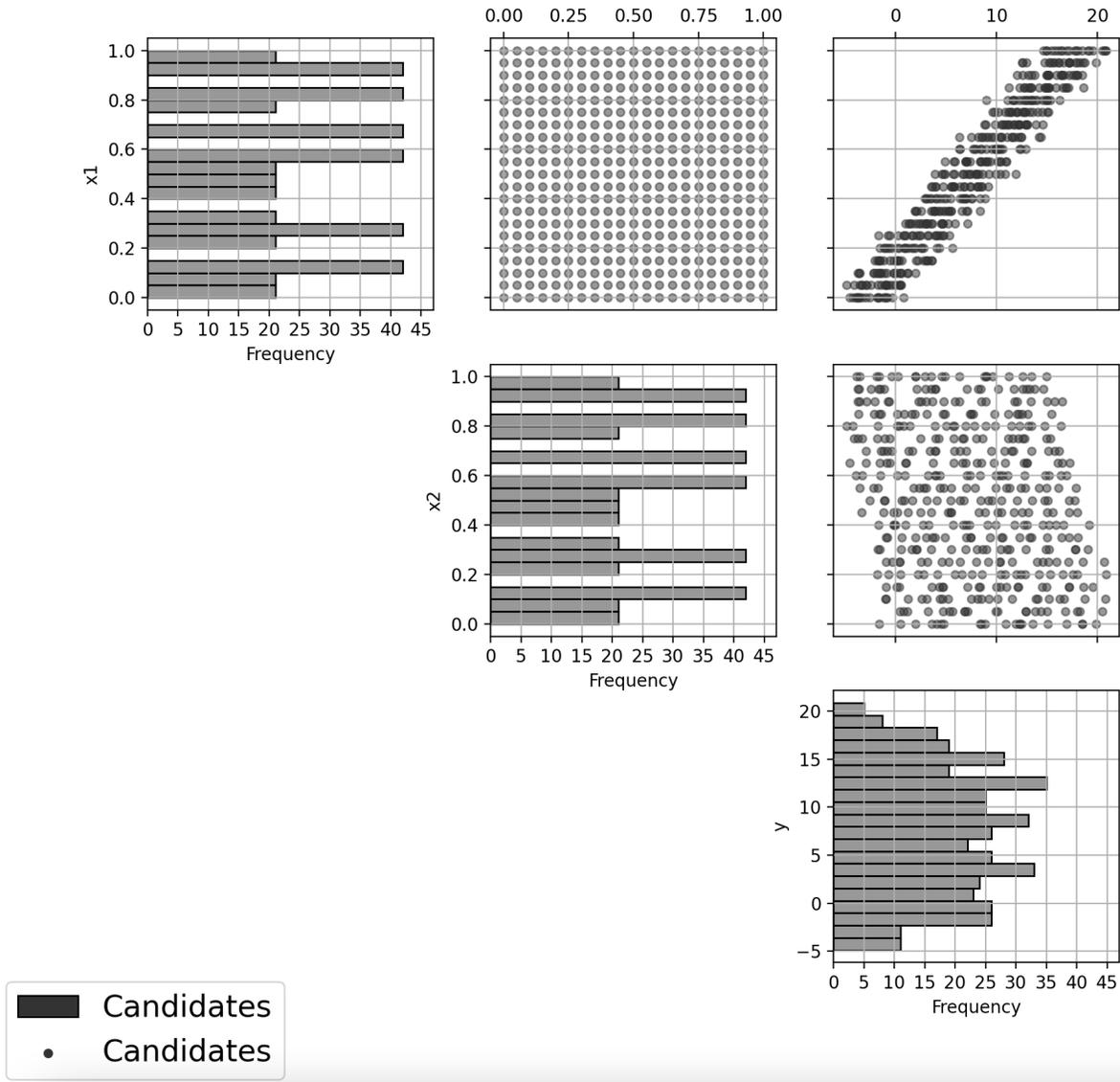


Fig. 77: Figure 4: Pairwise Scatterplots of the Candidate Set

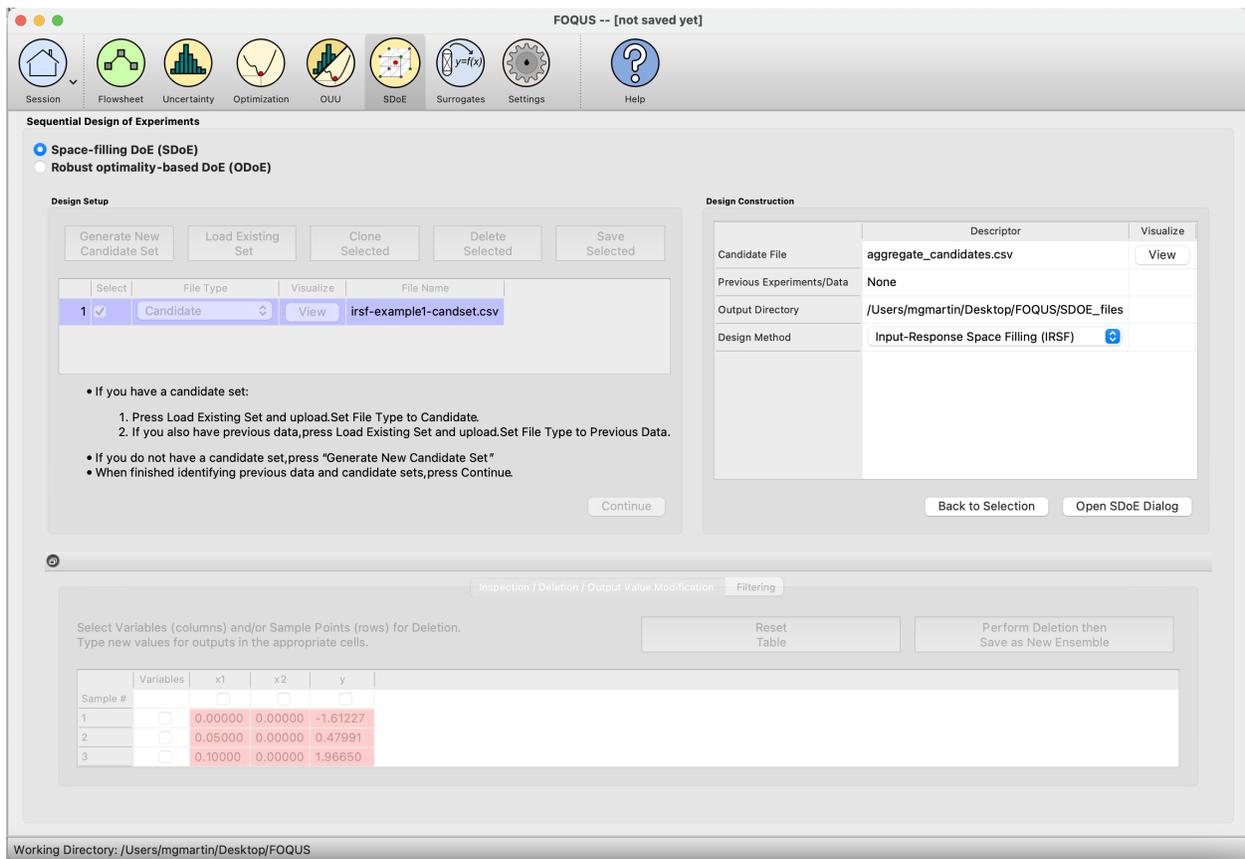


Fig. 78: Figure 5: Choose Design Method on the Right-Hand Side

Sequential Design of Experiments

Generate Design

Optimality Method Selection

Minimax
 Maximin

Desired Design Size

Design Size

	Include?	Name	Type	Min	Max
1	<input checked="" type="checkbox"/>	__id	Index	0.0	440.0
2	<input checked="" type="checkbox"/>	x1	Input	0.0	1.0
3	<input checked="" type="checkbox"/>	x2	Input	0.0	1.0
4	<input checked="" type="checkbox"/>	y	Response	-4.88	20.86

This performs a small number of iterations of the search algorithm to estimate timing for constructing the designs.

Fig. 79: Figure 6: Generate Design Box for Making Design Selections

possible to see the range of values for each of the columns in the spreadsheet. Here the two input columns range from 0 to 1, while the Y column ranges from -4.88 to 20.86. The user can change these values if they wish to rescale the ranges to widen or narrow them, but in general these values can be left as is. There is an automatic index column called “_id” in the first row of the design generation box. The **Type** is preselected as **Index**, though if not needed, the **Include?** could be unchecked to exclude this column from the candidate set. We will keep it as we have no other index column in the candidate set.

Next, we must confirm that each row has the correct **Type** indicated. The index column has the type **Index**. The two input columns **X1** and **X2** have type **Input**. The response variable **Y** currently has the type **Input** which must be changed to **Response** before moving forward with creating an input-response space filling design.

4. Once the choices for the design have been specified, click on the **Estimate Runtime** button to estimate the time required for creating the designs. For the computer on which this example was developed, if we ran 10 random starts, it is estimated that the algorithm would take 2 minutes and 50 seconds to generate the designs and identify those on the Pareto front. Note that the timing changes linearly, so using 20 random starts would take twice as long as using 10 random starts. Recall that the choice of the number of random starts involves a trade-off between getting the designs created quickly and the quality of the designs. For many applications, we would expect that using at least 20 random starts would produce designs that are of good quality. For this example we select to run 50 random starts, which is projected to take 13 minutes and 10 seconds.

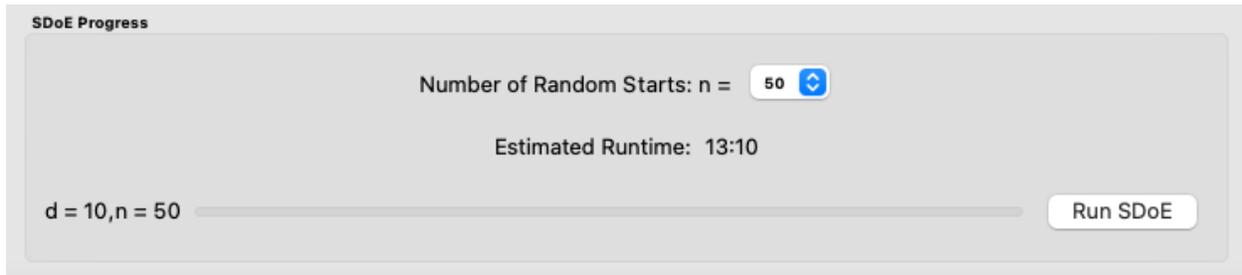


Fig. 80: Figure 7: Number of Random Starts

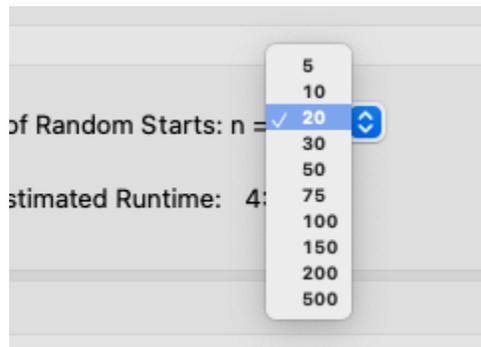


Fig. 81: Figure 8: Choices for Number of Random Starts

5. Once the algorithm has generated the designs, the left box called **Created Designs** populates with the Pareto front of designs that we have created. The Pareto front will populate a single row of the Created Designs box, and will display some useful information such as the **Number of Designs** found on the Pareto front, **Number of Random Starts** used, and the **Runtime**. If another design search is run afterwards, that Pareto front will populate the next row, and so on.
6. To examine the Pareto front and each of the designs on the Pareto front, select **View**. A plot of the Pareto front will appear, with color-coded points representing the different designs on the Pareto front. Larger values on the x- and y-axis indicate designs that have better space-filling properties in the input and response space, respectively. An ideal design would be near the top-right corner of the plot, with very large values on both the x- and y-axis, however this is



	Design Size, d	# of Random Starts, n	Runtime (in sec)	# of Designs	Plot SDOE
1	10	50	758.48	15	View

Fig. 82: Figure 9: Created Designs

rarely seen in practice. In reality, gains in space-filling in one space often come at the cost of space-filling in the other space. So, the Pareto front gives a spectrum of designs for which each is the best design for its given weighting of input and response space-filling. Along the ends of the Pareto front, one of the spaces is weighted much more heavily than the other. Closer to the center of the Pareto front, the two spaces are weighted more equally. Experimenters will need to examine many designs, with different levels of input and response space filling properties, to find the right balance for their individual needs.

As we explained in the Basics section, a Pareto front is made up of a collection of objectively best designs for different weightings of space-filling in the response and space-filling in the input spaces. A design that is on the Pareto front cannot be improved along one criterion of interest (space-filling in the response or space-filling in the input space) without worsening along the other criterion; if a design is located on the Pareto front, there exists no other design that is the same or better in both dimensions. Thus, it may be confusing to some users that this Pareto front below shows some pairs of designs connected by a vertical line, indicating one should outperform the other in the vertical dimension (space-filling in the response). However, this is simply a result of rounding in the horizontal dimension. The true values are in fact different by a small amount in space-filling in the input space.

Once the Pareto front has been examined, experimenters should further explore by **clicking on one of the color-coded design points within the plot to view that design**. Once a point is selected, a pairwise scatterplot of the chosen design will open, with the scatterplots and histograms being of the same color as the design point on the Pareto front for ease of comparison between designs. Multiple designs can be open simultaneously.

7. To get a better understanding of the different designs located on the Pareto front, we will examine three: one from the left end, one from the right end, and one from the middle. The three designs we will choose are Design 1 (purple), Design 15 (red), and Design 10 (green), as shown on the Pareto front plot above.

From the values on the axes in Figure 10 for each of the three design points, we can determine, even before viewing the individual designs, several important facts. We know that Design 1 (purple) is the best design if we want the objectively-best space-filling in the response space, and don't mind poor space-filling in the input space. Similarly, we know that Design 15 (red) is the best design if we want the objectively-best space-filling in the input space, and don't mind poor space-filling in the response space. We also know Design 10 (green) will offer a compromise, with moderate space-filling in both spaces. Design 10, or another compromise design along the Pareto front, is a good choice if we hope to balance space-filling in the input and response spaces.

Note: The design with the best space-filling in the input space overall, here Design 15 (red), is the same as a regular uniform space-filling design.

To determine how well a design fills the response space, we will look at the histogram for the response, Y, in the bottom-right for each of the designs and see how evenly spread the values appear. If we had a two- or higher-dimensional response space, we would examine the scatterplot(s) for the response variables for even spacing. We can confirm that Design 1 (purple) does have the best space-filling in the response space of the three. The values

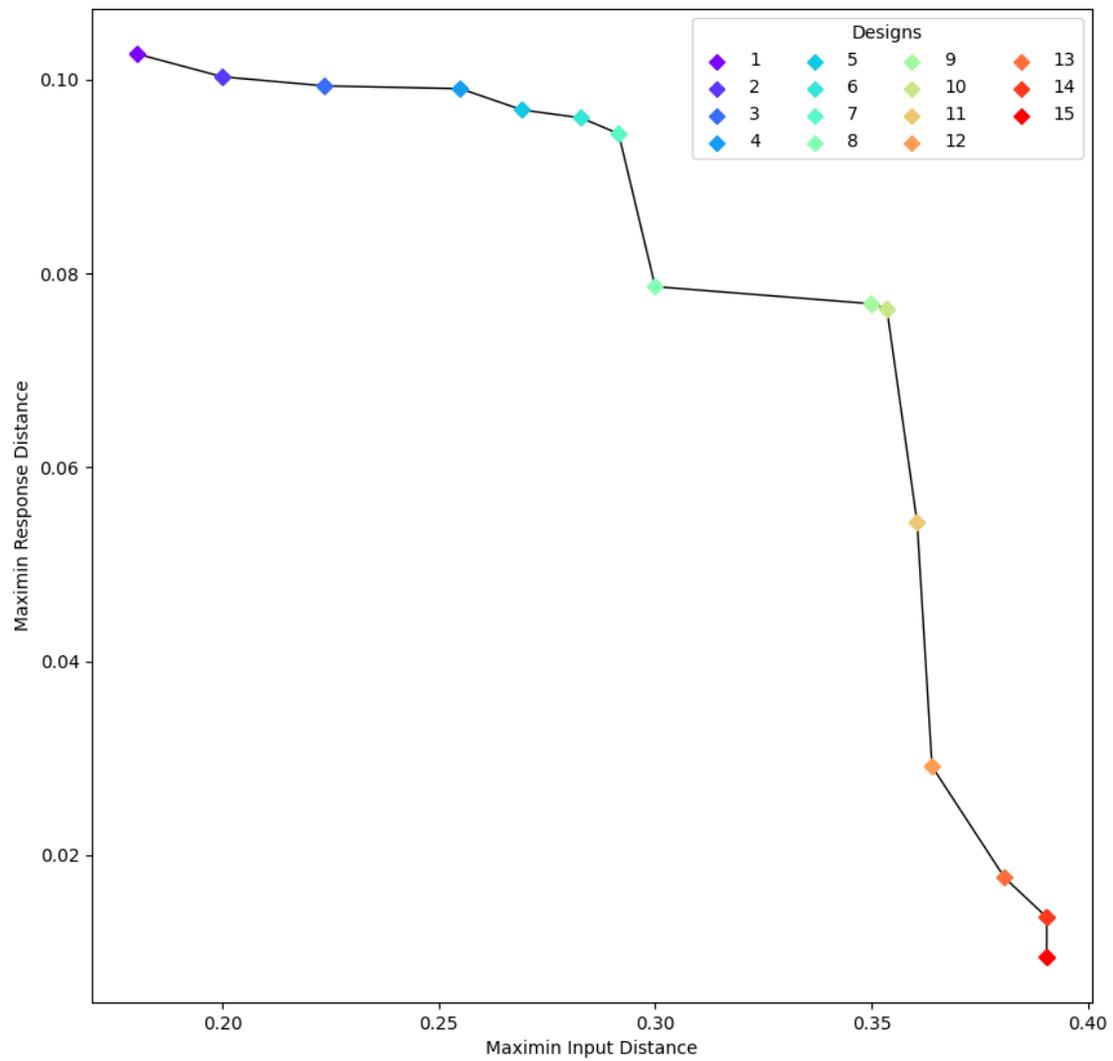


Fig. 83: Figure 10: Pareto Front of Created Designs

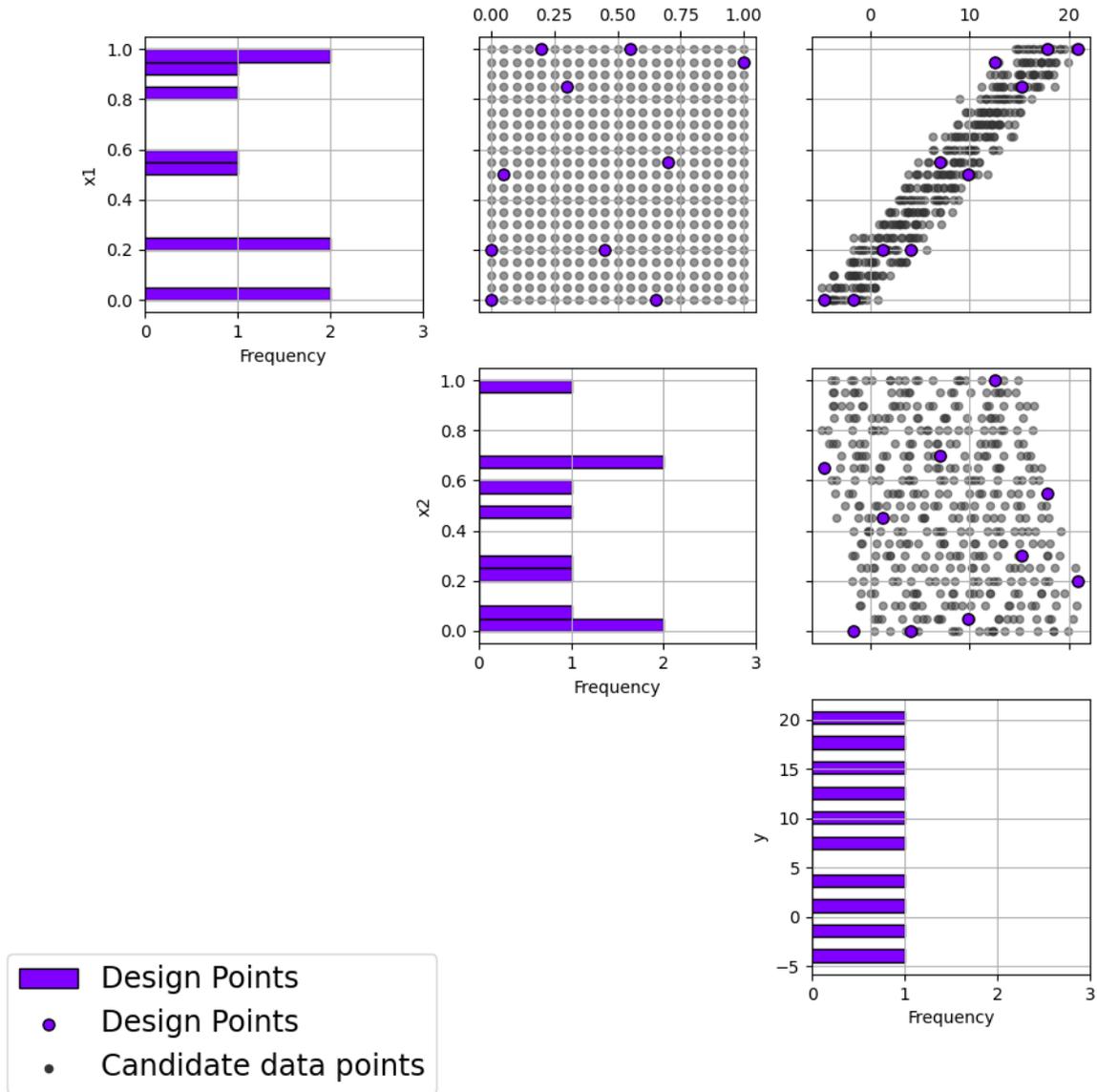


Fig. 84: Figure 11: Pairwise Scatterplot of Design 1

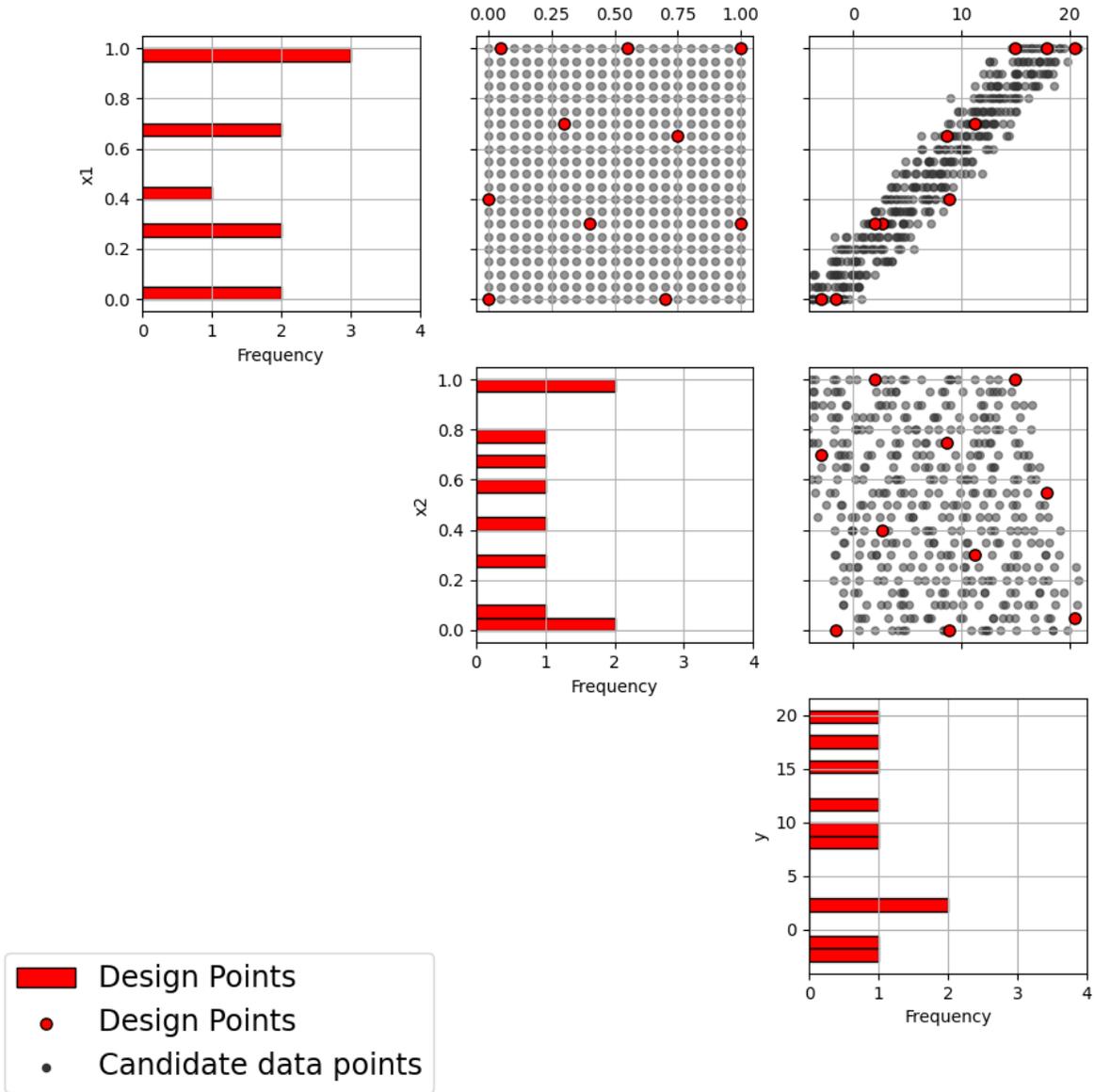


Fig. 85: Figure 12: Pairwise Scatterplot of Design 15

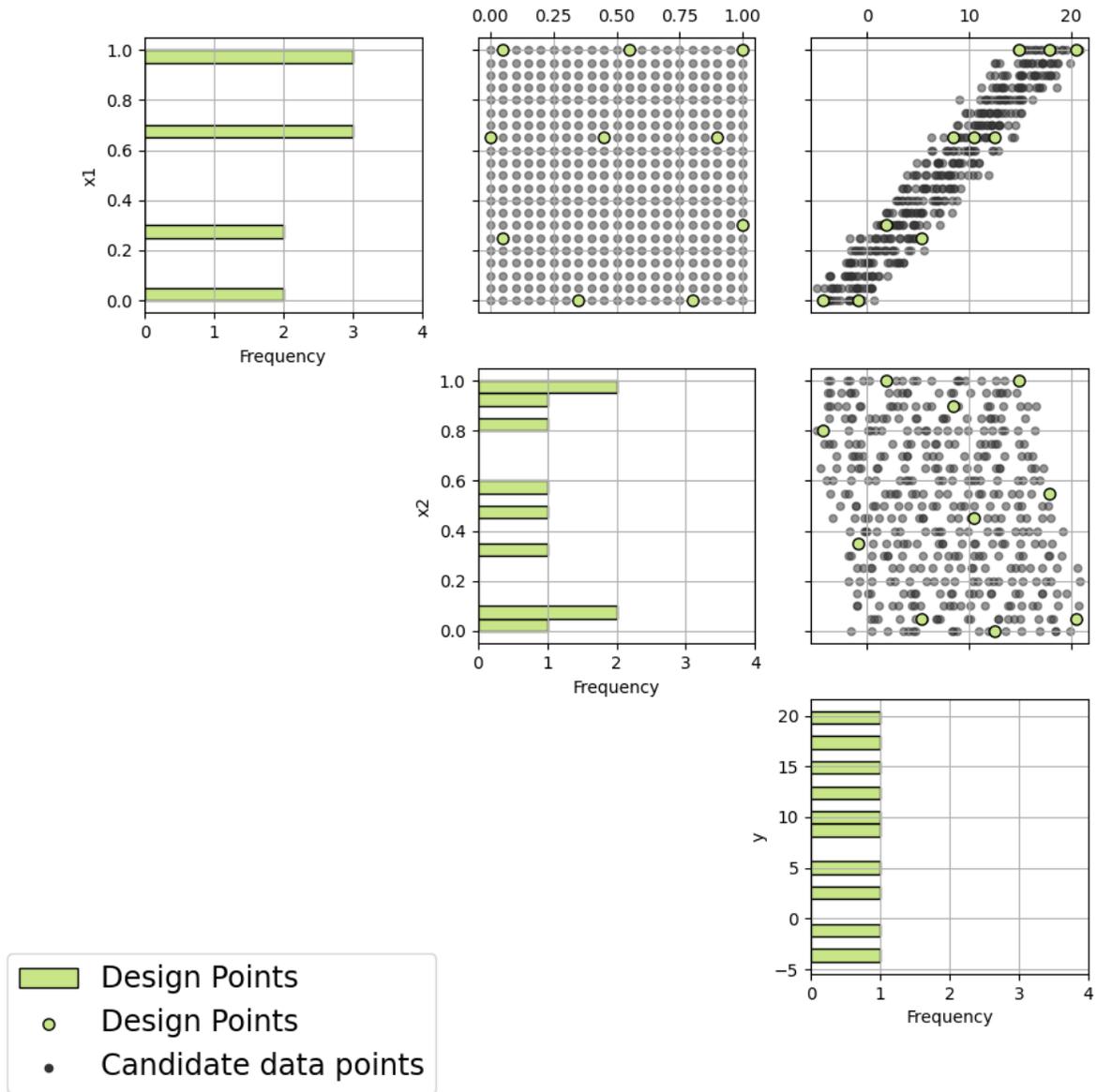


Fig. 86: Figure 13: Pairwise Scatterplot of Design 10

of the response are evenly spread throughout the space, with no large gaps. By contrast, Design 15 (red) has many holes and gaps in the response space.

Even though the criterion value for response space-filling in Design 10 (green) is less than Design 1 (purple), the response space-filling in Design 10 seems to fill the space fairly well. The differences in criterion values provide a useful summary of the trade-offs but it is important to also examine the scatterplots directly for a more intuitive illustration of what these trade-offs will look like in practice.

To examine input space-filling, we will now look at the scatterplot of the input variables, X1 and X2, located in the top-middle. If we had more than two input variables, we would look at a combination of pairwise scatterplots. It would be a bit harder to determine how well the space-filling of a given design appeared, so in that case, we rely more heavily on the position of the design on the Pareto front.

Here, Design 15 (red) definitely has the best input space-filling. The design points are spread apart with no large holes or gaps, covering the entire space well. The input space-filling in Design 1 (purple) has many large holes, and even that in Design 10 (green) has a hole or two.

With this variety of space-filling designs, plus the 12 more located on the Pareto front, it's easy to see there are many "best" designs for any given weighting of input and response space-filling. The Input-Response Space-Filling design tool gives the experimenter the flexibility to consider each design on the Pareto front to find the compromise between input and output space-filling to best fit the experimental objectives.

8. In the case of this example, we were hoping to find a design with good space-filling in both spaces. Design 10 (green) is an excellent candidate for this, though to be thorough we should also examine more designs along the Pareto front. In particular, Designs 4, 6, 7, and 11, and even 2, 3, and 13, should be explored to see how these other "best" designs balance space-filling uniquely in the two spaces.

ROBUST OPTIMALITY-BASED DESIGN OF EXPERIMENTS (ODOE)

9.1 Contents

9.1.1 ODoE

Overview

The FOQUS ODoE module supports several variants of optimal experimental design. This chapter presents an overview of these variants. Subsequently, details of the ODoE graphical user interface will be discussed in the *Tutorials* section.

Note: To run this version of ODoE, make sure you have the latest version of PSUADE installed (1.9.0).

ODoE

Variables

Suppose a simulation model is available for this study. Let this simulation model be represented by the following function:

$$Y = F(X, U)$$

which is characterized by two types of variables:

1. Design/Decision variables

- Notation: X with dimension n_x
- Definition: Design variables are continuous variables that are bounded in some specific ranges. A requirement is that the simulation output should be a smooth function of these design variables.

2. Continuous uncertain variables

- Notation: U with dimension n_u
- Definition: Continuous uncertain variables are associated with a joint probability distribution function from which a sample can be drawn to compute the basic statistics.

ODoE**Objective****Functions**

There are a few variants of the objective functions. They can be divided into two classes: those that optimize certain metrics associated with the posterior distributions of the uncertain variables (D-, A-, and E-optimality), and those that optimize certain metrics associated with the prediction uncertainties (G- and I-optimality).

1. D-optimality: D-optimal methods seek to find an optimal subset of points in the candidate set (provided by users) that minimizes the product of eigenvalues (or determinant) of the covariance matrix constructed from the posterior distributions of the uncertain variables.
2. A-optimality: A-optimal methods seek to find an optimal subset of points in the candidate set (provided by users) that minimizes the sum of eigenvalues of the covariance matrix constructed from the posterior distributions of the uncertain variables.
3. E-optimality: E-optimal methods seek to find an optimal subset of points in the candidate set (provided by users) that minimizes the maximum eigenvalue of the covariance matrix constructed from the posterior distributions of the uncertain variables.
4. G-optimality: G-optimal methods seek to find an optimal subset of points in the candidate set (provided by users) that minimizes the maximum prediction uncertainty (induced by the posterior distributions of the uncertain variables) among an evaluation set.
5. I-optimality: I-optimal methods seek to find an optimal subset of points in the candidate set (provided by users) that minimizes the mean prediction uncertainty (induced by the posterior distributions of the uncertain variables) among an evaluation set.

These methods can be computationally intensive when the candidate set is large and/or the sought-after optimal subset is large. FOQUS uses some global optimization algorithm in this context. Since there may exist many local minima, most of the time it may only be possible to find a sub-optimal subset given limited computational time (that is, exhaustive search may be computationally prohibitive).

9.1.2 Tutorials

This section walks through a couple of examples of running ODoE.

The files for these tutorials are located in: `examples/tutorial_files/ODOE`

The first example will use an existing candidate set and an existing evaluation set.

The second example will generate the candidate set and load an existing evaluation set.

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

Note: To run this version of ODoE, make sure you have the latest version of PSUADE installed (1.9.0).

Example 1: ODoE with Existing Candidate Set

In this example, the user will provide an existing candidate set.

1. Start FOQUS and click the 'SDoE' icon.

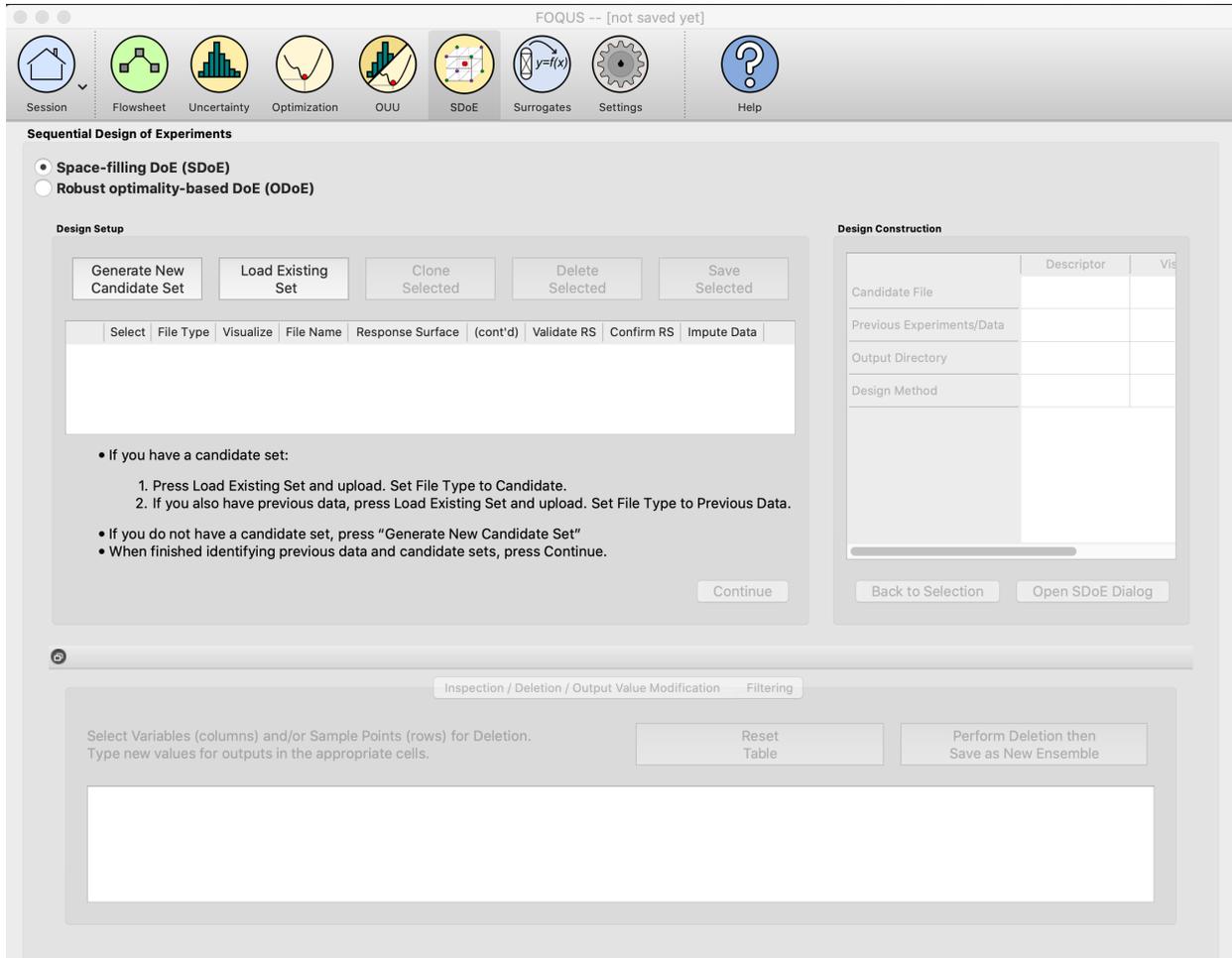


Fig. 1: SDoE Main Window

2. Select the radio button **Robust optimality-based DoE (ODoE)**
3. Click the **Browse** button to **Load RS Train Data**, browse and load the ODoE_example.csv from the examples folder.

First you will be prompted to specify the number of inputs in your training data.

After clicking **OK** the file loads and the screen will look like this:

4. Under **Input Setup** we need to select the type for each input. Leave **X1** and **X2** as **Variable** inputs and change **X3** and **X4** to be **Design** inputs.
5. Click **Confirm Inputs** and the **Simulation Ensemble Setup** window will pop up to generate our **Prior Sample**.
6. Switch to the **Sampling Scheme** tab and select **Monte Carlo** and leave the **# of Samples** at **1000**. Click **Generate Samples**. When the samples are generated, **Done!** will show up right next to the button. You can visualize the samples clicking on the **Preview Samples** button. Once the user is done, click the **Done** button so the generated samples get saved.

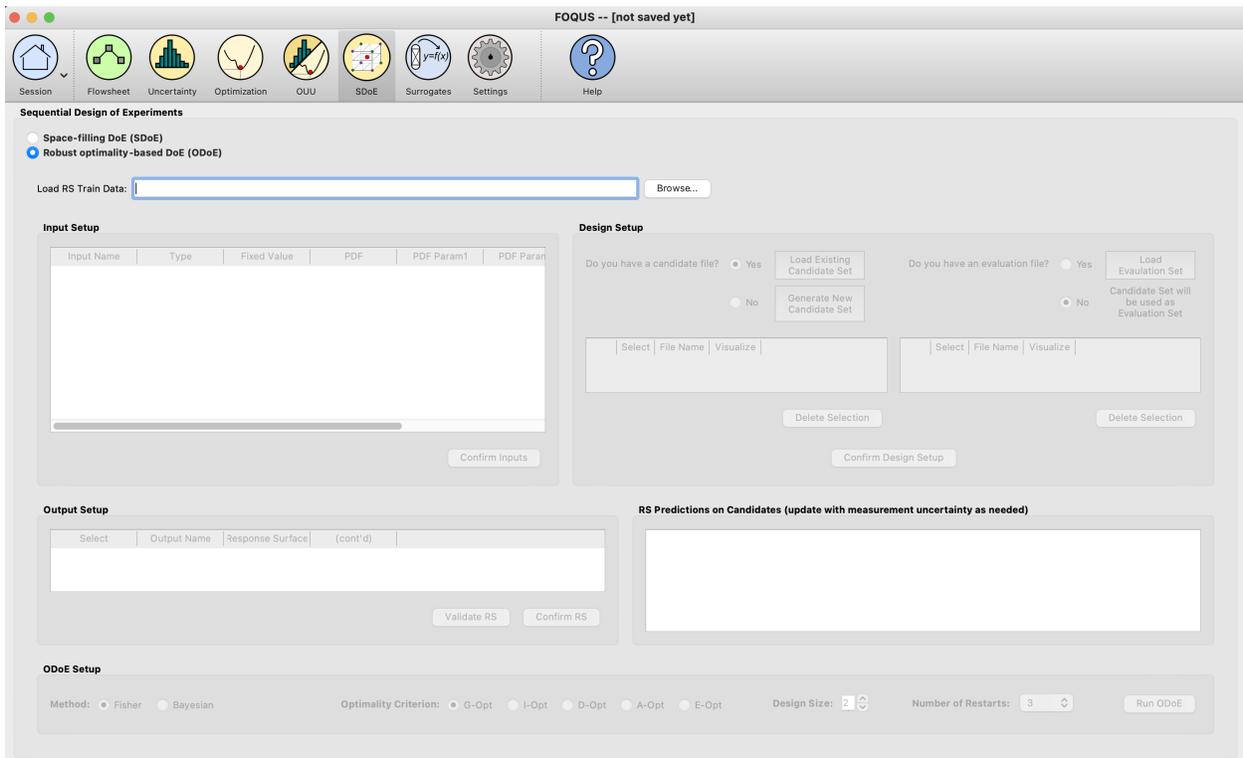


Fig. 2: ODoE Main Window

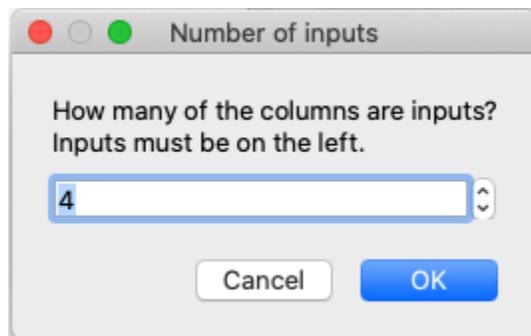


Fig. 3: ODoE Specify Number of Inputs

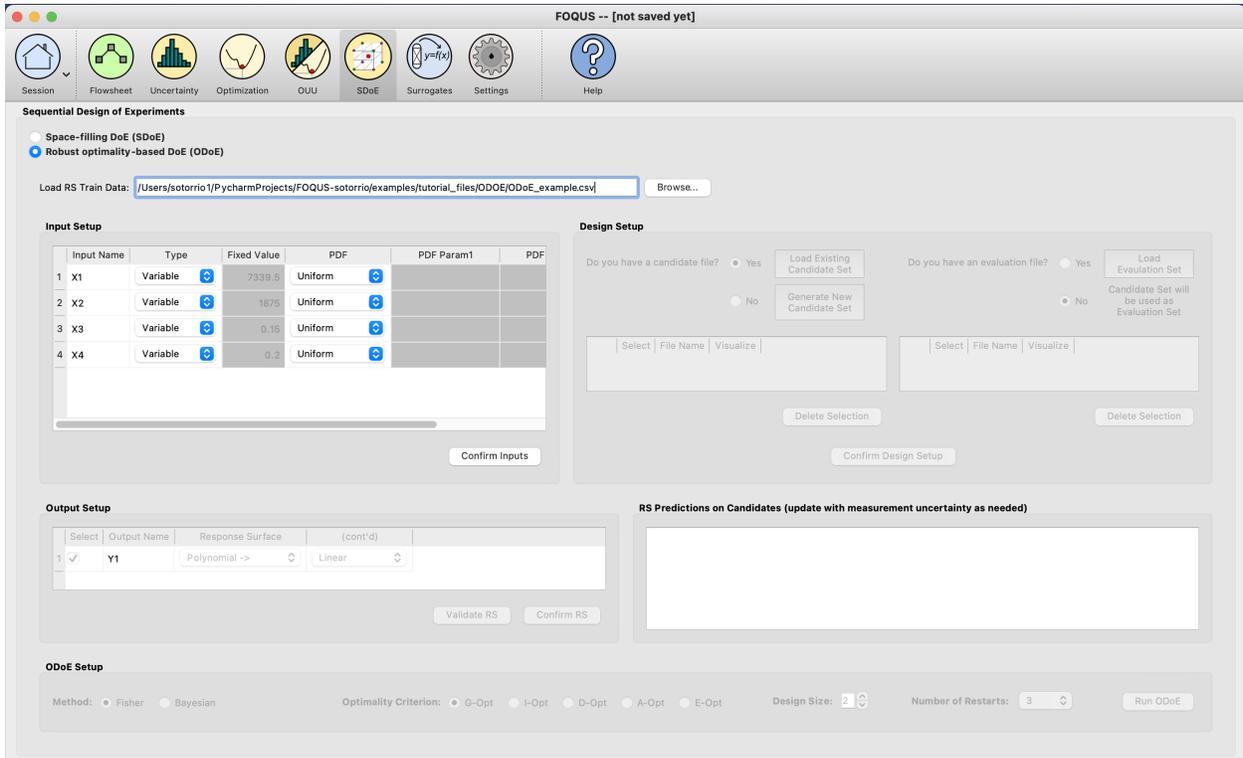


Fig. 4: ODoE Load RS Train Data

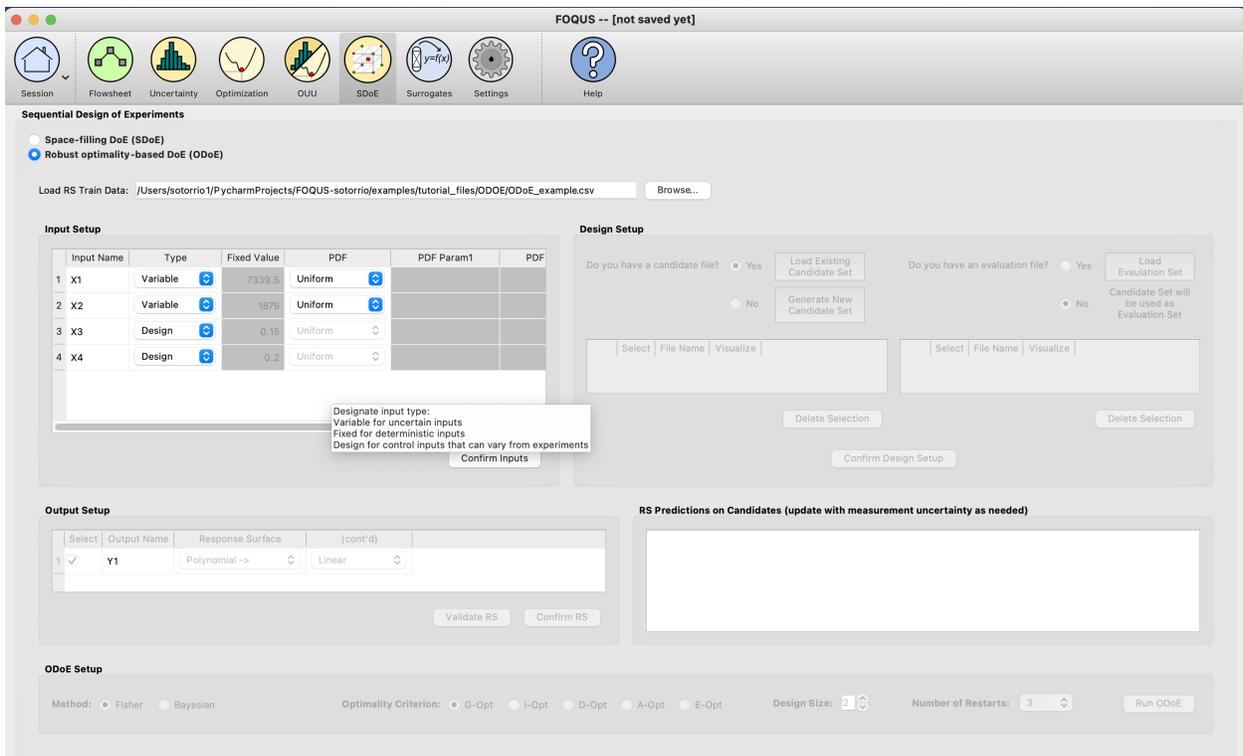


Fig. 5: ODoE Input Setup

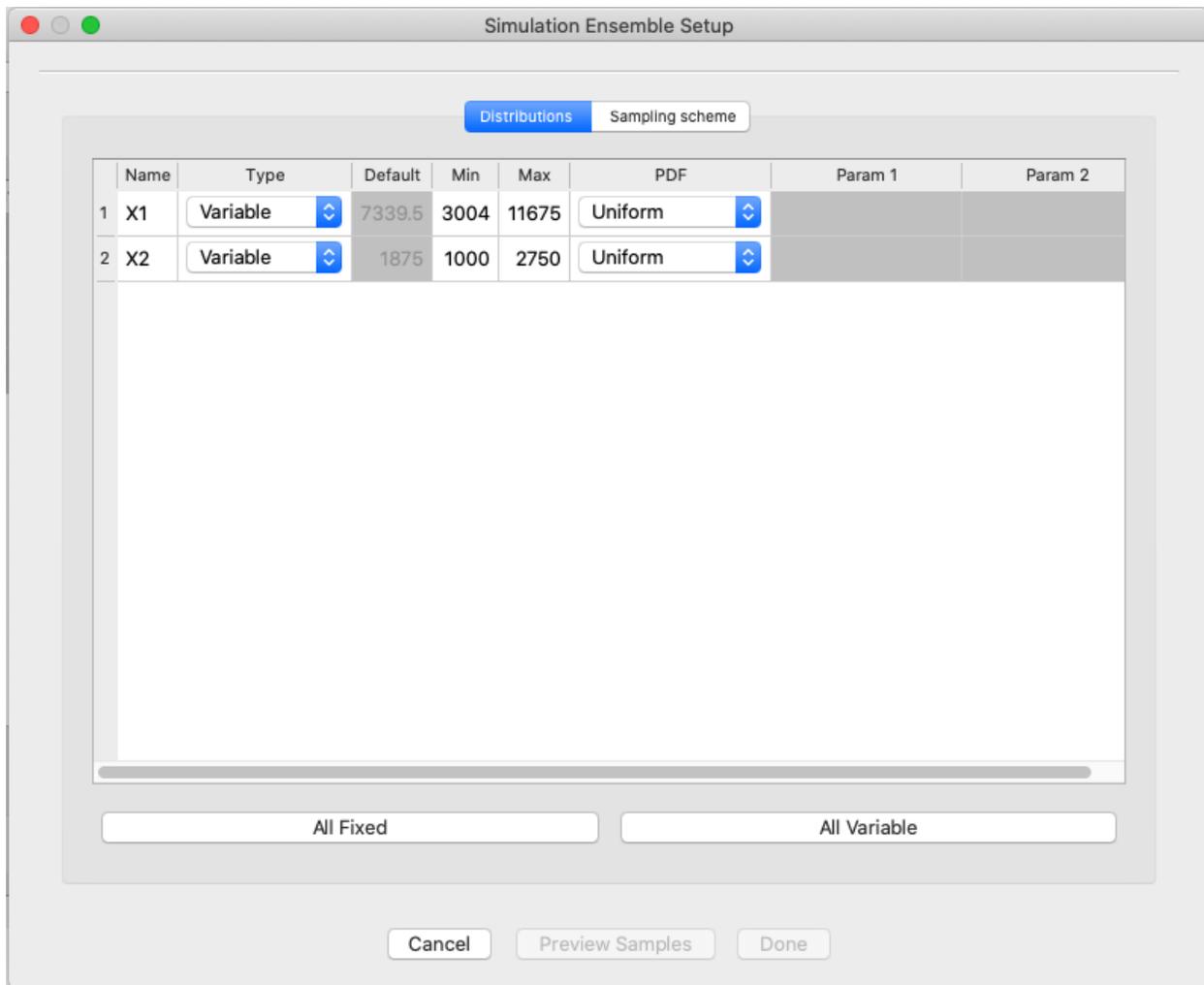


Fig. 6: ODoE Simulation Ensemble Setup for Prior Sample Generation

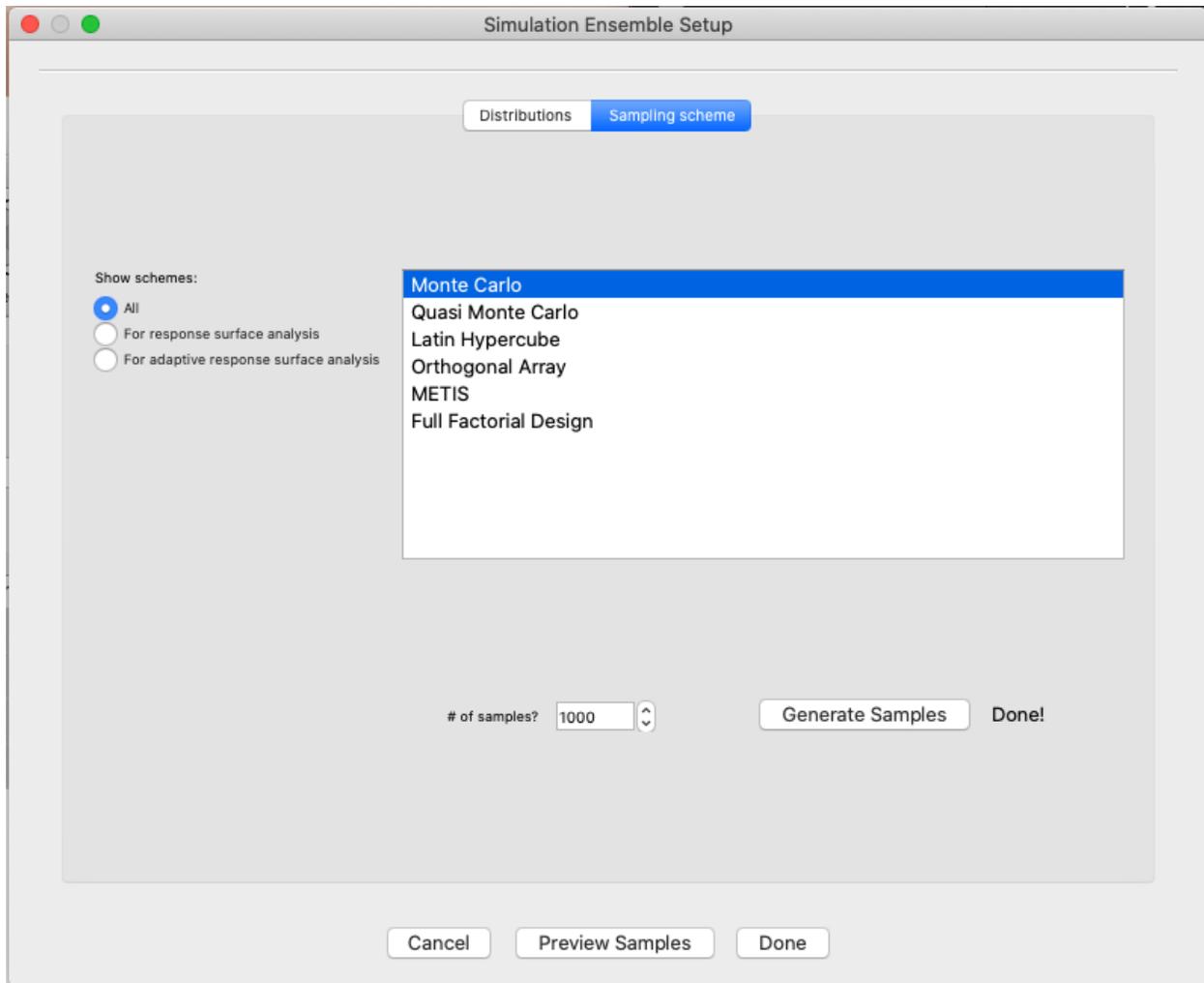


Fig. 7: ODoE Simulation Ensemble Setup for Prior Sample Generation - Sample Scheme

- Under **Design Setup** click on the **Load Existing Candidate Set** button.

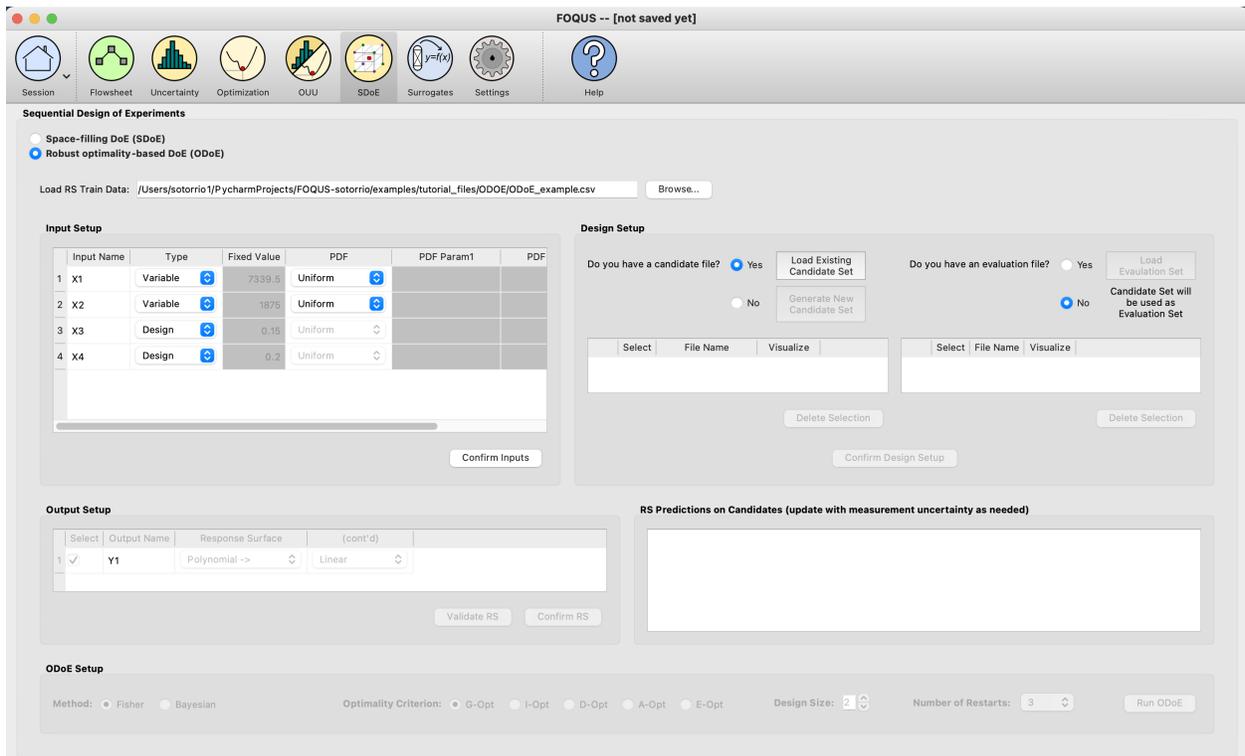


Fig. 8: ODoE Load Existing Candidate Set

Browse and load the CandidateSet.csv from the examples folder. The user can select the candidate and click on the **Delete Selection** button in case they want to delete the candidate set. To visualize the data, just click the **View** button under the **Visualize** column.

- On the right hand side of the **Design Setup** section, click on the **Load Evaluation Set** button.

Browse and load the EvaluationSet.csv file from the examples folder. Similar to the candidate set section, the user can select the evaluation set and click on the **Delete Selection** button in case they want to delete the evaluation set. To visualize the data, just click the **View** button under the **Visualize** column.

- Click on the **Confirm Design Setup** button and under the **Output Setup** section, select **MARS** in the **Response Surface** dropdown menu.
- Click **Validate RS** button and the user will get a informative message window and the response surface validation plot.

If the RS selected looks good, you can click the **Confirm RS** button. The response surface predictions on candidates will get populated in the table on the bottom right corner under **RS Predictions on Candidates**. The user can edit the **mean** and **standard deviation** columns in this table as needed.

- Under ODoE Setup select the **Method** (in this case Fisher), the **Optimality Criterion** (in this case G-Opt), **Design Size** (in this case 2) and **Number of Restarts** (in this case 3).

The choice of optimality criterion to use for design construction is driven by the objectives of the experimenter. If the primary focus of the experimenter is parameter estimation, then selecting the D- or A-optimality criterion is recommended. If the primary objective of the experimenter is precise prediction of the response of interest, then it is best to select the G- or I-optimality criterion. In this case, the experimenter was primarily interested in response prediction, so the G-optimal criterion was selected. Likewise, Design Size and Number of Restarts should be selected to best serve the needs of the experimental objectives. A larger design will allow more information to

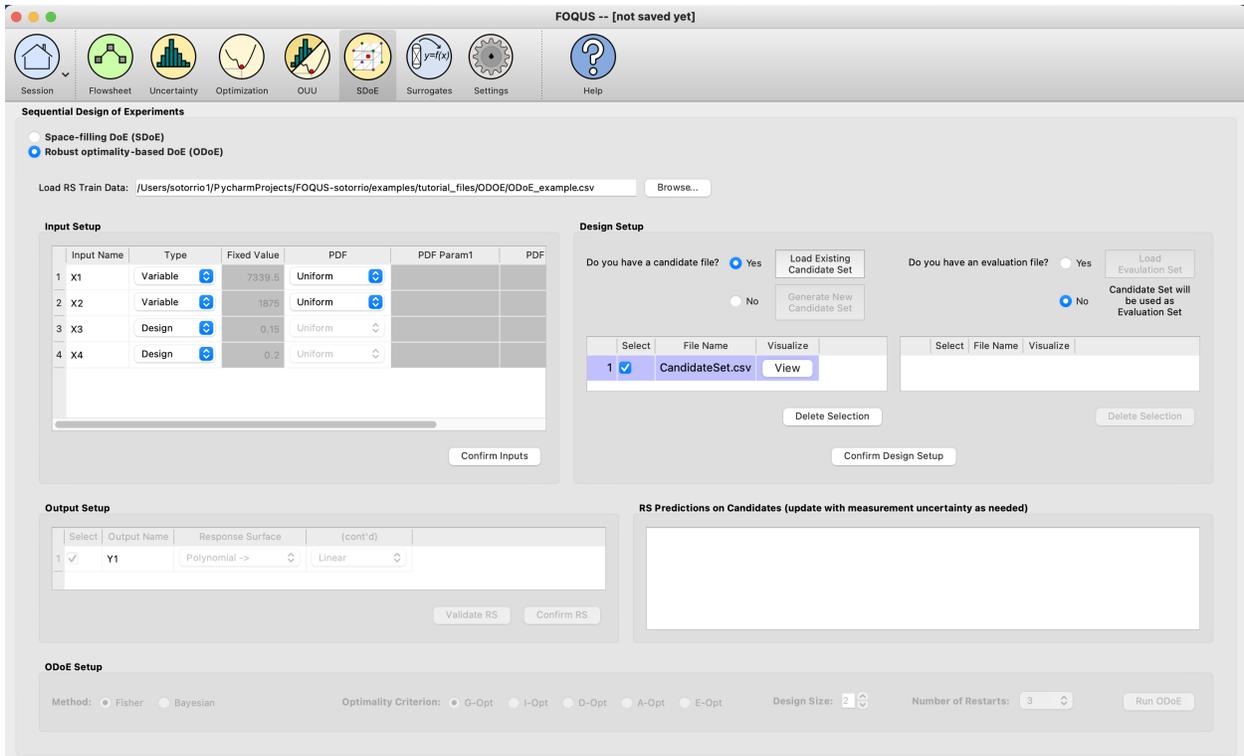


Fig. 9: ODOE Load Existing Candidate Set

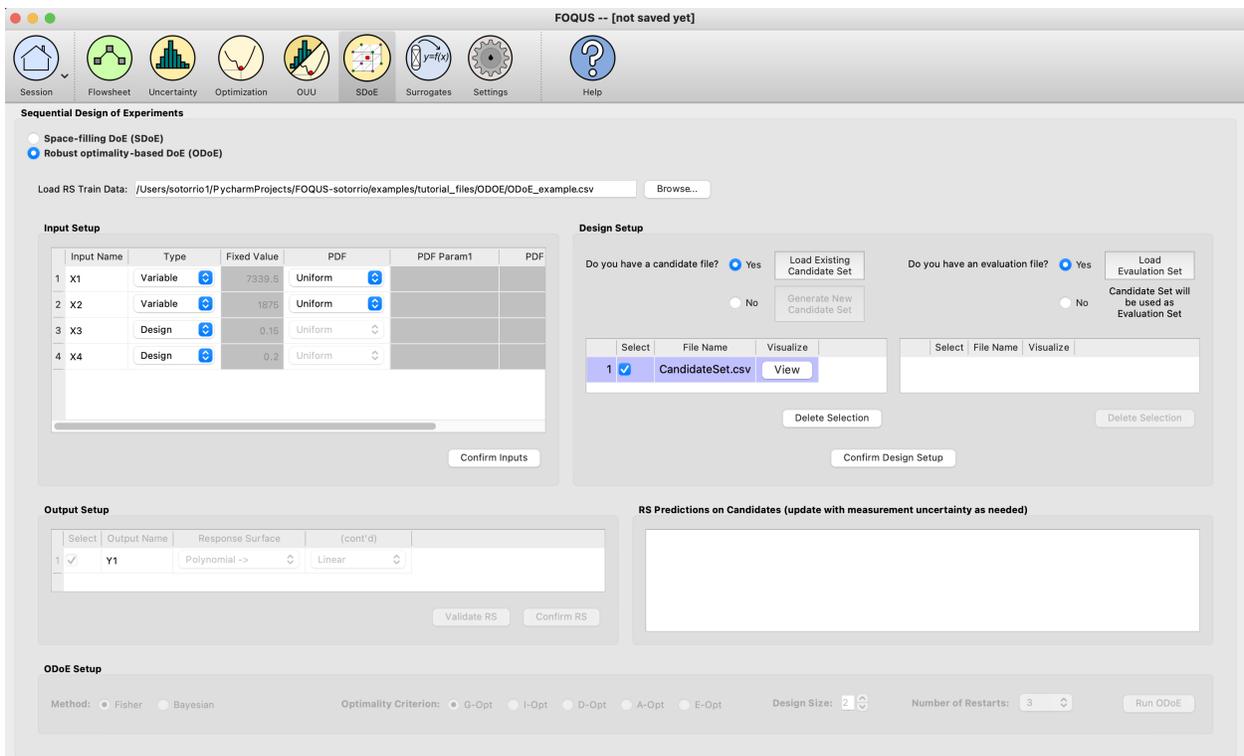


Fig. 10: ODOE Load Evaluation Set

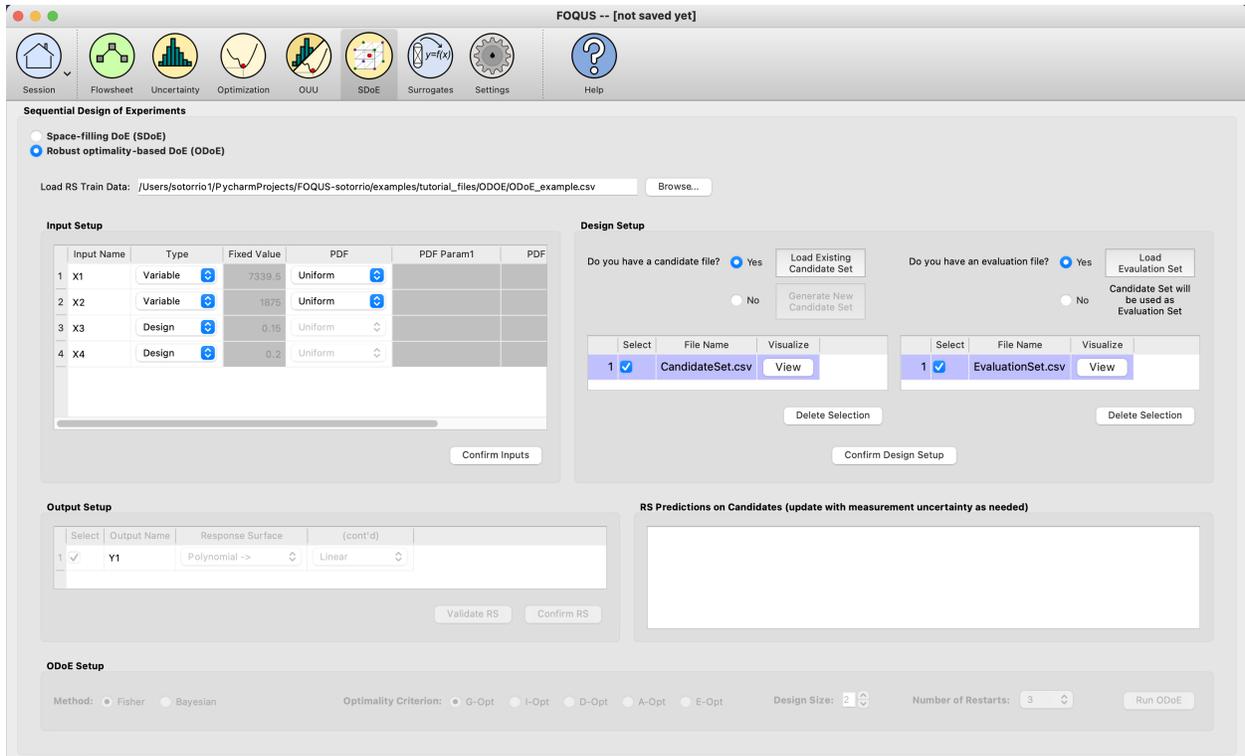


Fig. 11: ODoE Candidate and Evaluation Sets

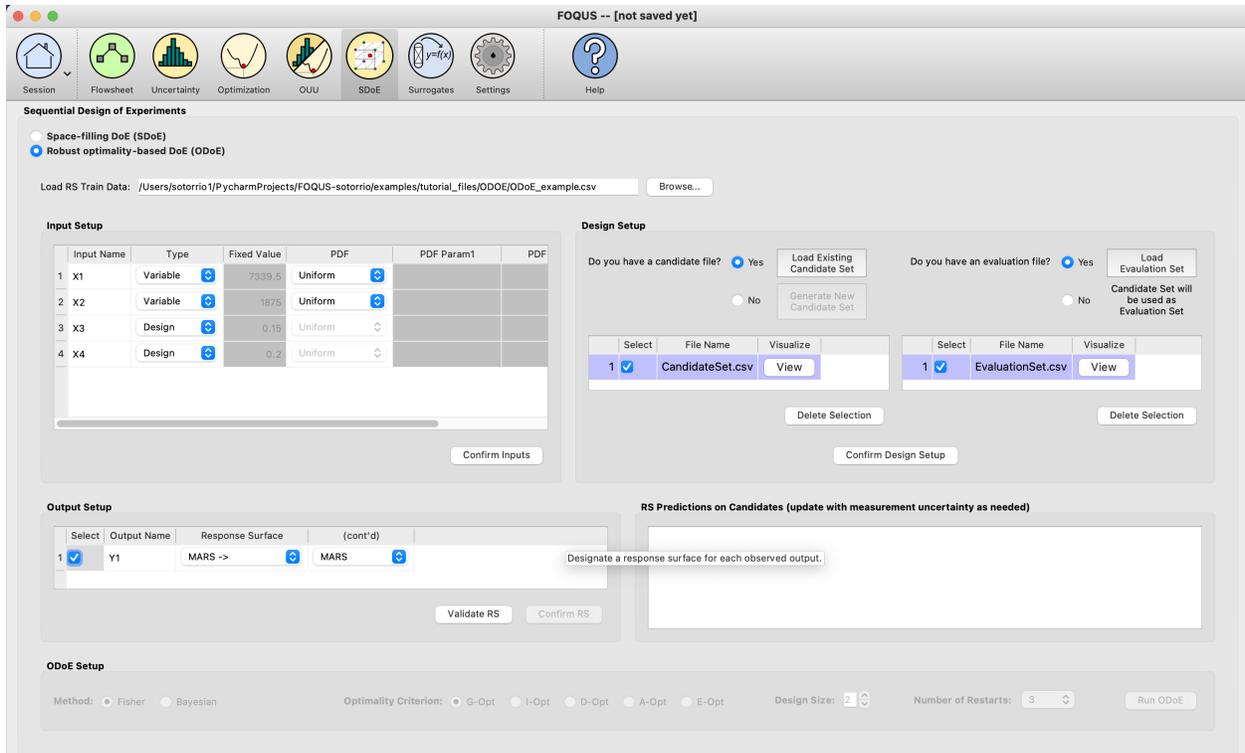


Fig. 12: ODoE Output Setup - MARS

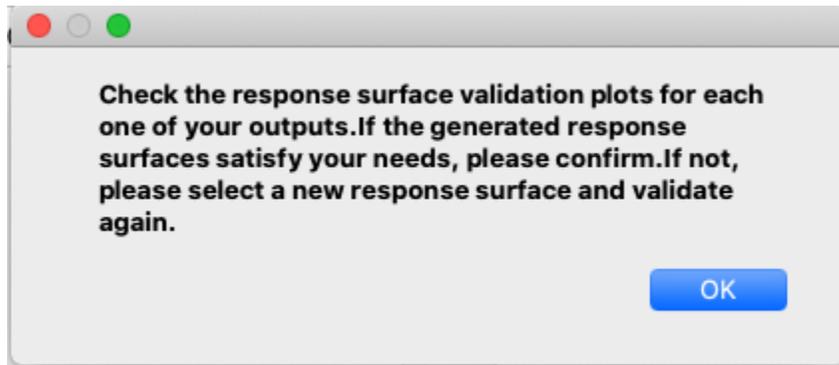


Fig. 13: ODoE Response Surface Validation Message

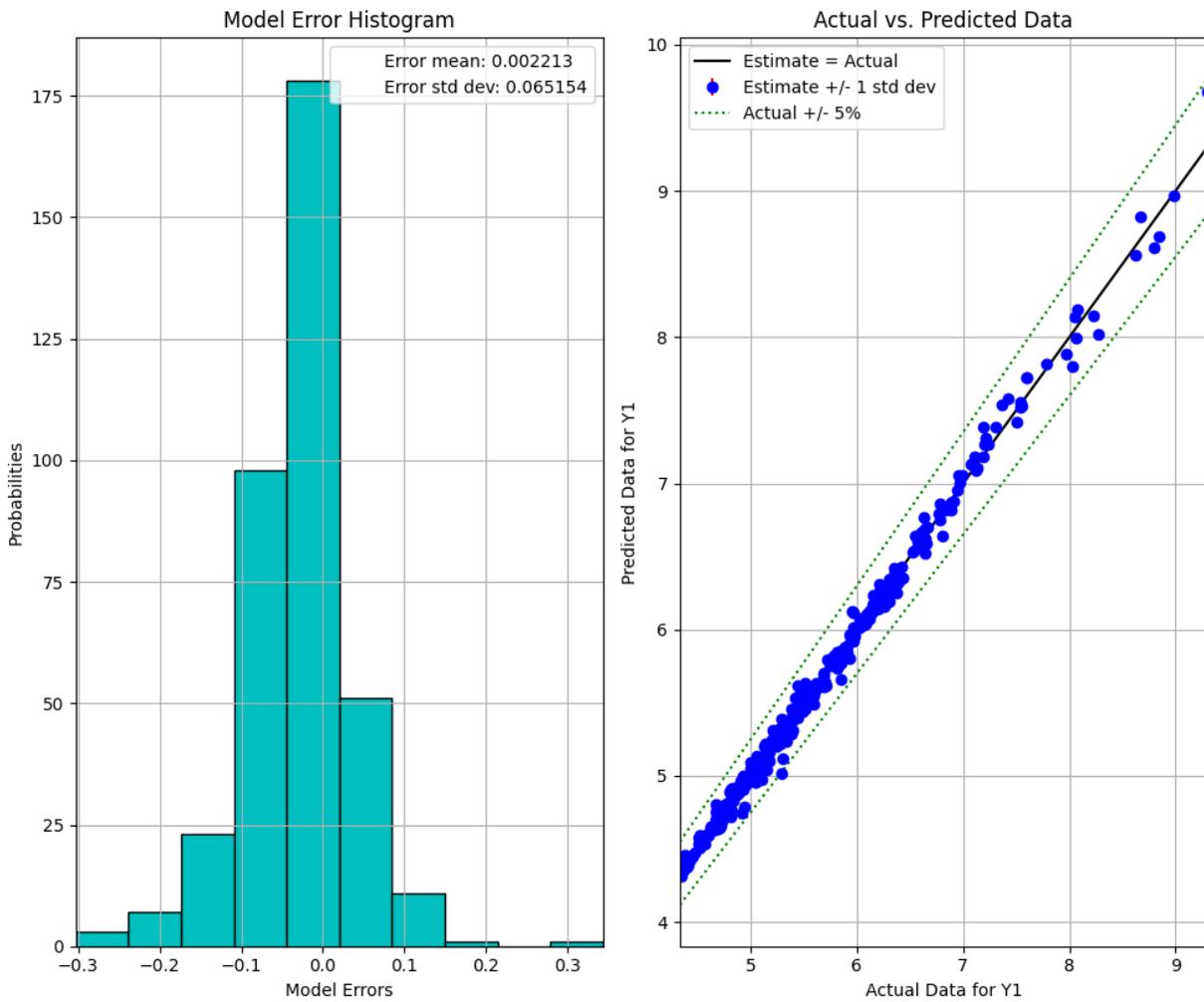


Fig. 14: ODoE Response Surface Validation Plot

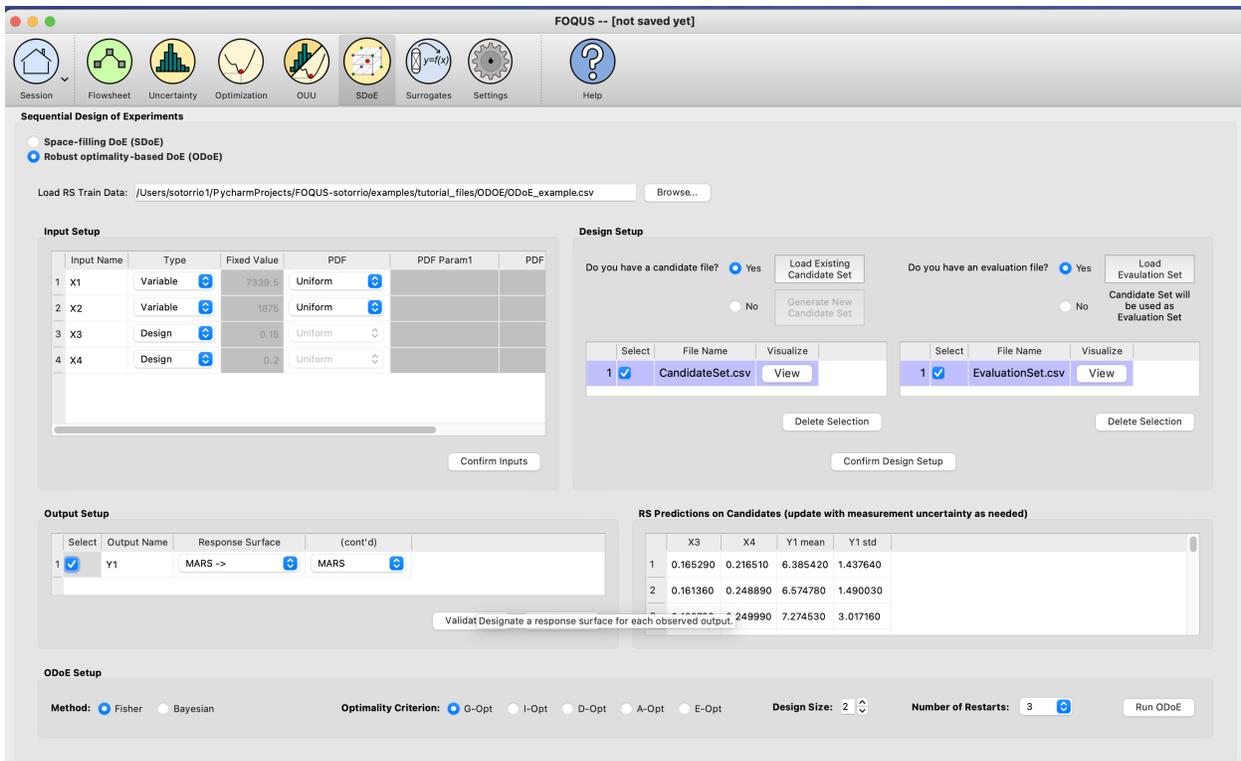


Fig. 15: ODoE Response Surface Confirmed and Predictions Generated

be collected than a smaller design, but will necessitate the use of more time and other experimental resources. The choice of design size is often dictated by the size of the experimental budget. Furthermore, the choice of Number of Restarts involves a trade-off between the quality of the design generated and the time to generate the design, with more restarts typically resulting in better designs. In this example, both design size and number of restarts were selected to fit within the given budgetary and time constraints of the experimenter.

Once those three parameters are decided, click the **Run ODoE** button. A window with PSUADE running will show up.

- Once PSUADE finishes generating the optimality-based design, another window will pop up with results information. A more thorough summary will also be saved in the **ODOE_files** directory as **odoe_results.txt**.

Example 2: ODoE Generating New Candidate Set

In this example, the user will generate a new candidate set.

- Start FOQUS and click the 'SDoE' icon.
- Select the radio button **Robust optimality-based DoE (ODOE)**
- Click the **Browse** button to **Load RS Train Data**, browse and load the **ODOE_example.csv** from the examples folder.

First you will be prompted to specify the number of inputs in your training data.

After clicking **OK** the file loads and the screen will look like this:

- Under **Input Setup** we need to select the type for each input. Leave **X1** and **X2** as **Variable** inputs and change **X3** and **X4** to be **Design** inputs.

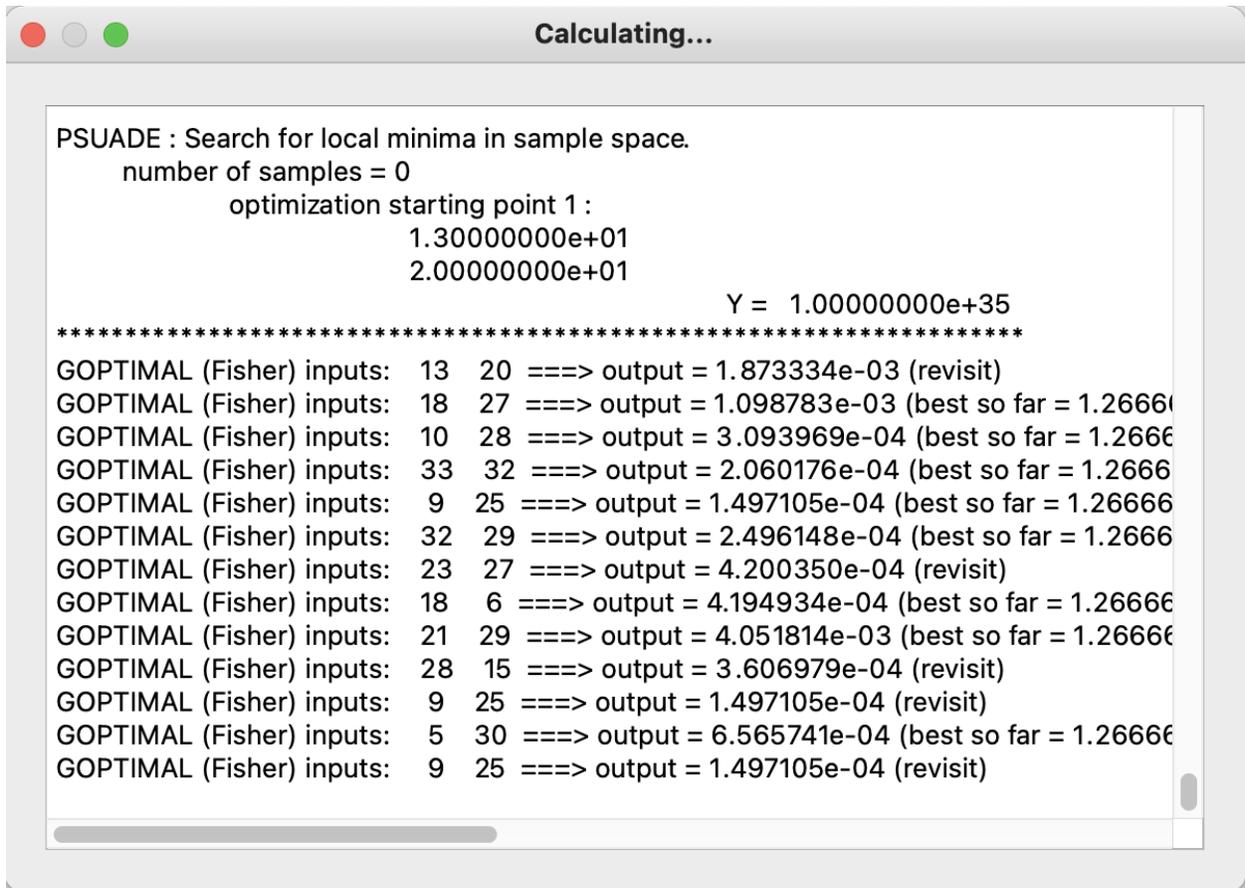


Fig. 16: ODoE PSUADE Running Window

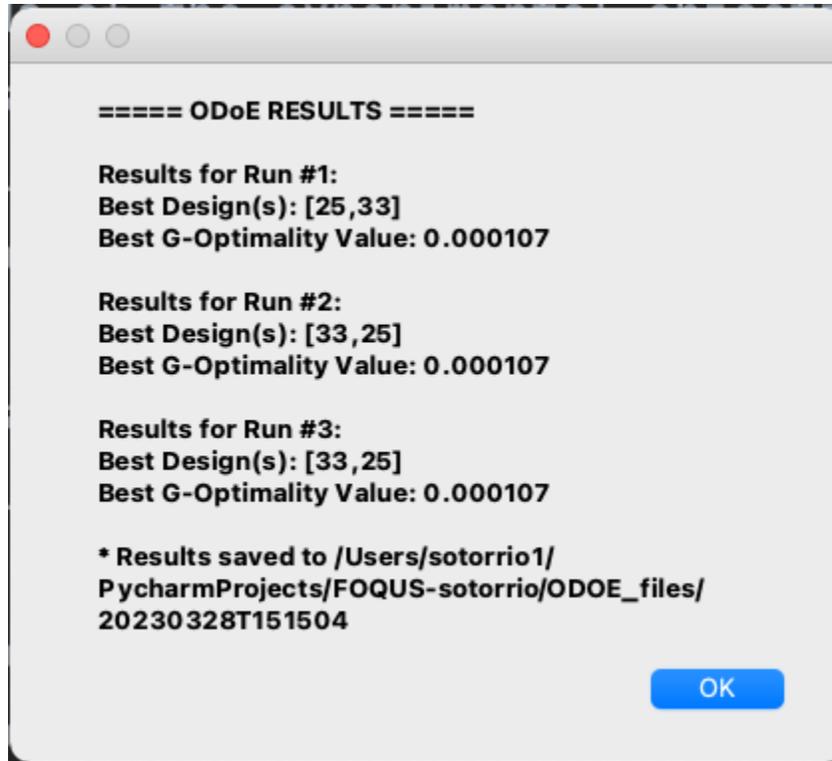


Fig. 17: ODoE Results Window

5. Click **Confirm Inputs** and the **Simulation Ensemble Setup** window will pop up to generate our **Prior Sample**.
6. Switch to the **Sampling Scheme** tab and select **Monte Carlo** and leave the **# of Samples** at **1000**. Click **Generate Samples**. When the samples are generated, **Done!** will show up right next to the button. You can visualize the samples clicking on the **Preview Samples** button. Once the user is done, click the **Done** button so the generated samples get saved.
7. Under **Design Setup** click on the **Generate New Candidate Set** button.

The **Simulation Ensemble Setup** window will pop up to generate our candidate set.

Switch to the **Sampling scheme** tab and select **Monte Carlo** and **25** samples. Click **Generate Samples**. When the samples are generated, **Done!** will show up right next to the button. You can visualize the samples clicking on the **Preview Samples** button. Once the user is done, click the **Done** button so the generated samples get saved.

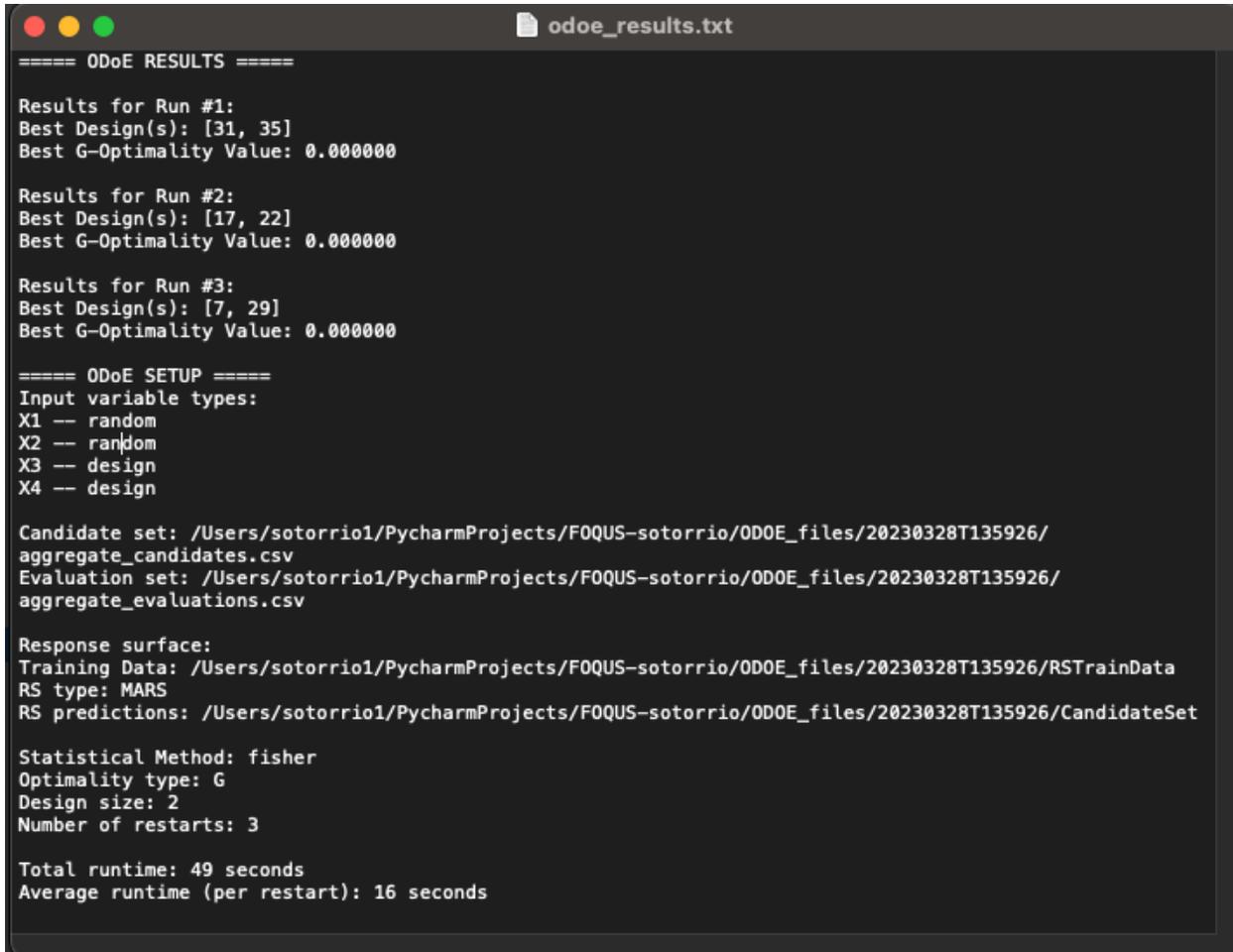
The user can select the candidate set and click on the **Delete Selection** button in case they want to delete the candidate set. To visualize the data, just click the **View** button under the **Visualize** column.

8. On the right hand side of the **Design Setup** section, click on the **Load Evaluation Set** button.

Browse and load the EvaluationSet.csv file from the examples folder. Similar to the candidate set section, the user can select the evaluation set and click on the **Delete Selection** button in case they want to delete the evaluation set. To visualize the data, just click the **View** button under the **Visualize** column.

9. Click on the **Confirm Design Setup** button and under the **Output Setup** section, select **MARS** in the **Response Surface** dropdown menu.
10. Click **Validate RS** button and the user will get a informative message window and the response surface validation plot.

If the RS selected looks good, you can click the **Confirm RS** button. The response surface predictions on



```
==== ODoE RESULTS ====

Results for Run #1:
Best Design(s): [31, 35]
Best G-Optimality Value: 0.000000

Results for Run #2:
Best Design(s): [17, 22]
Best G-Optimality Value: 0.000000

Results for Run #3:
Best Design(s): [7, 29]
Best G-Optimality Value: 0.000000

==== ODoE SETUP ====
Input variable types:
X1 -- random
X2 -- random
X3 -- design
X4 -- design

Candidate set: /Users/sotorrio1/PycharmProjects/FOQUS-sotorrio/ODoE_files/20230328T135926/
aggregate_candidates.csv
Evaluation set: /Users/sotorrio1/PycharmProjects/FOQUS-sotorrio/ODoE_files/20230328T135926/
aggregate_evaluations.csv

Response surface:
Training Data: /Users/sotorrio1/PycharmProjects/FOQUS-sotorrio/ODoE_files/20230328T135926/RSTrainData
RS type: MARS
RS predictions: /Users/sotorrio1/PycharmProjects/FOQUS-sotorrio/ODoE_files/20230328T135926/CandidateSet

Statistical Method: fisher
Optimality type: G
Design size: 2
Number of restarts: 3

Total runtime: 49 seconds
Average runtime (per restart): 16 seconds
```

Fig. 18: ODoE Results File

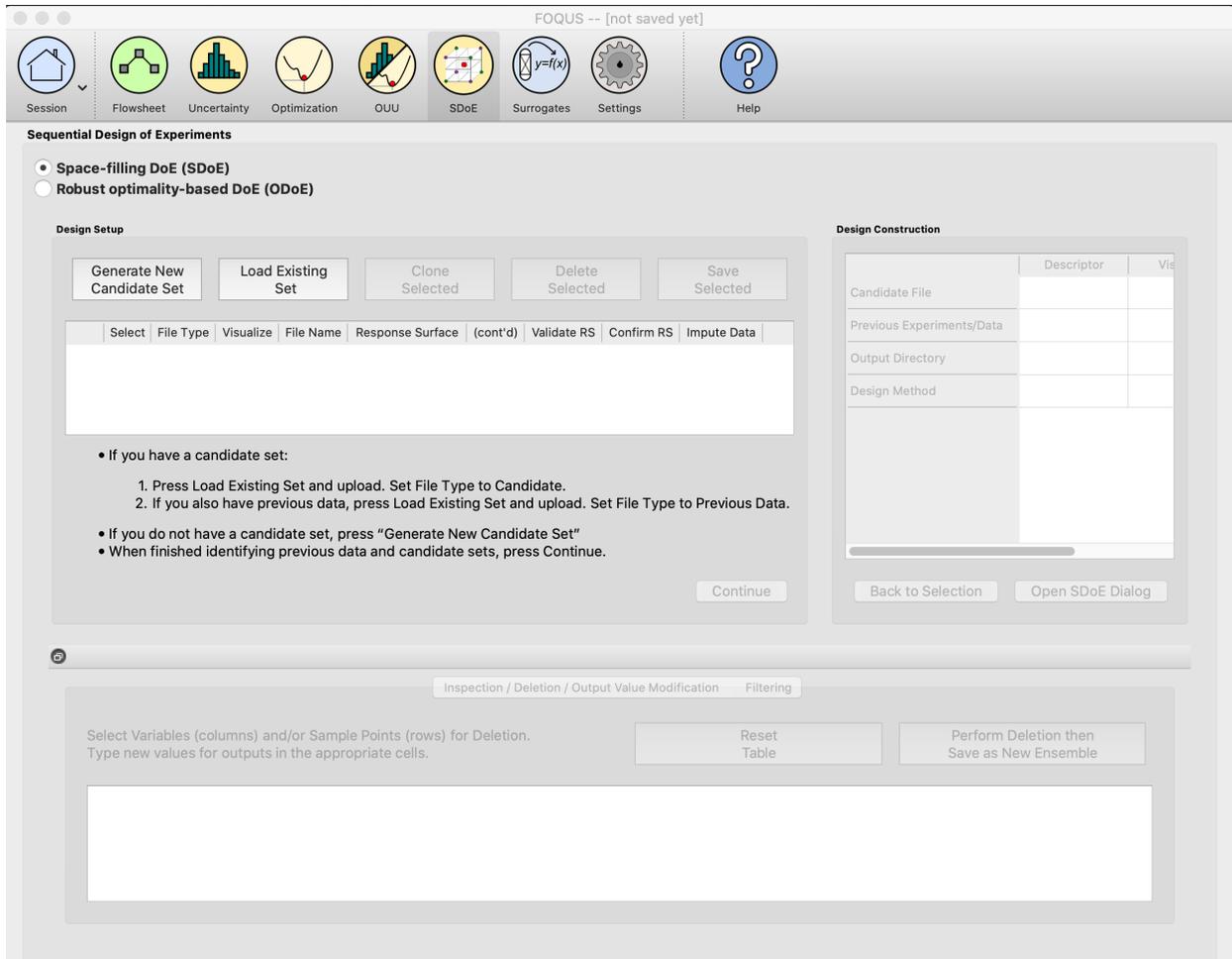


Fig. 19: SDoE Main Window

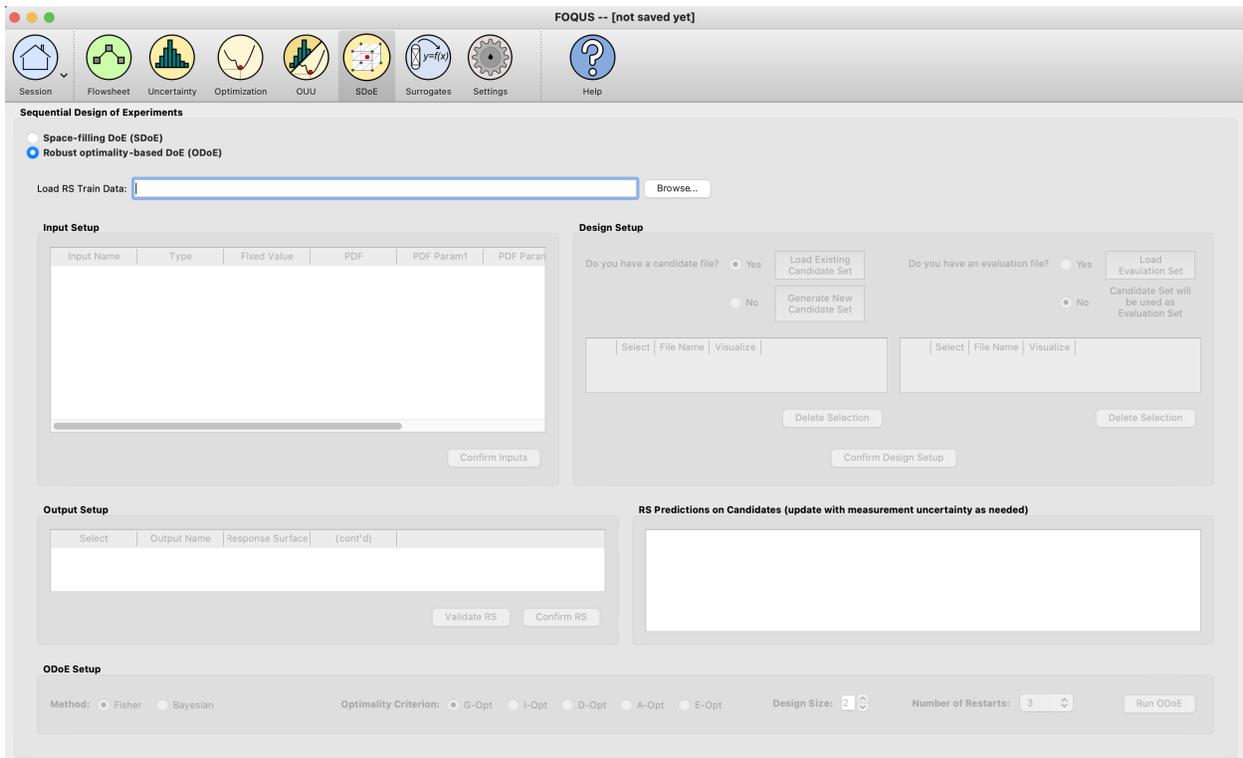


Fig. 20: ODoE Main Window

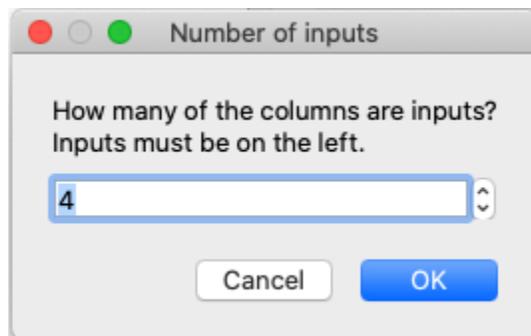


Fig. 21: ODoE Specify Number of Inputs

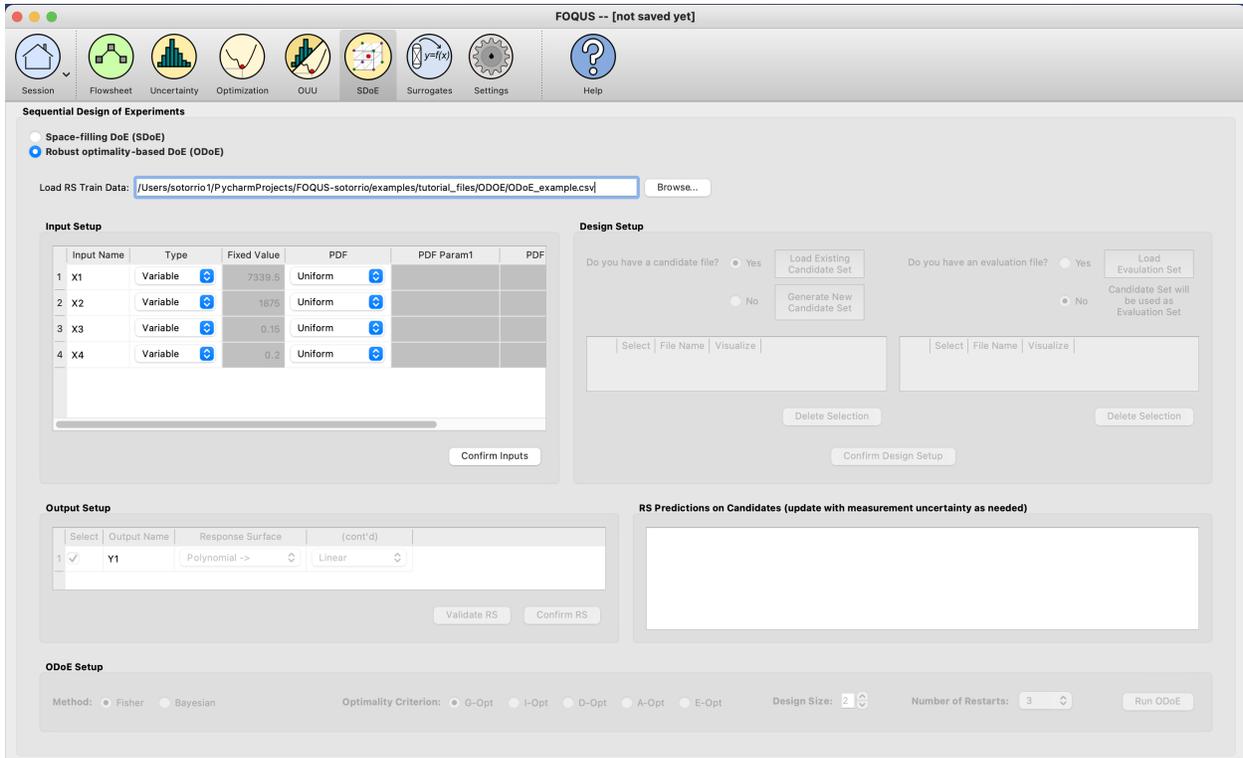


Fig. 22: ODoE Load RS Train Data

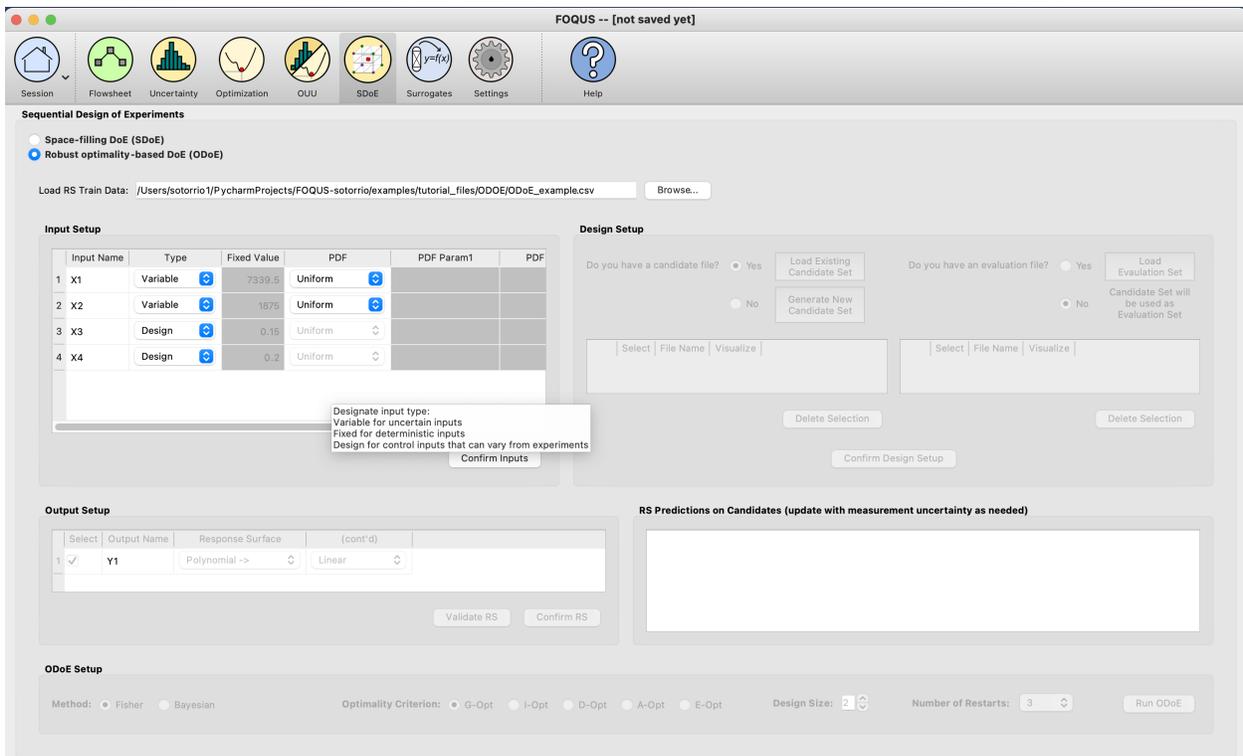


Fig. 23: ODoE Input Setup

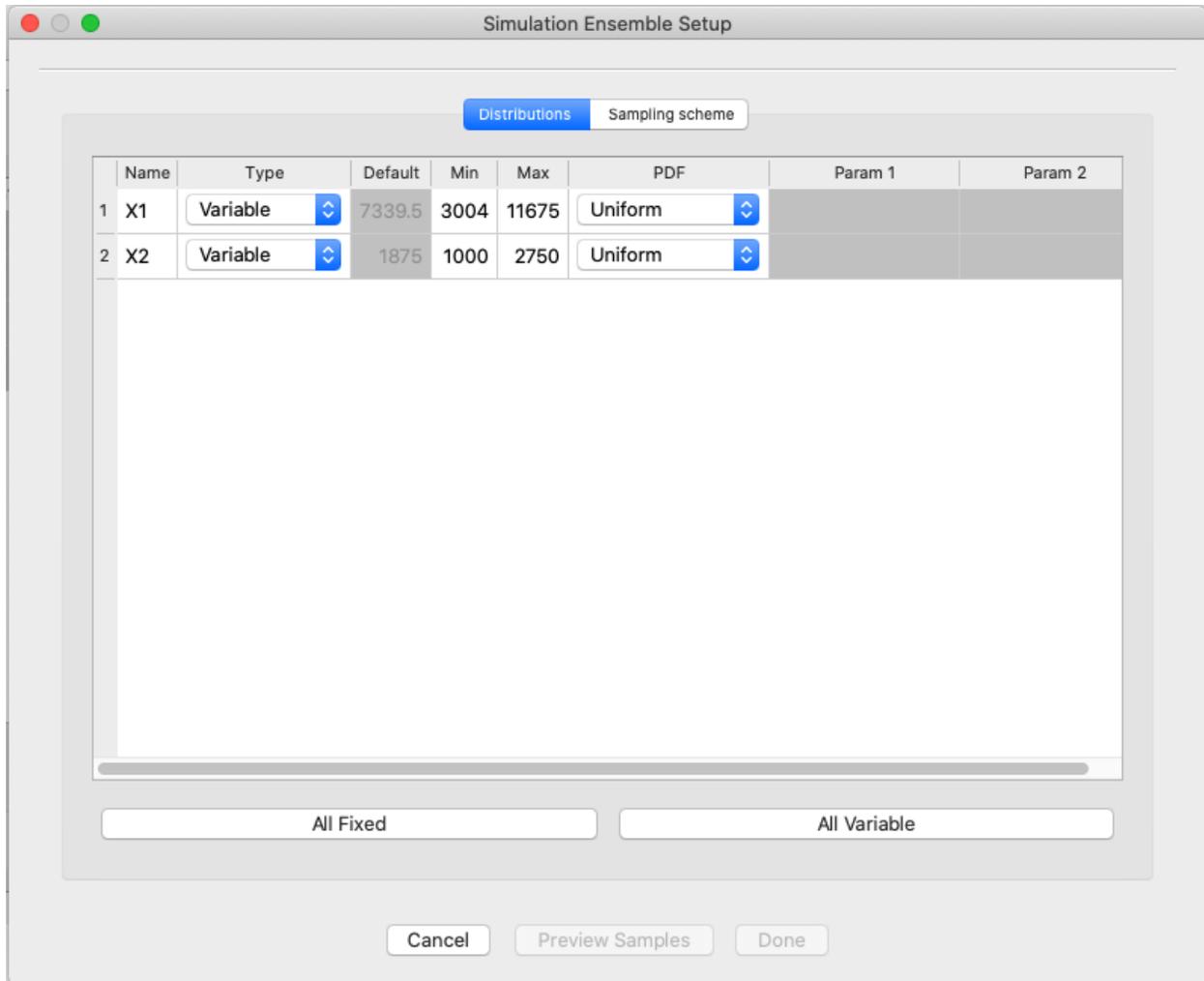


Fig. 24: ODoE Simulation Ensemble Setup for Prior Sample Generation

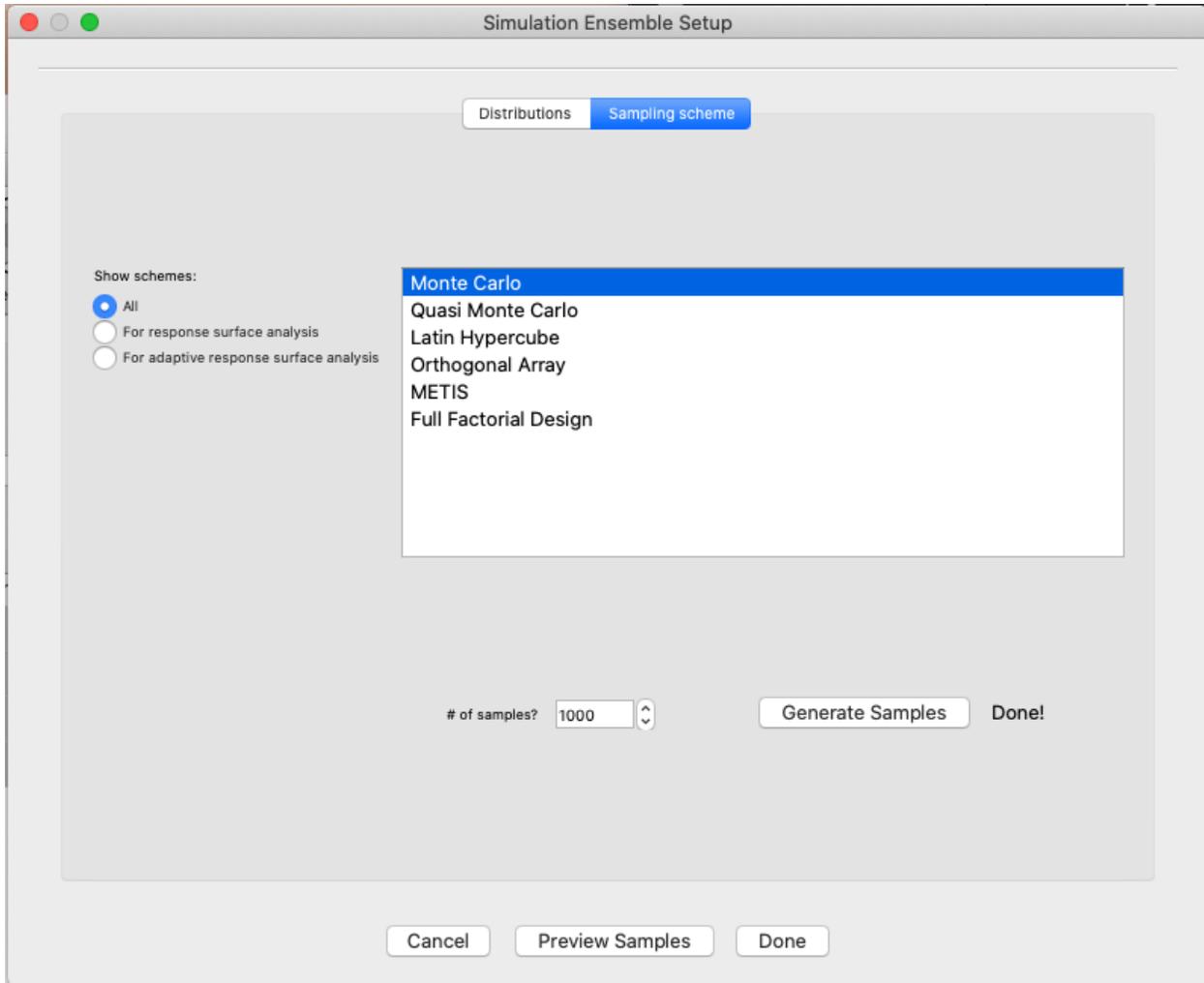


Fig. 25: ODoE Simulation Ensemble Setup for Prior Sample Generation - Sample Scheme

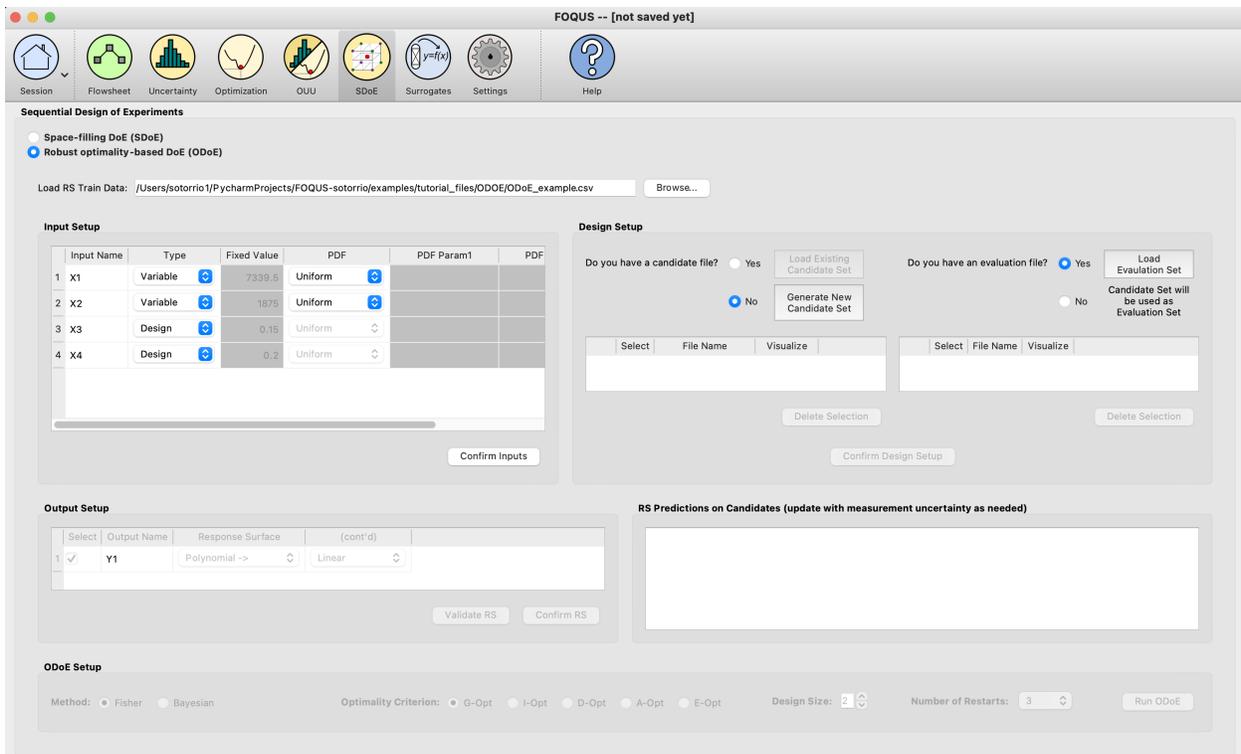


Fig. 26: ODoE Generate New Candidate Set

candidates will get populated in the table on the bottom right corner under **RS Predictions on Candidates**. The user can edit the **mean** and **standard deviation** columns in this table as needed.

- Under ODoE Setup select the **Method** (in this case Fisher), the **Optimality Criterion** (in this case G-Opt), **Design Size** (in this case 2) and **Number of Restarts** (in this case 3).

The choice of optimality criterion to use for design construction is driven by the objectives of the experimenter. If the primary focus of the experimenter is parameter estimation, then selecting the D- or A-optimality criterion is recommended. If the primary objective of the experimenter is precise prediction of the response of interest, then it is best to select the G- or I-optimality criterion. In this case, the experimenter was primarily interested in response prediction, so the G-optimal criterion was selected. Likewise, Design Size and Number of Restarts should be selected to best serve the needs of the experimental objectives. A larger design will allow more information to be collected than a smaller design, but will necessitate the use of more time and other experimental resources. The choice of design size is often dictated by the size of the experimental budget. Furthermore, the choice of Number of Restarts involves a trade-off between the quality of the design generated and the time to generate the design, with more restarts typically resulting in better designs. In this example, both design size and number of restarts were selected to fit within the given budgetary and time constraints of the experimenter.

Once those three parameters are decided, click the **Run ODoE** button. A window with PSUADE running will show up.

- Once PSUADE finishes generating the optimality-based design, another window will pop up with results information. A more thorough summary will also be saved in the **ODOE_files** directory as **odoe_results.txt**.

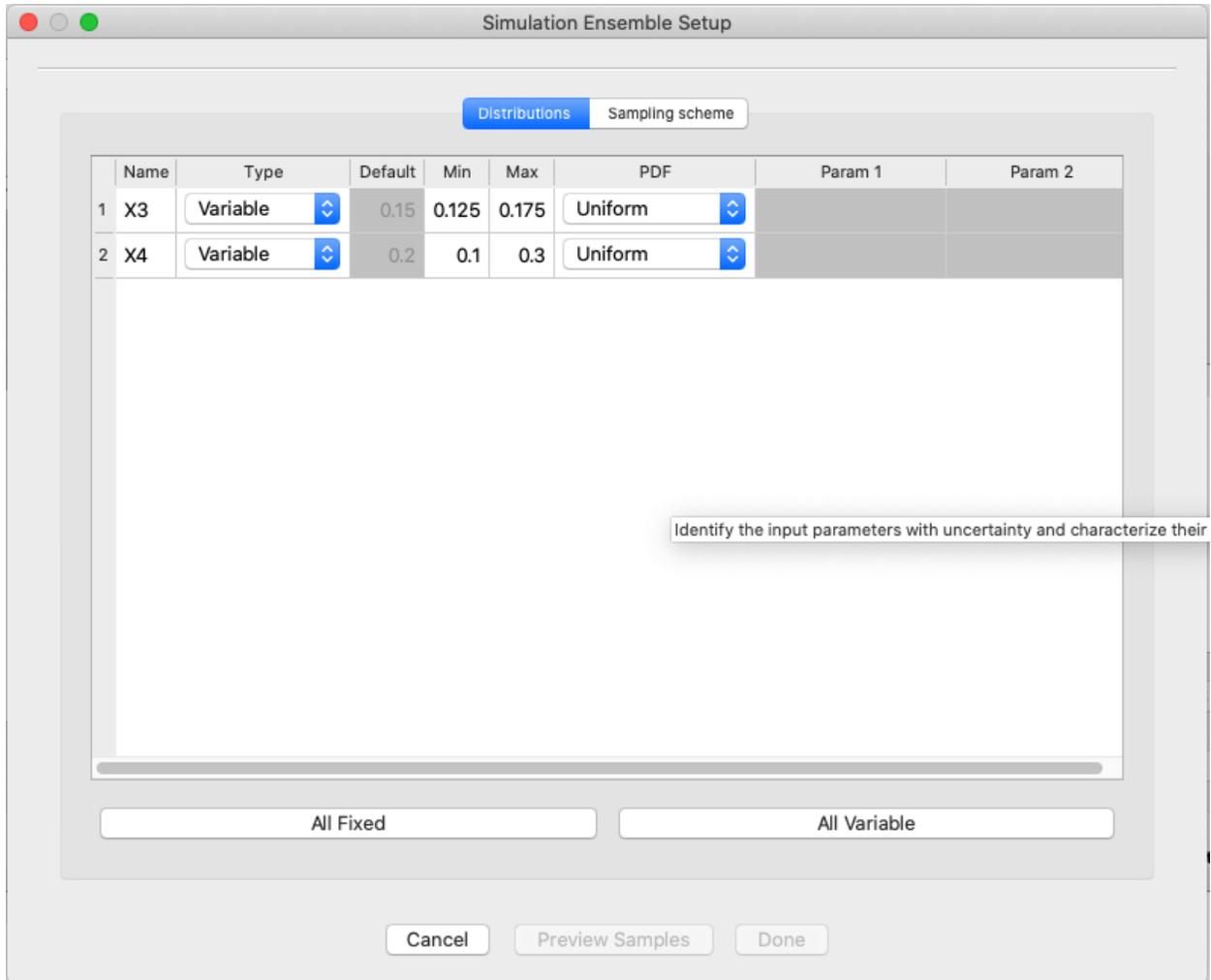


Fig. 27: ODoE Generate New Candidate Set - Simulation Ensemble Setup

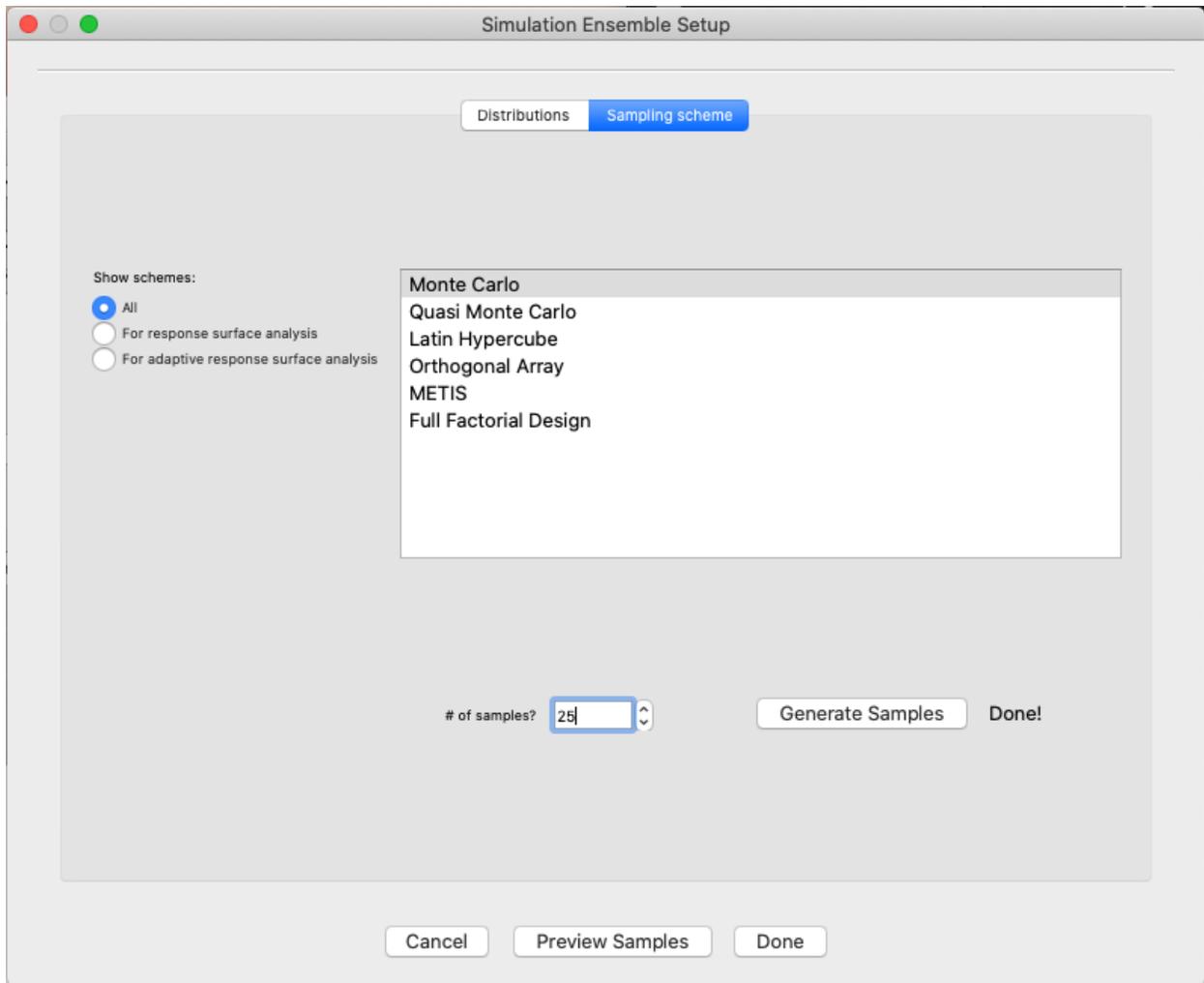


Fig. 28: ODoE Generate New Candidate Set - Sampling Scheme

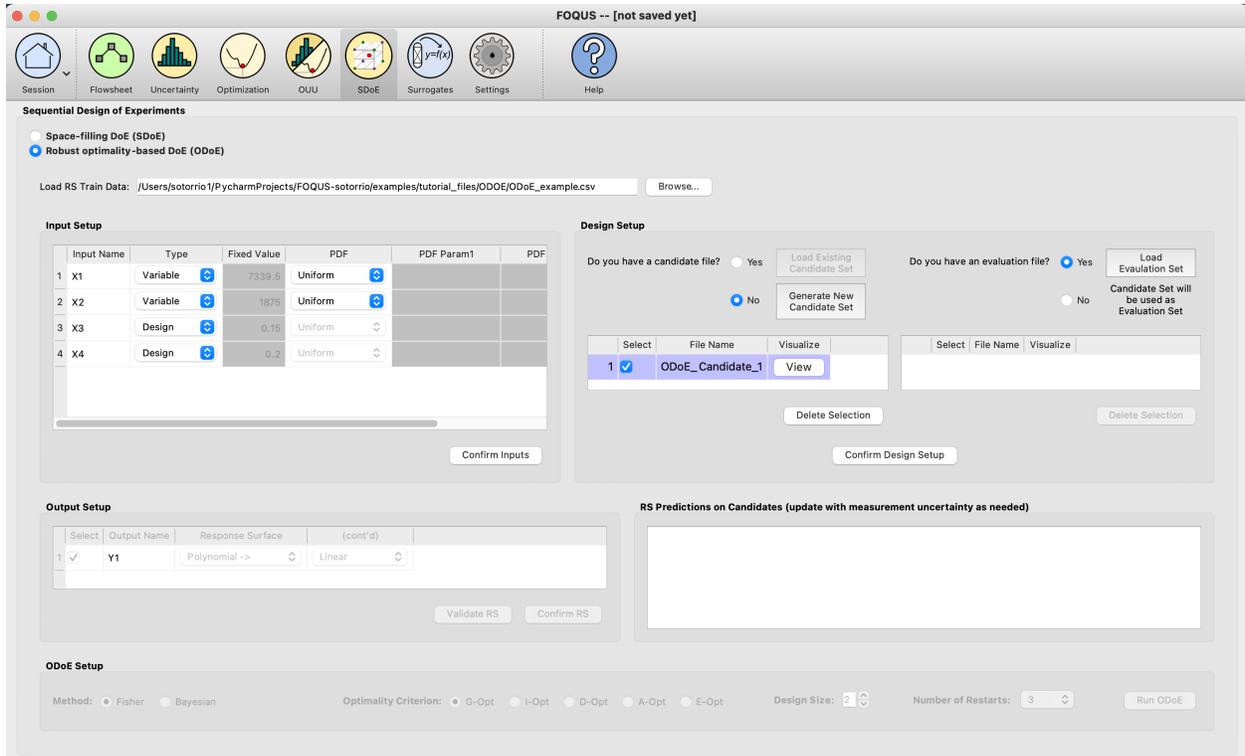


Fig. 29: ODoE Load Evaluation Set

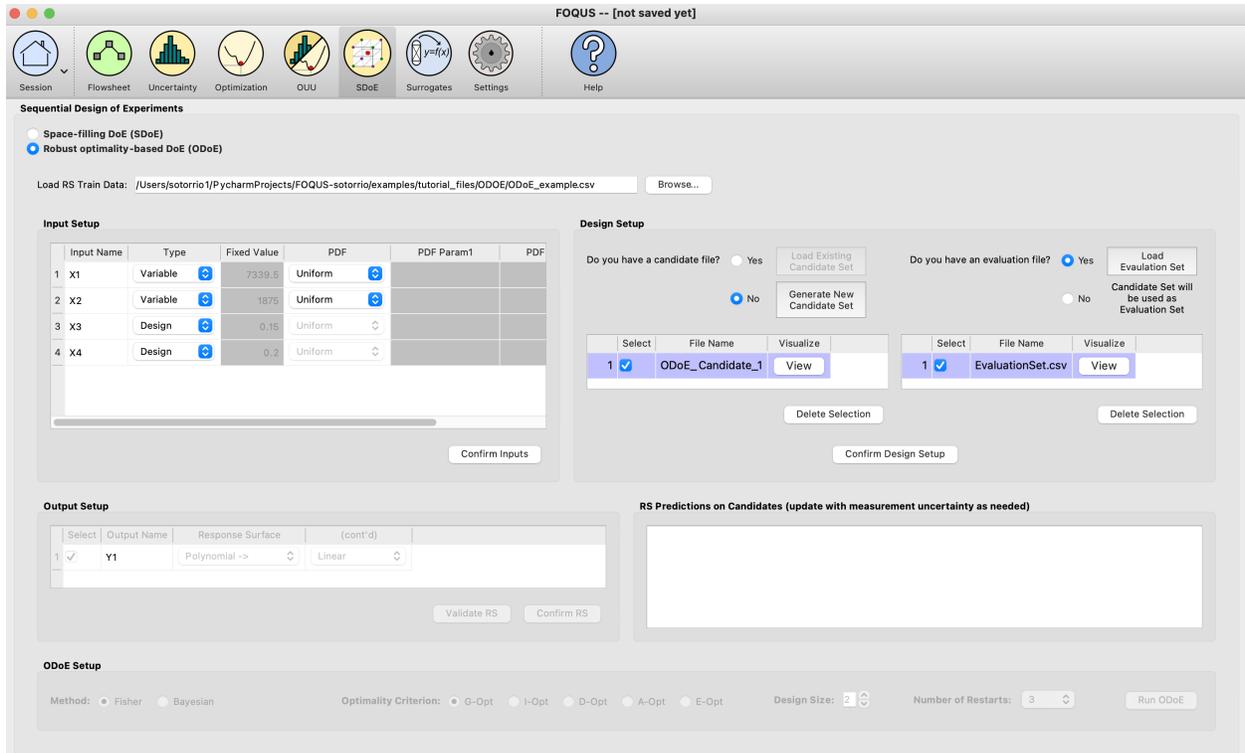


Fig. 30: ODoE Candidate and Evaluation Sets

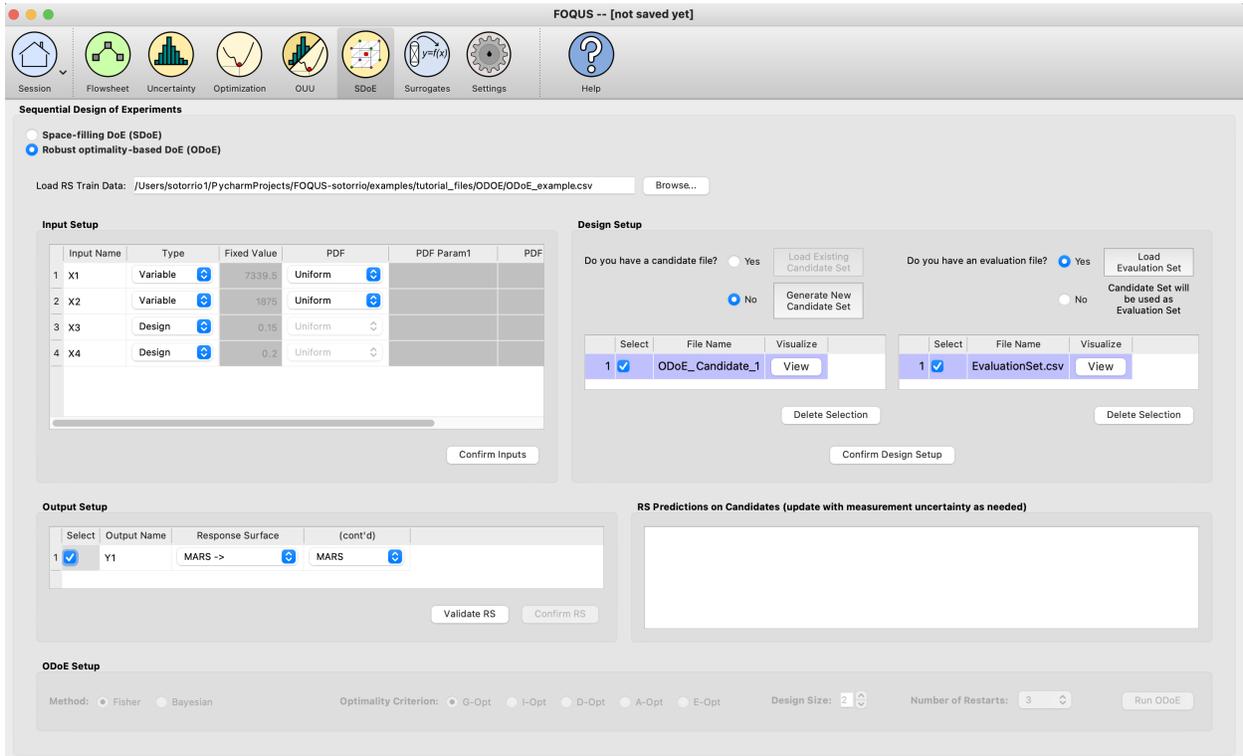


Fig. 31: ODoE Output Setup - MARS

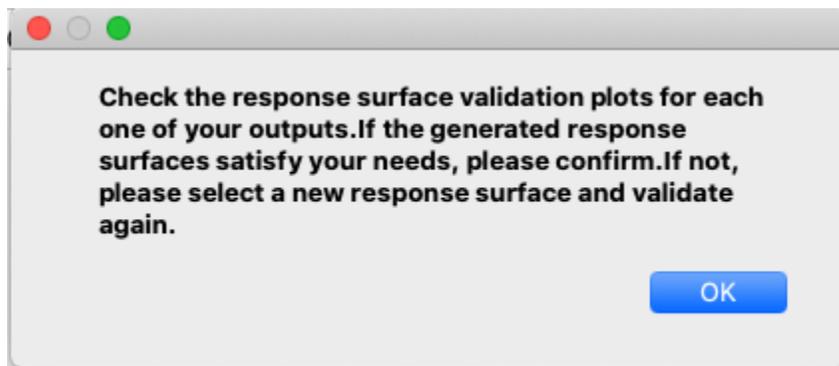


Fig. 32: ODoE Response Surface Validation Message

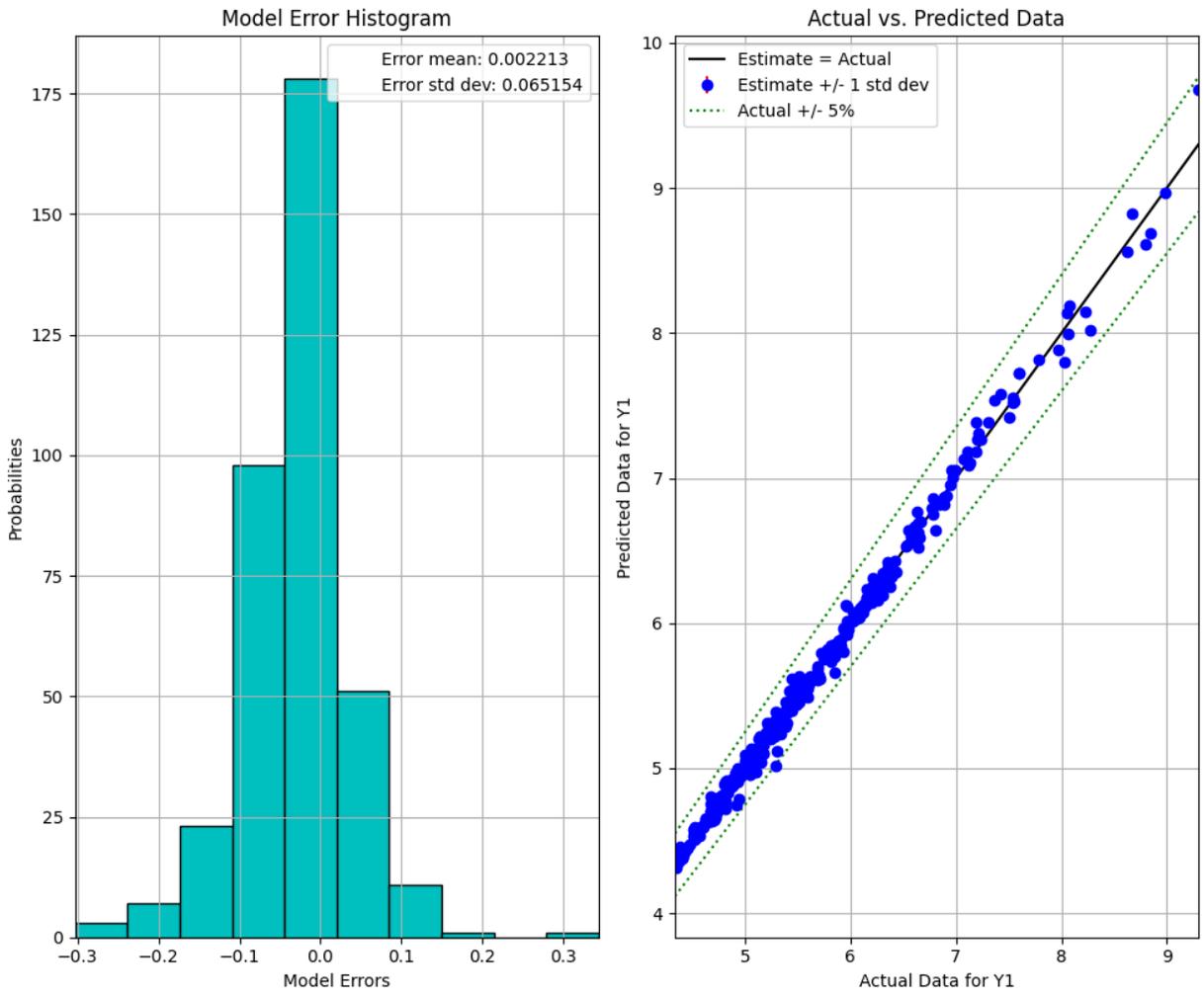


Fig. 33: ODoE Response Surface Validation Plot

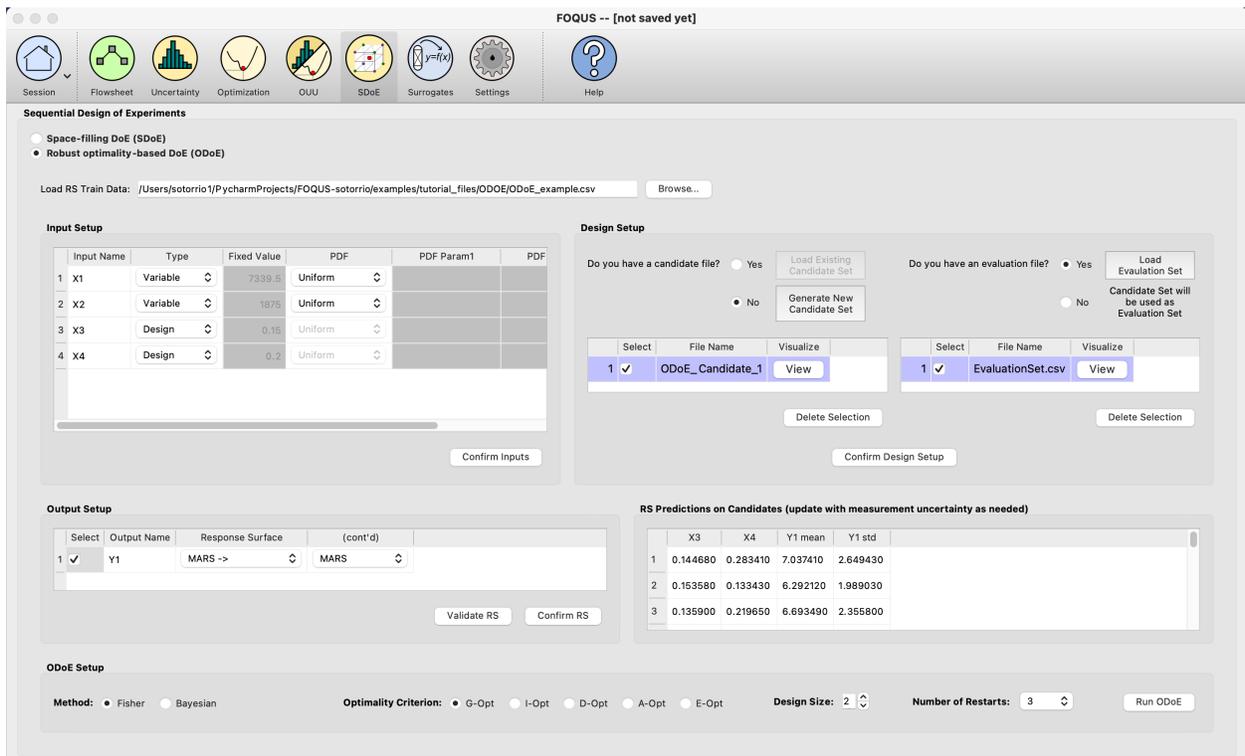
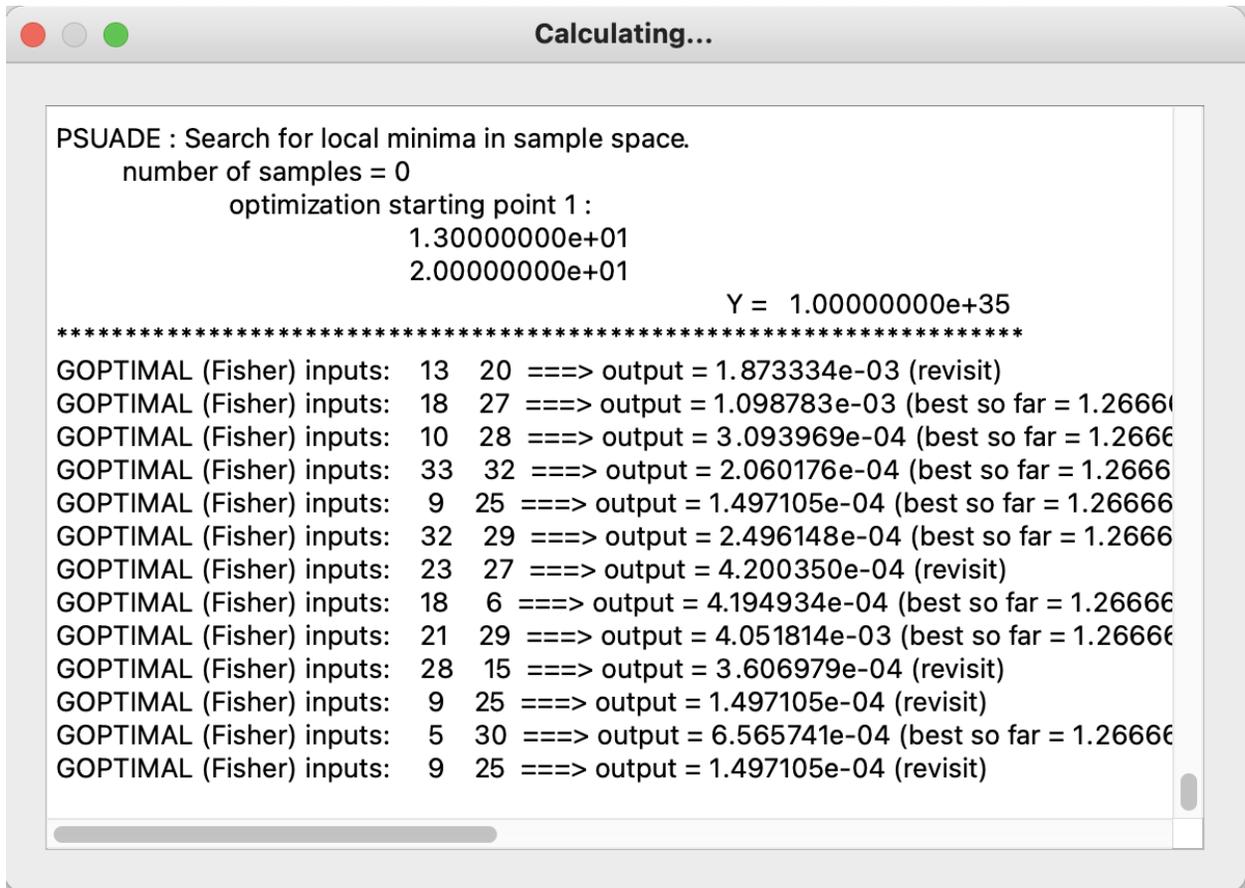


Fig. 34: ODoE Response Surface Confirmed and Predictions Generated



```
Calculating...

PSUADE : Search for local minima in sample space.
  number of samples = 0
    optimization starting point 1 :
      1.30000000e+01
      2.00000000e+01
      Y = 1.00000000e+35
*****
GOPTIMAL (Fisher) inputs: 13 20 ==> output = 1.873334e-03 (revisit)
GOPTIMAL (Fisher) inputs: 18 27 ==> output = 1.098783e-03 (best so far = 1.26666)
GOPTIMAL (Fisher) inputs: 10 28 ==> output = 3.093969e-04 (best so far = 1.26666)
GOPTIMAL (Fisher) inputs: 33 32 ==> output = 2.060176e-04 (best so far = 1.26666)
GOPTIMAL (Fisher) inputs: 9 25 ==> output = 1.497105e-04 (best so far = 1.26666)
GOPTIMAL (Fisher) inputs: 32 29 ==> output = 2.496148e-04 (best so far = 1.26666)
GOPTIMAL (Fisher) inputs: 23 27 ==> output = 4.200350e-04 (revisit)
GOPTIMAL (Fisher) inputs: 18 6 ==> output = 4.194934e-04 (best so far = 1.26666)
GOPTIMAL (Fisher) inputs: 21 29 ==> output = 4.051814e-03 (best so far = 1.26666)
GOPTIMAL (Fisher) inputs: 28 15 ==> output = 3.606979e-04 (revisit)
GOPTIMAL (Fisher) inputs: 9 25 ==> output = 1.497105e-04 (revisit)
GOPTIMAL (Fisher) inputs: 5 30 ==> output = 6.565741e-04 (best so far = 1.26666)
GOPTIMAL (Fisher) inputs: 9 25 ==> output = 1.497105e-04 (revisit)
```

Fig. 35: ODoE PSUADE Running Window

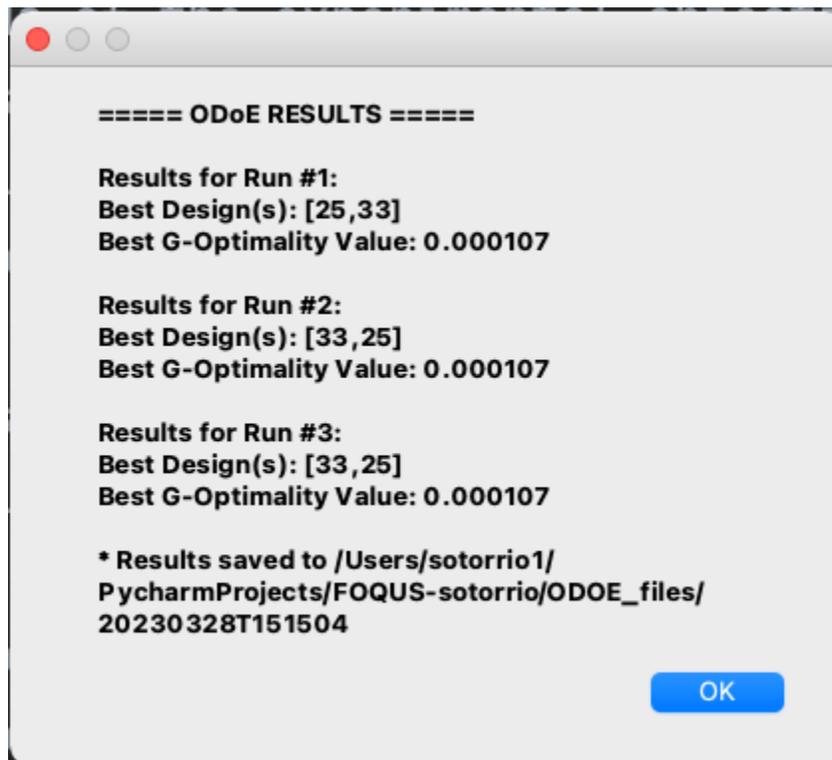
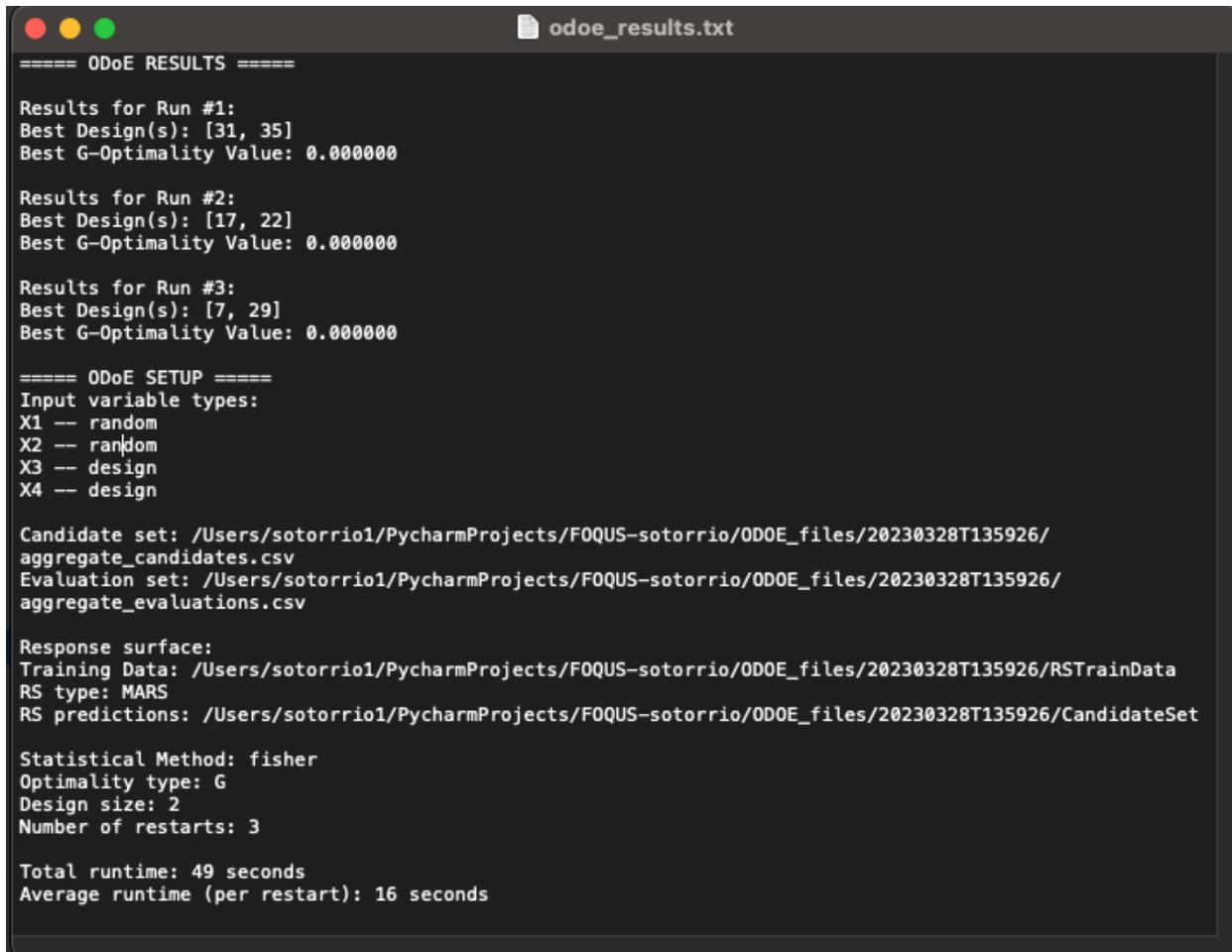


Fig. 36: ODoE Results Window



```
odoe_results.txt
==== ODoE RESULTS ====

Results for Run #1:
Best Design(s): [31, 35]
Best G-Optimality Value: 0.000000

Results for Run #2:
Best Design(s): [17, 22]
Best G-Optimality Value: 0.000000

Results for Run #3:
Best Design(s): [7, 29]
Best G-Optimality Value: 0.000000

==== ODoE SETUP ====
Input variable types:
X1 -- random
X2 -- random
X3 -- design
X4 -- design

Candidate set: /Users/sotorrio1/PycharmProjects/FOQUS-sotorrio/ODoE_files/20230328T135926/
aggregate_candidates.csv
Evaluation set: /Users/sotorrio1/PycharmProjects/FOQUS-sotorrio/ODoE_files/20230328T135926/
aggregate_evaluations.csv

Response surface:
Training Data: /Users/sotorrio1/PycharmProjects/FOQUS-sotorrio/ODoE_files/20230328T135926/RSTrainData
RS type: MARS
RS predictions: /Users/sotorrio1/PycharmProjects/FOQUS-sotorrio/ODoE_files/20230328T135926/CandidateSet

Statistical Method: fisher
Optimality type: G
Design size: 2
Number of restarts: 3

Total runtime: 49 seconds
Average runtime (per restart): 16 seconds
```

Fig. 37: ODoE Results File

HEAT INTEGRATION

10.1 Tutorial

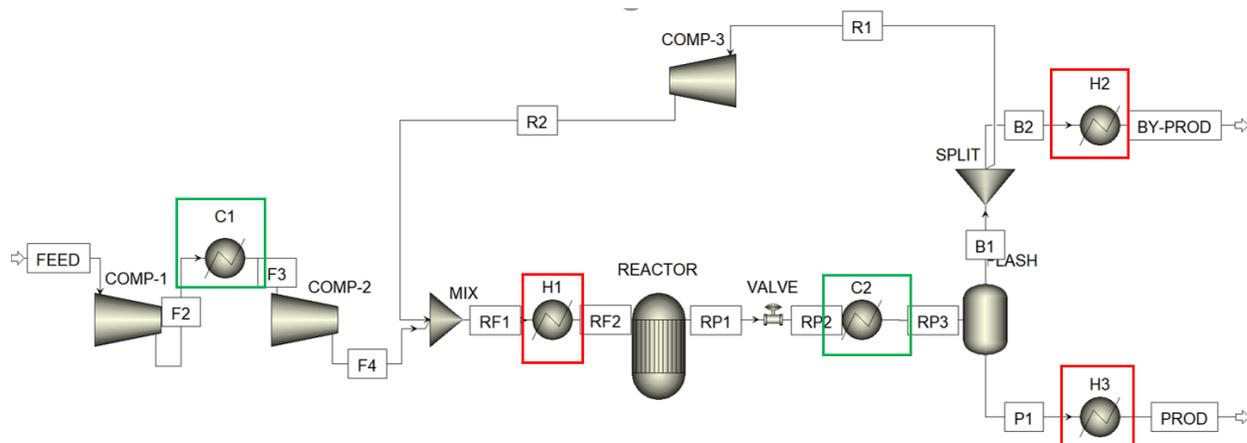
10.1.1 Tutorial: Heat Integration with FOQUS

The files for this tutorial are located in: `examples/tutorial_files/Heat_Integration`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

Motivation:

Methanol Production involves heating and cooling of process streams at different stages of the process, mainly fresh feed intercooling between compressors, mixed feed preheating before the reactor, intermediate cooling before flash, and heating of products and byproducts from the flash.



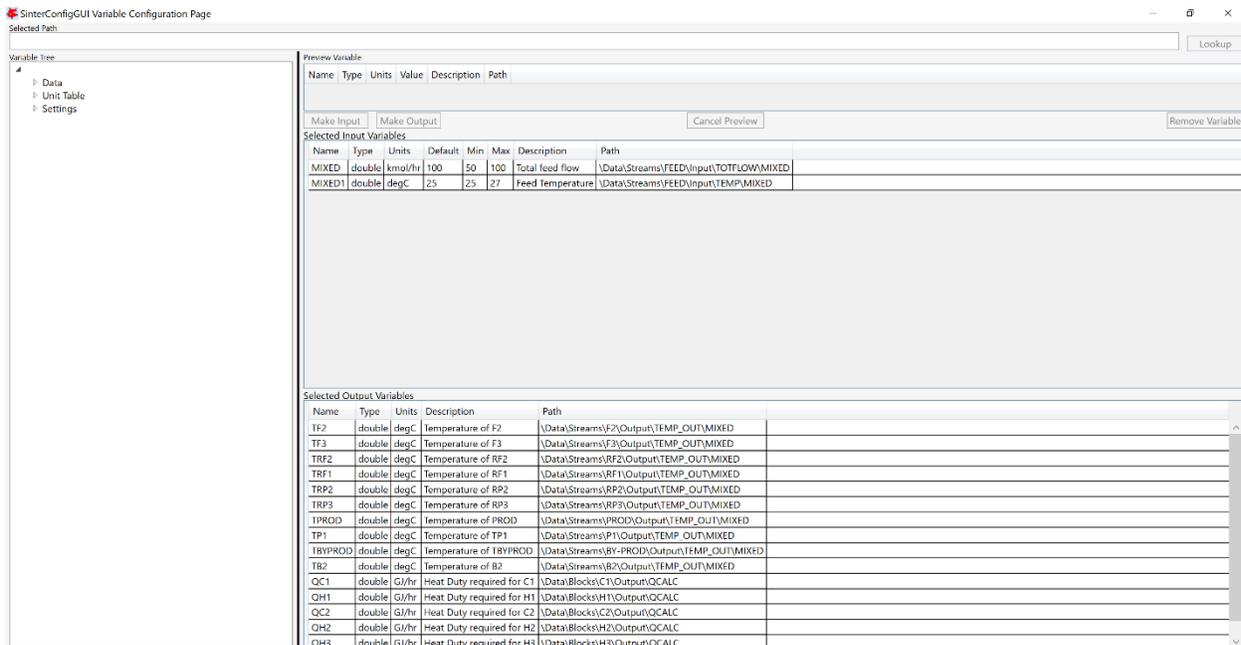
As shown in the figure above, there are 2 hot streams being cooled in C1, C2, and 3 cold streams being heated in H1,H2,H3. Clearly, there is a potential to perform heat integration among these process streams in order to minimize total utility and energy consumption while achieving the target temperatures.

Aim:

The aim of this tutorial is to implement heat integration for an Aspen Plus methanol production flowsheet, by using the heat integration plugin within FOQUS, in order to obtain the minimum utility consumption of the process.

Procedure:

1. Firstly, a SimSinter Configuration file must be created corresponding to the Aspen Plus backup file, which is located in `examples/tutorial_files/Heat_Integration`. The simulation model is available in it. Note: Ensure that Aspen v10 is used for this example. Select the fresh feed flowrate and temperature as “input variables”, and inlet, outlet temperatures of the process streams passing through all heaters and coolers (F2,F3,RF1,RF2,RP2,RP3,B2,BY-PROD,P1,PROD), along with heat duty of each heater, cooler as “output variables”.



2. Once the SimSinter file is saved in .json format, upload it to turbine and keep the simulation name as “MethanolHI”.
3. In the Flowsheet Window, add a node named “methanol_HI” which would contain the simulation.
4. Open the node editor for the given node, select model type as “Turbine” and model as “MethanolHI”. All the selected input and output variables of the simulation should be visible in the GUI.
5. Add heat integration tags beside each output variable. In this case, the order of tags for heat duty and temperature variables is as follows: Heat Duty of Heaters/Coolers: [“Block name”, “Blk_Var”, “heater”, “Q”] Where name is the block name of each heater/cooler in the Aspen model.

Inlet/Outlet temperatures: [“Block name”, “Port_Material_In/Out”, “heater”, “T”] Where name is the block name of the heater/cooler in the Aspen model, associated with the concerned inlet/outlet stream.

NOTE: Ensure that all the variables are of the type “float” in the GUI

6. Run the flowsheet simulation node for testing once. The heat integration tags for output variables are seen in the rightmost column of the node editor, as shown below:
7. Add another node to the flowsheet window named “HI”

Node Edit

Name: Visible

Error Status

Code:

Message:

Model

Type: Model:

Input Variables

Output Variables

[Tags](#)

	Name	Value	Unit	Type	Description	Tags
1	QC1	-0.241522542	GJ/hr	float	Heat Duty required for C1	["Block C1", "Blk_Var", "heater", "Q"]
2	QC2	-2.23267236	GJ/hr	float	Heat Duty required for C2	["Block C2", "Blk_Var", "heater", "Q"]
3	QH1	1.0500302	GJ/hr	float	Heat Duty required for H1	["Block H1", "Blk_Var", "heater", "Q"]
4	QH2	0.0532461406	GJ/hr	float	Heat Duty required for H2	["Block H2", "Blk_Var", "heater", "Q"]
5	QH3	0.237281192	GJ/hr	float	Heat Duty required for H3	["Block H3", "Blk_Var", "heater", "Q"]
6	status	0.0		float	Simulation Status Code	[]
7	TB2	50.0000928	degC	float	Temperature of B2	["Block H2", "Port_Material_In", "heater", "T"]
8	TBYPROD	125.0	degC	float	Temperature of TBYPROD	["Block H2", "Port_Material_Out", "heater", "T"]
9	TF2	111.0177	degC	float	Temperature of F2	["Block C1", "Port_Material_In", "heater", "T"]
10	TF3	30.0	degC	float	Temperature of F3	["Block C1", "Port_Material_Out", "heater", "T"]
11	TP1	50.0000928	degC	float	Temperature of TP1	["Block H3", "Port_Material_In", "heater", "T"]
12	TPROD	125.0	degC	float	Temperature of PROD	["Block H3", "Port_Material_Out", "heater", "T"]
13	TRF1	90.2717798	degC	float	Temperature of RF1	["Block H1", "Port_Material_In", "heater", "T"]

8. Open the node editor for it, and enter the heat integration plugin. In its input variables, enter number of streams as 5. Keep all other input values default.
9. Connect both the nodes through an edge connector.
10. Run the flowsheet simulation.

Result:

Minimum Utility Requirements: Q (cooling water) = 1.2346 GJ/hr Q (IP Steam) = 0.101 GJ/hr

The screenshot shows the FOQUS software interface. The main window displays a flowsheet with a single node connected to a heat integration plugin. The node editor window is open, showing the following output variables table:

	Name	Value	Unit	Type	Description
1	Capital.Cost	0.0959749553324946	\$MM	float	Approximated capital cost for heat exchanger ne
2	Cooling.Water.Consumption	1.2346	GJ/hr	float	Cooling water (20 C) consumption (Cost: \$0.21/k
3	FH.Heat.Addition1	NaN	GJ/hr	float	Heat addition to feed water heater 1
4	FH.Heat.Addition2	NaN	GJ/hr	float	Heat addition to feed water heater 2
5	FH.Heat.Addition3	NaN	GJ/hr	float	Heat addition to feed water heater 3
6	FH.Heat.Addition4	NaN	GJ/hr	float	Heat addition to feed water heater 4
7	FH.Heat.Addition5	NaN	GJ/hr	float	Heat addition to feed water heater 5
8	Heat.Exchanger.Area	244.6738	m^2	float	Heat exchanger area
9	IP.Steam.Consumption	0.101	GJ/hr	float	Intermediate pressure steam (230 C) consumpti
10	LP.Steam.Consumption	0.0	GJ/hr	float	Low pressure steam (164 C) consumption (Cost:
11	Total.Cost	0.04085237535679426	\$MM/yr	float	Approximated total annualized cost for heat exc
12	Utility.Cost	0.0085688	\$MM/yr	float	Utility cost

PYOMO-FOQUS

11.1 Tutorial

11.1.1 Tutorial: Running PYOMO Optimization Model in FOQUS

Consider the following optimization problem to be solved with FOQUS using PYOMO.

$$\begin{aligned} & \min y \\ & \text{Subject to:} \\ & y = x_1 + x_2 \\ & ax_1 + bx_2 \geq c \end{aligned}$$

The complete FOQUS file (**Pyomo_Test_Example.foqus**), with the code written, is located in:
`examples/tutorial_files/PYOMO`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

Instructions

1. Open FOQUS, and under the Flowsheet Tab, create a Node.
2. Open the Node Editor, and let the Model Type be “None”.
3. Add the model parameters `a`, `b`, `c` as “Input Variables” within the Node Editor, with values 1, 2, 3 respectively.
4. Add `x1`, `x2`, `y`, `converged`, and `optimal` as “Output Variables” within the Node Editor.

Note that `x1`, `x2` and `y` correspond to the optimization variable values.

`converged` is meant to be a binary variable that would denote whether the optimization model has converged, by checking the solver status.

`optimal` is meant to be a binary variable that would denote whether the solver returns an optimal solution.

5. Under Node Script, set Script Mode to “Post”. This will ensure that the node script runs after the node simulation. Enter the following PYOMO code for the optimization model:

```
1 from pyomo.environ import (Var,  
2                             Constraint,  
3                             ConcreteModel,
```

(continues on next page)

(continued from previous page)

```

4         PositiveReals,
5         Objective)
6 from pyomo.opt import SolverFactory
7 import pyutilib.subprocess.GlobalData
8
9 pyutilib.subprocess.GlobalData.DEFINE_SIGNAL_HANDLERS_DEFAULT = False
10 m = ConcreteModel()
11 m.x1 = Var(within=PositiveReals)
12 m.x2 = Var(within=PositiveReals)
13 m.y = Var()
14 m.c1 = Constraint(expr=x["a"]*m.x1+x["b"]*m.x2 >= x["c"])
15 m.c2 = Constraint(expr=m.x1+m.x2 == m.y)
16 m.o = Objective(expr=m.y)
17 opt = SolverFactory("ipopt")
18 r = opt.solve(m)
19 f["x1"] = m.x1.value
20 f["x2"] = m.x2.value
21 f["y"] = m.y.value
22 f["converged"] = (str(r.solver.status) == "ok")
23 f["optimal"] = (str(r.solver.termination_condition) == "optimal")

```

In the above code, lines 1-6 are used to import the PYOMO package and SolverFactory function to develop the model and solve it by accessing an appropriate solver.

A PYOMO Concrete Model is declared, defining the variables, declaring the constraints using the parameters defined within “Input Variables” of the Node, and defining the objective function with lines 10 to 16.

Line 17 sets the solver to ipopt and line 18 sends the problem to be solved to the solver. Ipopt is a nonlinear optimization solver.

Note: ipopt will need to be available in your environment. To install it into your conda environment you should use the command: `conda install -c conda-forge ipopt` The conda install method is preferred for Windows users.

Once the model is solved, the values of decision variables x1, x2, y are assigned to the Node Output Variables in lines 19 to 21.

The code lines 22 and 23 check the solver status and termination condition. If the solver status is “ok”, it means that the model has converged, and the ‘converged’ variable is assigned the value 1. Else, it is assigned the value 0, which means that the model has not converged. If the solver termination condition is “optimal”, it means that the solver has found an optimal solution for the optimization model. Else, the solution is either feasible if the solver status is “ok”, or infeasible altogether.

6. Click the Run button to run the python script and check the Node Output Variables section.

It should be noted that the parameter values within Node Input Variables can be changed as per user’s requirement, to run different cases.

Note: For more information on building and solving pyomo models, refer to the pyomo documentation: https://pyomo.readthedocs.io/en/stable/solving_pyomo_models.html

IDAES-FOQUS

12.1 Tutorial

12.1.1 Tutorial: Running IDAES model in FOQUS

The NETL's Institute for the Design of Advanced Energy Systems is developing an equation-oriented framework for simulation and optimization of energy systems. A library of unit models is available to create and solve process flowsheets, therefore, a tutorial has been developed in FOQUS to import IDAES unit models, build a flowsheet, and simulate it.

The case study consists of the separation of Toluene-Benzene mixture (Figure 1). First the mixture is heated to 370K, and then separated in the Flash Tank. Consider the following process flowsheet that has been developed in FOQUS, using IDAES :

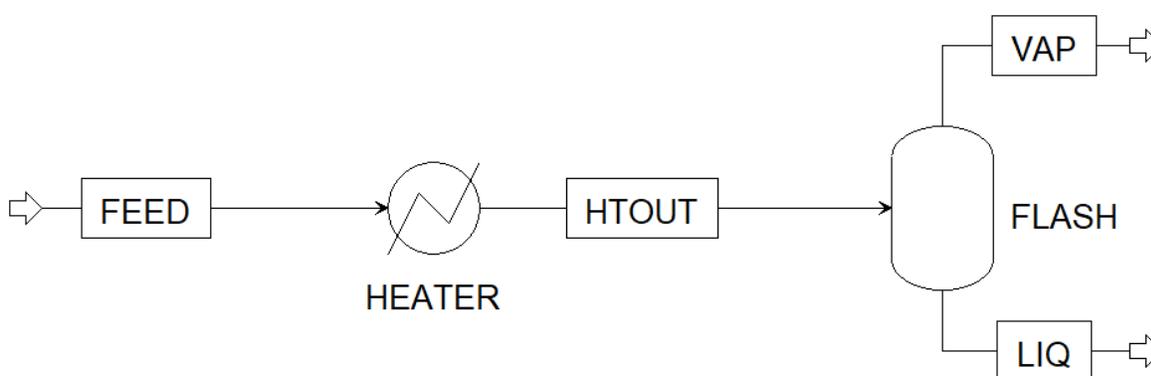


Fig. 1: Figure 1: Heater Flash Flowsheet

Feed Conditions:

Flowrate = 0.277778 mol/s

Temperature = 353 K

Pressure = 101325 K

Benzene Mole Fraction = 0.4

Toluene Mole Fraction = 0.6

Heater Specification:

Outlet Temperature (HTOUT stream) = 370 K

Flash Specification:

Heat Duty = 0 W

Pressure Drop = 0 Pa

The following steps show how to import Python, Pyomo, and IDAES libraries and models, build the flowsheet, select input variables, and solve the simulation in FOQUS:

Instructions

1. Open FOQUS, and under the Flowsheet Tab, create a Node named “Flowsheet”.
2. Open the Node Editor and let the Model Type be “None”.
3. Add the following input variables with their corresponding values in the Node Editor: heater_inlet_molflow: 0.277778 mol/s
heater_inlet_pressure: 101325 Pa
heater_inlet_temperature: 353 K
heater_inlet_benzene_molfrac: 0.4
heater_inlet_toluene_molfrac: 0.6
heater_outlet_temperature: 370 K
flash_heat_duty: 0 W
flash_pressure_drop: 0 Pa
4. Add the following output variables in the Node Editor: heater_heat_duty W
flash_liq_molflow mol/s
flash_liq_pressure Pa
flash_liq_temperature K
flash_liq_benzene_molfrac
flash_liq_toluene_molfrac
flash_vap_molflow mol/s
flash_vap_pressure Pa
flash_vap_temperature K
flash_vap_benzene_molfrac
flash_vap_toluene_molfrac
5. As stated in previous tutorials, the FOQUS simulation node allows the user to type a python script under the Node Script option. In this node script section, this tutorial shows how to import python libraries, Pyomo libraries, IDAES libraries and models, build and solve the flowsheet. Note that in this example, process conditions are fixed in order to have 0 degrees of freedom. Hence, the optimization actually gets solved as a simulation problem. A critical step is to link the FOQUS variables (input and output) to the IDAES mathematical model, thus, setting the inlet conditions of the process before solving the simulation problem. Finally, under Node Script, set Script Mode to “Post”. This will ensure that the node script runs after the node simulation. Enter the following code:

```

1 # Import objects from pyomo package
2 from pyomo.environ import ConcreteModel, SolverFactory, TransformationFactory, value
3
4 import pyutilib.subprocess.GlobalData
5 pyutilib.subprocess.GlobalData.DEFINE_SIGNAL_HANDLERS_DEFAULT = False
6
7 # Import the main FlowsheetBlock from IDAES. The flowsheet block will contain the
8   ↳ unit model
9
10 import idaes
11 from idaes.core.flowsheet_model import FlowsheetBlock
12
13 # Import the BTX_ideal property package to create a properties block for the
14   ↳ flowsheet
15 from idaes.generic_models.properties.activity_coeff_models import BTX_activity_
16   ↳ coeff_VLE
17
18 # Import heater unit model from the model library
19 from idaes.generic_models.unit_models.heater import Heater
20
21 # Import flash unit model from the model library
22 from idaes.generic_models.unit_models.flash import Flash
23
24 # Import methods for unit model connection and flowsheet initialization
25 from pyomo.network import Arc, SequentialDecomposition
26
27 # Import idaes logger to set output levels
28 import idaes.logger as idaeslog
29
30 # Create the ConcreteModel and the FlowsheetBlock, and attach the flowsheet block
31   ↳ to it.
32 m = ConcreteModel()
33
34 m.fs = FlowsheetBlock(default={"dynamic": False}) # dynamic or ss flowsheet needs
35   ↳ to be specified here
36
37 # Add properties parameter block to the flowsheet with specifications
38 m.fs.properties = BTX_activity_coeff_VLE.BTXParameterBlock(default={"valid_phase":
39   ('Liq', 'Vap'),
40   "activity_coeff_model":
41   "Ideal"})
42
43 # Create an instance of the heater unit, attaching it to the flowsheet
44 # Specify that the property package to be used with the heater is the one we
45   ↳ created earlier.
46 m.fs.heater = Heater(default={"property_package": m.fs.properties})
47
48 m.fs.flash = Flash(default={"property_package": m.fs.properties})
49
50 # Connect heater and flash models using an arc
51 m.fs.heater_flash_arc = Arc(source=m.fs.heater.outlet, destination=m.fs.flash.inlet)
52
53 TransformationFactory("network.expand_arcs").apply_to(m)

```

(continues on next page)

(continued from previous page)

```

48
49 #Feed Specifications to heater
50 m.fs.heater.inlet.flow_mol.fix(x["heater_inlet_molflow"]) # mol/s
51 m.fs.heater.inlet.mole_frac_comp[0, "benzene"].fix(x["heater_inlet_benzene_molfrac
↪"])
52 m.fs.heater.inlet.mole_frac_comp[0, "toluene"].fix(x["heater_inlet_toluene_molfrac
↪"])
53 m.fs.heater.inlet.pressure.fix(x["heater_inlet_pressure"]) # Pa
54 m.fs.heater.inlet.temperature.fix(x["heater_inlet_temperature"]) # K
55
56 # Unit model specifications
57 m.fs.heater.outlet.temperature.fix(x["heater_outlet_temperature"]) # K
58 m.fs.flash.heat_duty.fix(x["flash_heat_duty"]) # W
59 m.fs.flash.deltaP.fix(x["flash_pressure_drop"]) # Pa
60
61 #Flowsheet Initialization
62 def function(unit):
63     unit.initialize(outlvl=1)
64
65 opt = SolverFactory('ipopt')
66 seq = SequentialDecomposition()
67 seq.options.select_tear_method = "heuristic"
68 seq.run(m, function)
69
70 # Solve the flowsheet using ipopt
71 opt = SolverFactory('ipopt')
72 solve_status = opt.solve(m)
73
74 #Assign the simulation result from IDAES model to FOQUS output values
75 f["flash_liq_molflow"] = value(m.fs.flash.liq_outlet.flow_mol[0])
76 f["flash_liq_benzene_molfrac"] = value(m.fs.flash.liq_outlet.mole_frac_comp[0,
↪"benzene"])
77 f["flash_liq_toluene_molfrac"] = value(m.fs.flash.liq_outlet.mole_frac_comp[0,
↪"toluene"])
78 f["flash_liq_temperature"] = value(m.fs.flash.liq_outlet.temperature[0])
79 f["flash_liq_pressure"] = value(m.fs.flash.liq_outlet.pressure[0])
80 f["flash_vap_molflow"] = value(m.fs.flash.vap_outlet.flow_mol[0])
81 f["flash_vap_benzene_molfrac"] = value(m.fs.flash.vap_outlet.mole_frac_comp[0,
↪"benzene"])
82 f["flash_vap_toluene_molfrac"] = value(m.fs.flash.vap_outlet.mole_frac_comp[0,
↪"toluene"])
83 f["flash_vap_temperature"] = value(m.fs.flash.vap_outlet.temperature[0])
84 f["flash_vap_pressure"] = value(m.fs.flash.vap_outlet.pressure[0])
85 f["heater_heat_duty"] = value(m.fs.heater.heat_duty[0])

```

Note: ipopt will need to be available in your environment. This should be available through the following command during the generic install of IDAES in the environment: `idaes get-extensions`

Once the model is solved, the values of flowsheet output variables are assigned to the node output variables.

- Click the Run button to run the python script and check the node output variables section, note that their values should have changed.

It should be noted that the values within Node Input Variables can be changed as per user's requirement, to run different cases.

Note: For more information on installing IDAES, along with building and solving IDAES models, refer to the IDAES documentation: <https://idaes-pse.readthedocs.io/en/stable/index.html>

This tutorial demonstrates the capability of simulating IDAES based process models in FOQUS. However, optimization problems can also be solved using IDAES in FOQUS, by providing the required degrees of freedom.

It is recommended that FOQUS and IDAES must be installed in the same conda environment for this example to run successfully.

The complete FOQUS file (**FOQUS_IDAES_Example.foqus**), that includes the IDAES model, is located in: `examples/tutorial_files/IDAES`. The **examples/** directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

FOQUS-MATLAB

13.1 Contents

13.1.1 MATLAB-FOQUS

interface

Introduction

MATLAB® is a proprietary interpreted programming language developed by MathWorks, and is highly used in many science and engineering areas for numeric computing. Some important advantages of MATLAB include its ease of use and the large number of the available high-level functions for many applications. In this way, the motivation to develop an interface between MATLAB and FOQUS is intended to facilitate to FOQUS users the use of MATLAB models and its integration with other FOQUS supported modeling environments such as Aspen Plus and gPROMS, enabling the possibility to build highly complex cross-platform models which can then directly leverage FOQUS capabilities for advanced analysis.

Two different but equivalent approaches were implemented for interfacing MATLAB and FOQUS, which can be used depending on the user needs. These two approaches are described below:

Warning: The setup steps for the two approaches shown below were tested using MATLAB R2019b and Python 3.6, however they must work for other MATLAB and Python versions.

Option 1: MATLAB - FOQUS direct

This approach is best suited for MATLAB simulations that are not computationally intensive, although it can be used in those situations as well. This approach is fully integrated with FOQUS, and it is implemented in a simple way to enable running MATLAB simulations within FOQUS.

To be able to call MATLAB models from FOQUS through the FOQUS plugin implementation, it is required to setup properly the MATLAB engine API for Python, which is available for MATLAB-version R2014b or greater. MATLAB supports Python versions 2.7, 3.3, 3.4, 3.5, 3.6, 3.7, and 3.8. Further details regarding specific MATLAB and Python versions compatibilities are given [here](#).

To install the MATLAB engine package follows the steps below, which require compatible versions of Python and MATLAB already installed, and also a valid MATLAB license. The steps below assume that the Python distribution installed is Anaconda, but they also work for any other Python distribution.

1. Find out the MATLAB installation directory. To do this, just launch a new MATLAB session and type the instruction below:

```
matlabroot
```

2. Open an Anaconda command prompt. (Optional: activate the conda python environment if you are using a specific python environment for the installation).
3. Based on your operating system, move to the MATLAB installation folder, and then to the location that contains the python engine setup file. To do this, just type the instruction below:

```
cd %matlabroot%\extern\engines\python
```

Note: %matlabroot% is the MATLAB installation folder from step 1.

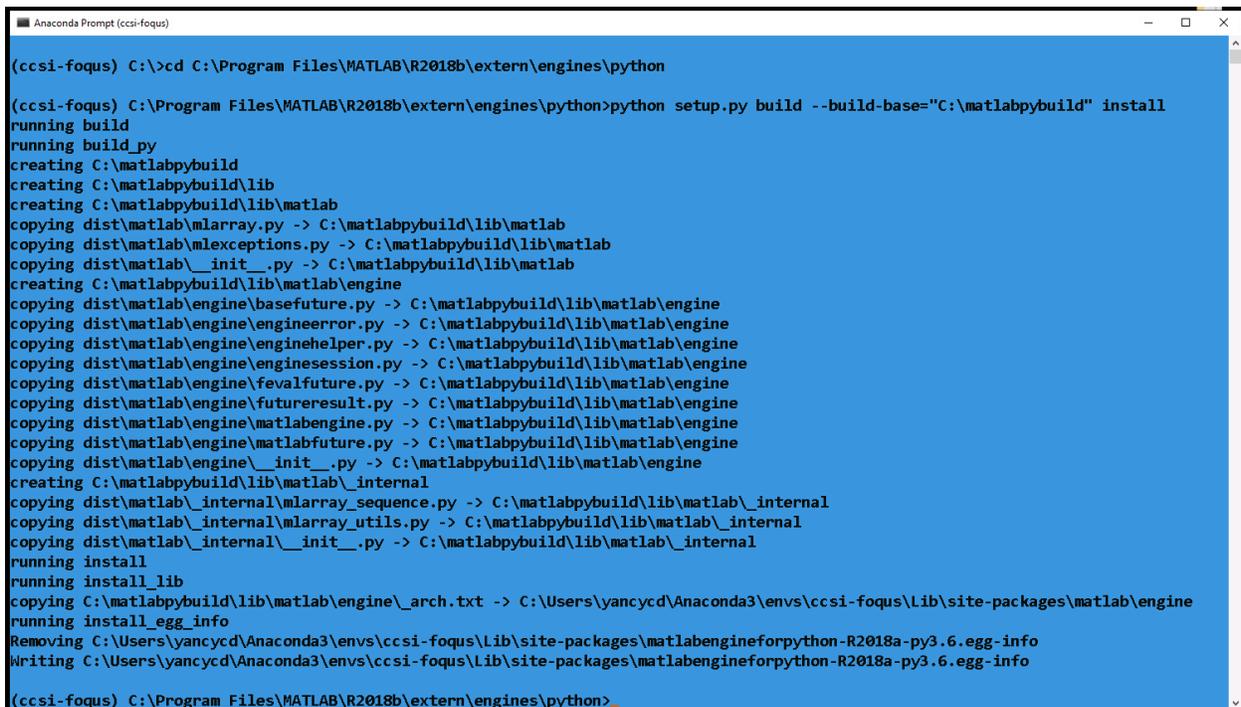
Now, if you list all files in this directory, you must see a setup.py file there.

4. Install the MATLAB engine package by typing the code below:

```
python setup.py build --build-base="C:\matlabpybuild" install
```

Note: C:\matlabpybuild is a folder to build the Python package. Users can use any folder that they have access to.

If the MATLAB engine package was installed correctly, a similar message to the Figure 1 must be seen on the terminal window.



```
(ccsi-foqus) C:\>cd C:\Program Files\MATLAB\R2018b\extern\engines\python

(ccsi-foqus) C:\Program Files\MATLAB\R2018b\extern\engines\python>python setup.py build --build-base="C:\matlabpybuild" install
running build
running build_py
creating C:\matlabpybuild
creating C:\matlabpybuild\lib
creating C:\matlabpybuild\lib\matlab
copying dist\matlab\mlarray.py -> C:\matlabpybuild\lib\matlab
copying dist\matlab\mlexceptions.py -> C:\matlabpybuild\lib\matlab
copying dist\matlab\__init__.py -> C:\matlabpybuild\lib\matlab
creating C:\matlabpybuild\lib\matlab\engine
copying dist\matlab\engine\basefuture.py -> C:\matlabpybuild\lib\matlab\engine
copying dist\matlab\engine\engineerror.py -> C:\matlabpybuild\lib\matlab\engine
copying dist\matlab\engine\enginehelper.py -> C:\matlabpybuild\lib\matlab\engine
copying dist\matlab\engine\enginesession.py -> C:\matlabpybuild\lib\matlab\engine
copying dist\matlab\engine\fevalfuture.py -> C:\matlabpybuild\lib\matlab\engine
copying dist\matlab\engine\futureresult.py -> C:\matlabpybuild\lib\matlab\engine
copying dist\matlab\engine\matlabengine.py -> C:\matlabpybuild\lib\matlab\engine
copying dist\matlab\engine\matlabfuture.py -> C:\matlabpybuild\lib\matlab\engine
copying dist\matlab\engine\__init__.py -> C:\matlabpybuild\lib\matlab\engine
creating C:\matlabpybuild\lib\matlab\_internal
copying dist\matlab\_internal\mlarray_sequence.py -> C:\matlabpybuild\lib\matlab\_internal
copying dist\matlab\_internal\mlarray_utils.py -> C:\matlabpybuild\lib\matlab\_internal
copying dist\matlab\_internal\__init__.py -> C:\matlabpybuild\lib\matlab\_internal
running install
running install_lib
copying C:\matlabpybuild\lib\matlab\engine\_arch.txt -> C:\Users\yancyd\Anaconda3\envs\ccsi-foqus\Lib\site-packages\matlab\engine
running install_egg_info
Removing C:\Users\yancyd\Anaconda3\envs\ccsi-foqus\Lib\site-packages\matlabengineforpython-R2018a-py3.6.egg-info
Writing C:\Users\yancyd\Anaconda3\envs\ccsi-foqus\Lib\site-packages\matlabengineforpython-R2018a-py3.6.egg-info

(ccsi-foqus) C:\Program Files\MATLAB\R2018b\extern\engines\python>
```

Fig. 1: Figure 1 - Terminal window message after installing the MATLAB engine package

Now, to run MATLAB models within FOQUS follow the steps below:

1. Create a node simulation in the FOQUS flowsheet editor and define all input and output variables of the model.
2. Create a MATLAB function calling the model.

3. Call FOQUS plugin named “matlab_fs” to start a new MATLAB session. This can be done in the Model section at node editor. In “Type” option choose “plugin”, and in “Model” option choose “matlab_fs”.
4. Connect to the current MATLAB session from the node script.
5. Create a MATLAB array object in the FOQUS node script containing the input parameters for the MATLAB model.
6. Call the MATLAB function/model.
7. Retrieve the outputs from the MATLAB function to FOQUS output variables.
8. Terminate MATLAB session.

Further details on how to use this option to interface MATLAB-FOQUS are given in the example presented in the *tutorial 1*.

Option 2: MATLAB script implementation

This approach is best suited for MATLAB simulations that are computationally intensive, and FOQUS is used for data analysis and surrogate modeling.

In this option, the MATLAB-FOQUS interface runs MATLAB models directly in the MATLAB environment, but making the results/outputs fully compatible with FOQUS modules. This is automatically achieved through a MATLAB script `fokus_matlab_script.m` provided with the FOQUS distribution, which can be executed directly in MATLAB. To use the script, it is necessary to define the inputs for MATLAB models in the same order as were defined in the FOQUS flowsheet.

The MATLAB script takes three inputs: 1) the MATLAB function containing the model, 2) the name of the PSUADE file containing the samples space for the model, which needs to be created previously in the uncertainty module in FOQUS, 3) the path where the MATLAB function and PSUADE file are located.

The MATLAB script uses some functions available in FOQUS base code to handle *PSUADE full file format* and sample data objects, and these functions are written in Python. For this reason, before using the script, it is necessary to configure MATLAB to execute Python modules. The steps for this configuration are given below:

1. Find out where Python executable is located. To do this, open an Anaconda command prompt or a Terminal and type the code below:

```
python -c "import sys; print(sys.executable)"
```

2. Open a new MATLAB session and type the code below:

```
pyenv('Version', '%pythonroot%python.exe')
```

Note: `%pythonroot%` is the Python executable folder found in step 1. You can also verify if the Python config was stored in MATLAB by typing again `pyenv`, and then you must see the previous message again.

Warning: `pyenv` was first introduced in MATLAB R2019b. In older MATLAB versions, you need to use `pyversion`, as shown below:

```
pyversion('%pythonroot%python.exe')
```

3. Now, type the code line below:

```
py.numpy.arange(1)
```

Note: If you do not get errors, then the Python configuration is ready and skip the following steps. If you got this, or any similar error: `Unable to resolve the name py.numpy.arange`, then you need to verify that the folder containing the Python binary files is included in the system environment variables, for this, go to step 4.

4. In MATLAB, type the code below to see all folders that are added to the system path:

```
getenv('PATH')
```

Note: Check if `%pythonroot%\Library\bin` is already in the path, if not, follows step 5.

5. In MATLAB, type the code below:

```
setenv('PATH', [%pythonroot%\Library\bin', pathsep, getenv('PATH')])
```

Note: Replace `%pythonroot%` with the Python executable folder found in step 1. You can also add manually the folder containing the Python binary files to the system environment variables, but this will depend on the specific operating system.

6. Type again the code below:

```
py.numpy.arange(1)
```

Note: This time everything should work fine without errors.

After completing the configuration part to execute Python modules within MATLAB, the general steps to interfacing MATLAB and FOQUS are as follows:

1. Create a node simulation in the FOQUS flowsheet editor and define all input and output variables of the model.
2. Create a new ensemble for the sample space using the uncertainty quantification module in FOQUS.
3. Export the UQ Ensemble to *PSUADE full file format*.
4. Create a MATLAB function calling the model (it is necessary to define the inputs for the MATLAB function in the same order as were defined in the FOQUS flowsheet in step 1).
5. Execute the MATLAB script `foqus_matlab_script.m` provided with FOQUS calling the MATLAB model function and the PSUADE file.
6. A new csv file `outputs.csv` fully compatible with FOQUS and containing the results from MATLAB simulations for the entire sample space is created.
7. Now, the `outputs.csv` file can be imported in FOQUS to use the different FOQUS capabilities for subsequent analysis.

Further details on how to use this option to interface MATLAB-FOQUS are given in the example presented in the *tutorial 2*.

13.1.2 MATLAB-FOQUS interface - tutorials

Problem Statement: Steady-State Continuous Stirred Tank Reactor (CSTR)

This example solves a non-linear system of equations which describes a mathematical model for a Continuous Stirred Tank Reactor (CSTR) at steady state. The CSTR is cooled with a cooling coil, and a simple exothermic reaction takes place inside the reactor (see Figure 1). Model main assumptions: 1) the reactant is perfectly mixed, and 2) the volume, heat capacities and densities are constants. Further details regarding the model are given in Vojtesek and Dostal, 2011.

Source: Jiri Vojtesek and Petr Dostal. Use of MATLAB Environment for Simulation and Control of CSTR. International Journal of Mathematics and Computers in Simulation, 6(5), 2011.

Fig. 2: Figure 1 - Representation of a CSTR with exothermic reaction

The CSTR model in steady-state is represented by the following non-linear system of equations, which was obtained from mass and energy balances of the reactant and cooling.

$$a_1 \cdot (T_0 - T) + a_2 \cdot k_1 \cdot c_A + a_3 \cdot q_c \cdot \left(1 - e^{\frac{a_4}{q_c}}\right) \cdot (T_{c0} - T) = 0$$

$$a_1 \cdot (c_{A0} - c_A) - k_1 \cdot c_A = 0$$

$$k_1 = k_0 \cdot e^{\frac{-E}{R \cdot T}}$$

$$q_0 = q$$

$$q_{c0} = q_c$$

where a_{1-4} are constants calculated as follows:

$$a_1 = \frac{q}{V}; \quad a_2 = \frac{-\Delta H}{\rho \cdot c_p}; \quad a_3 = \frac{\rho_c \cdot c_{pc}}{\rho \cdot c_p \cdot V}; \quad a_4 = \frac{-h_a}{\rho_c \cdot c_{pc}}$$

The fixed parameters of the system are given below.

Parameter	Symbol [Unit]	Value
Reactor's volume	V [l]	100
Reaction rate constant	k_0 [min ⁻¹]	7.2e10
Activation energy divided by R	E/R [K]	1e4
Reactant's feed temperature	T_0 [K]	350
Inlet coolant temperature	T_{c0} [K]	350
Reaction heat	ΔH [cal · mol ⁻¹]	-2e5
Specific heat of the reactant	c_p [cal · g ⁻¹ · K ⁻¹]	1
Specific heat of the cooling	c_{pc} [cal · g ⁻¹ · K ⁻¹]	1
Density of the reactant	ρ [g · l ⁻¹]	1e3
Density of the cooling	ρ_c [g · l ⁻¹]	1e3
Feed concentration	c_{A0} [mol · l ⁻¹]	1
Heat transfer coefficient	h_a [cal · min ⁻¹ · K ⁻¹]	7e5
Volumetric flow rate of reactant	q_0 [l · min ⁻¹]	100
Volumetric flow rate of cooling	q_{c0} [l · min ⁻¹]	80

The variables in the system of equations are described below:

Variable	Symbol [Unit]
Final reactant concentration	c_A [mol · l ⁻¹]
Volumetric flow rate of products	q [l · min ⁻¹]
Volumetric flow rate of cooling	q_c [l · min ⁻¹]
Product temperature	T [K]
Reaction rate	k_1 [min ⁻¹]
Conversion	X_A [-]

The conversion of reactant A is defined as:

$$X_A = \frac{c_{A0} - c_A}{c_{A0}}$$

Tutorial 1: MATLAB - FOQUS direct

Step 1: Flowsheet Setup - create a node simulation in the FOQUS flowsheet editor, and name it “CSTR_Steady_State”.

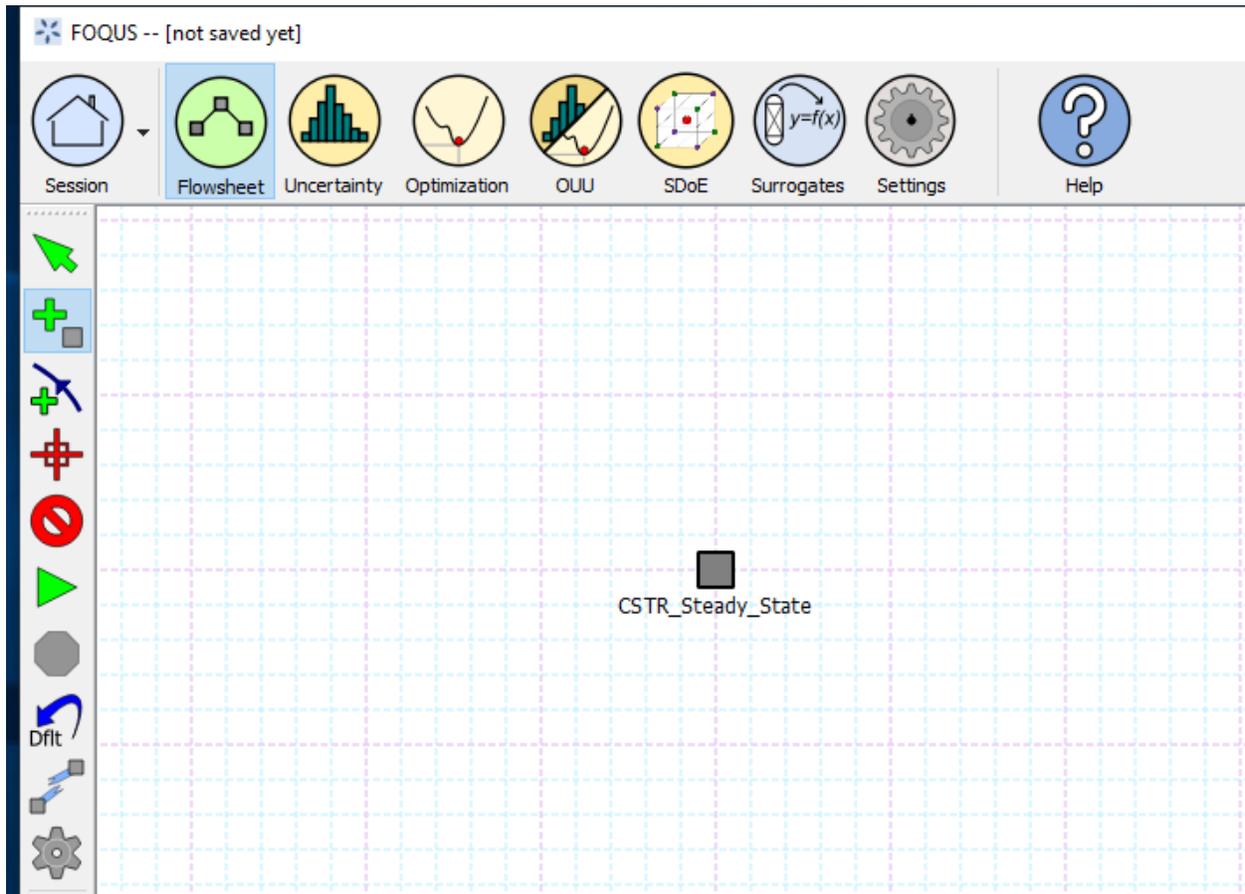


Fig. 3: Figure 2 - Flowsheet Setup

Step 2: Define all input and output variables of the model as described in Figures 3 and 4.

Step 3: Create a MATLAB function solving the non-linear system of equations presented above. The MATLAB

Node Edit

Variables | Position | Node Script

Name: Visible

Error Status

Code:

Message:

Model

Type: Model:

Input Variables

	Name	Value	Unit	Type	Default	Min	Max	Description	Tag
1	CA0	1.0	mol/l	float	1.0	0.0	10.0	Feed concentration	[]
2	cp	1.0	cal/g/K	float	1.0	0.1	10.0	Specific heat of the reactant	[]
3	cpc	1.0	cal/g/K	float	1.0	0.1	10.0	Specific heat of the cooling	[]
4	delH_neg	200000.0	cal/mol	float	200000.0	10000.0	1000000.0	Reaction heat	[]
5	E_R	10000.0	K	float	10000.0	1000.0	100000.0	Activation energy to R	[]
6	ha	700000.0	cal/min/K	float	700000.0	10000.0	1000000.0	Heat transfer coefficient	[]
7	k0	72000000000.0	1/min	float	72000000000.0	1000000000.0	100000000000.0	Reaction rate constant	[]
8	q0	100.0	l/min	float	100.0	10.0	500.0	Volumetric flow rate of reactant	[]
9	qc0	80.0	l/min	float	80.0	10.0	500.0	Volumetric flow rate of cooling	[]
10	rho	1000.0	g/l	float	1000.0	700.0	1500.0	Density of the reactant	[]
11	rho_c	1000.0	g/l	float	1000.0	700.0	1500.0	Density of the cooling	[]
12	T0	350.0	K	float	350.0	298.0	500.0	Reactant's feed temperature	[]
13	TC0	350.0	K	float	350.0	298.0	500.0	Inlet coolant temperature	[]
14	V	100.0	l	float	100.0	50.0	150.0	Reactor's volume	[]

Legend:

Output Variables

Settings

Fig. 4: Figure 3 - Input Variables

Node Edit

Apply
 Revert
 Run (this node only for testing)
 Stop Run

Variables Position Node Script

Name: CSTR_Steady_State Visible

Error Status

Code: 0

Message: Finished Normally

Model

Type: None Model:

Input Variables

Output Variables

+ - Tags

	Name	Value	Unit	Type	Description	Tags
1	a1	0	1/min	float	constant 1	[]
2	CA	0	mol/l	float	Final concentration of reactant A	[]
3	k1	0	1/min	float	Reaction rate	[]
4	q	0	l/min	float	Volumetric flow rate of reactant	[]
5	qc	0	l/min	float	Volumetric flow rate of cooling	[]
6	T	0	K	float	Product's temperature	[]
7	XA	0	-	float	Conversion of reactant A	[]

Settings

Fig. 5: Figure 4 - Output Variables

function file (along with the FOQUS file for this example) can be found in the folder:
examples/tutorial_files/MATLAB-FOQUS/Tutorial_1

Note: The **examples/** directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

Step 4: Load FOQUS plugin named “matlab_fs” in the simulation node as shown Figure 5. In the node editor, under “Type” option, choose “plugin”, and under “Model” option choose “matlab_fs”.

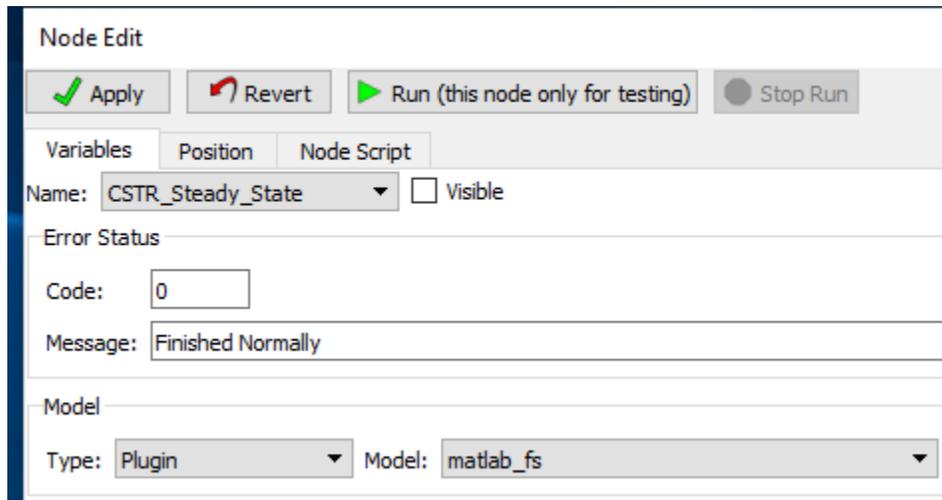


Fig. 6: Figure 5 - FOQUS plugin for MATLAB-FOQUS interface

Step 5: In the Node Script tab write the code as shown in Figure 6.

Note: The code shown in Figure 6 is intended to: 1) connect to the current MATLAB session, 2) create a MATLAB array object containing the input parameters for the MATLAB model, 3) Call the MATLAB function/model, and 4) Retrieve the outputs from the MATLAB function to FOQUS output variables.

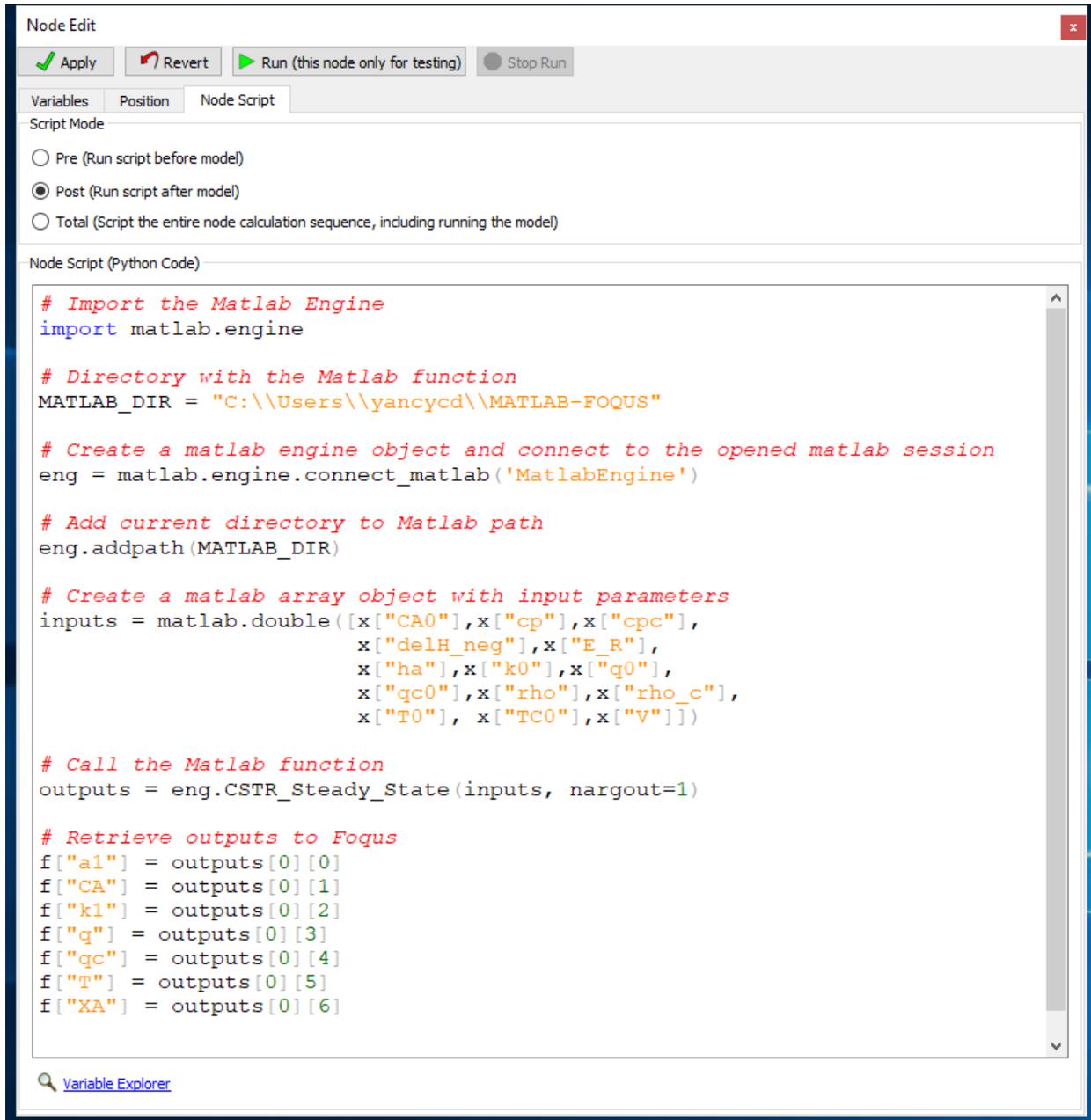
- The code is below:

```

1  # Import the Matlab Engine
2  import matlab.engine
3
4  # Directory with the Matlab function
5  MATLAB_DIR = "C:\\Users\\yancyd\\MATLAB-FOQUS"
6
7  # Create a matlab engine object and connect to the opened matlab session
8  eng = matlab.engine.connect_matlab('MatlabEngine')
9
10 # Add current directory to Matlab path
11 eng.addpath(MATLAB_DIR)
12
13 # Create a matlab array object with input parameters
14 inputs = matlab.double([x["CA0"], x["cp"], x["cpc"],
15                        x["delH_neg"], x["E_R"],
16                        x["ha"], x["k0"], x["q0"],

```

(continues on next page)



The screenshot shows a 'Node Edit' window with a toolbar at the top containing 'Apply', 'Revert', 'Run (this node only for testing)', and 'Stop Run'. Below the toolbar are tabs for 'Variables', 'Position', and 'Node Script'. The 'Node Script' tab is active, showing a 'Script Mode' section with radio buttons for 'Pre (Run script before model)', 'Post (Run script after model)' (which is selected), and 'Total (Script the entire node calculation sequence, including running the model)'. The main area contains Python code for connecting to the Matlab engine and running a CSTR model. The code includes comments and function calls for setting the directory, connecting to the engine, adding the current directory to the path, creating an array of input parameters, calling the 'CSTR_Steady_State' function, and retrieving the outputs to the Foqus environment.

```
# Import the Matlab Engine
import matlab.engine

# Directory with the Matlab function
MATLAB_DIR = "C:\\Users\\yancyd\\MATLAB-FOQUS"

# Create a matlab engine object and connect to the opened matlab session
eng = matlab.engine.connect_matlab('MatlabEngine')

# Add current directory to Matlab path
eng.addpath(MATLAB_DIR)

# Create a matlab array object with input parameters
inputs = matlab.double([x["CA0"], x["cp"], x["cpc"],
                        x["delH_neg"], x["E_R"],
                        x["ha"], x["k0"], x["q0"],
                        x["qc0"], x["rho"], x["rho_c"],
                        x["T0"], x["TC0"], x["V"]])

# Call the Matlab function
outputs = eng.CSTR_Steady_State(inputs, nargout=1)

# Retrieve outputs to Foqus
f["a1"] = outputs[0][0]
f["CA"] = outputs[0][1]
f["k1"] = outputs[0][2]
f["q"] = outputs[0][3]
f["qc"] = outputs[0][4]
f["T"] = outputs[0][5]
f["XA"] = outputs[0][6]
```

Variable Explorer

Fig. 7: Figure 6 - Node Script Code

(continued from previous page)

```

17     x["qc0"], x["rho"], x["rho_c"],
18     x["T0"], x["TC0"], x["V"]])
19
20 # Call the Matlab function
21 outputs = eng.CSTR_Steady_State(inputs, nargout=1)
22
23 # Retrieve outputs to FOCUS
24 f["a1"] = outputs[0][0]
25 f["CA"] = outputs[0][1]
26 f["k1"] = outputs[0][2]
27 f["q"] = outputs[0][3]
28 f["qc"] = outputs[0][4]
29 f["T"] = outputs[0][5]
30 f["XA"] = outputs[0][6]

```

Step 6: Run the node simulation to test if the simulation is working properly.

Step 7: Under the uncertainty tab in FOQUS, select Add New option to generate a new simulation ensemble. Select Use Flowsheet option. Fix all variables except the volume, which will be a variable with bounds 50-150 l. Select Latin Hypercube sampling method with 100 samples, and then generate the samples. Figure 7 represents the simulation ensemble generation.

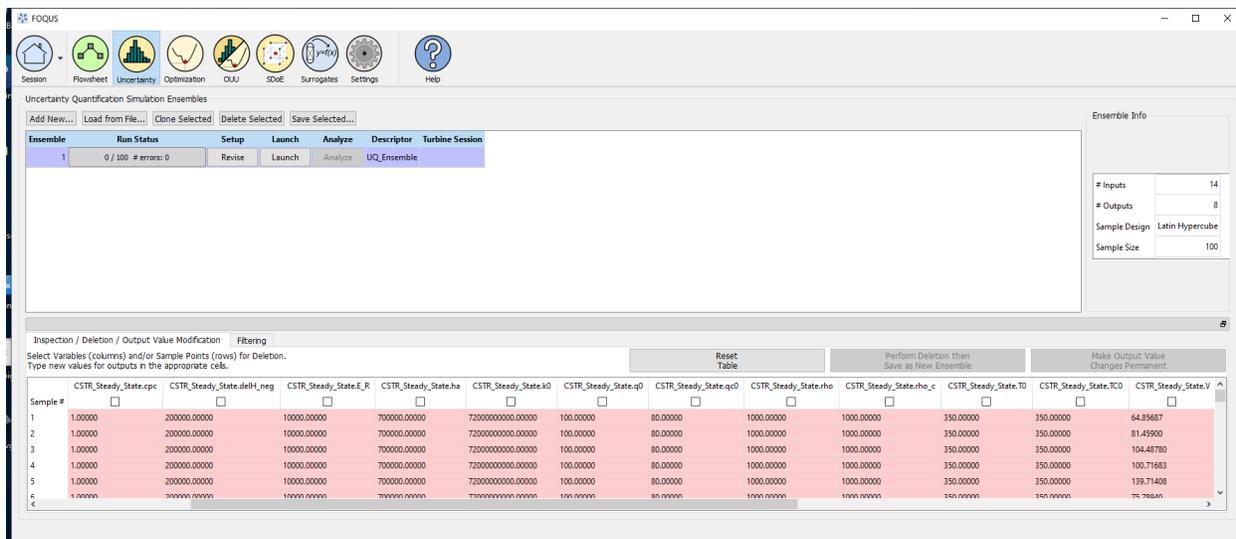


Fig. 8: Figure 7 - Ensemble Generation

Step 8: Launch the simulations. Figure 8 represents the simulation results.

Now, plotting the conversion vs the reactor's volume, a similar figure to Figure 9 must be obtained.

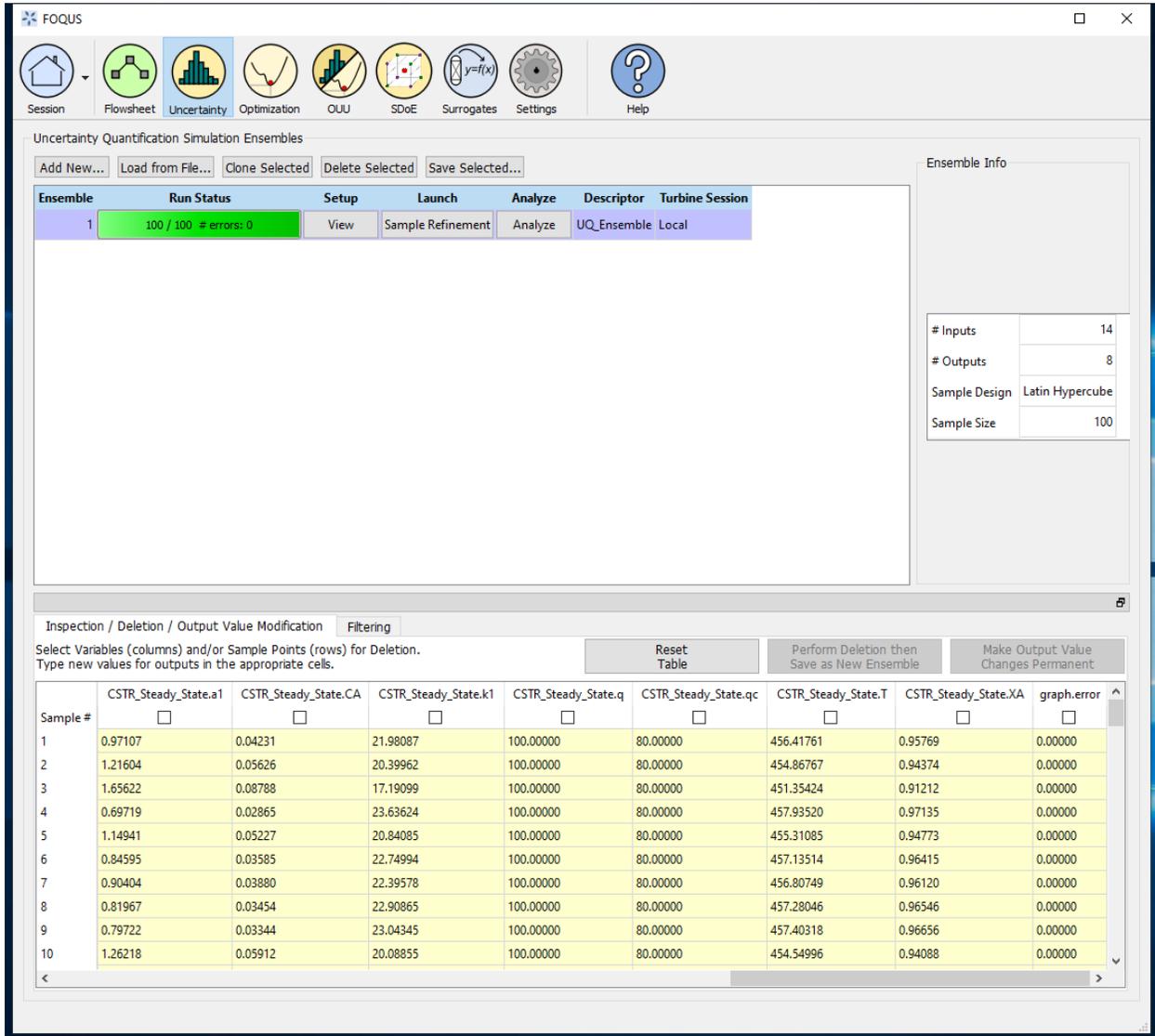


Fig. 9: Figure 8 - Ensemble Results

Fig. 10: Figure 9 - Conversion of Reactant A vs Reactor's Volume

Tutorial 2: MATLAB script implementation

Step 1: Follow steps 1-3 from the Tutorial 1: MATLAB - FOQUS direct section. Users need to take care when defining the MATLAB function for the model in step 3 as it is necessary to define the MATLAB function inputs in the same order as were defined in the FOQUS flowsheet.

Step 2: Follow step 6 from the Tutorial 1: MATLAB - FOQUS direct section to generate a new simulation ensemble.

Step 3: Select the new generated UQ_Ensemble and click on Save Selected to save the ensemble as a PSUADE file. Choose a folder to save the file and name it as data.dat.

Step 4: Create a new MATLAB script to call the matlab_foqus_script.m file (which is distributed with FOQUS and can be found in examples/tutorial_files/MATLAB-FOQUS/Tutorial_2), and pass to it the MATLAB function containing the model. Below is an example of the code that needs to be executed. In examples/tutorial_files/MATLAB-FOQUS/Tutorial_2 you can find a MATLAB file name example_2_matlab_foqus.m with the code, and you can simply execute it:

```

1 % This is the path where the MATLAB model, the "matlab_foqus_script.m" file,
  → and the PSUADE file "data.dat" are located
2 path = "C:\Users\yancyd\MATLAB-FOQUS\";
3 % This is the PSUADE file name
4 PsuadFileName = 'data.dat';
5 % This is the MATLAB function name that contains the model
6 MatlabFunctionName = @(x) CSTR_Steady_State(x);
7 % Call the "matlab_foqus_script.m" file
8 matlab_foqus_script(MatlabFunctionName, PsuadFileName, path)

```

Note: After executing the code above, a new outputs.csv is created with the sample results from MATLAB. This is a file fully compatible with FOQUS.

Step 5: Under the uncertainty module, click on Load from File. Then choose .csv format file option and select the outputs.csv file created in the previous step. A new window will ask you the number of inputs that contain the outputs.csv file (see Figure 10), for this example is 14.

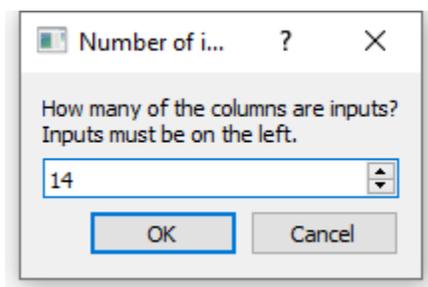


Fig. 11: Figure 10 - Number of Inputs in the Outputs File

Step 6: Now, you have a new ensemble named “output.csv” with all input and outputs variables (see Figure 11), which can be used for other advanced analysis in the uncertainty module or any other FOQUS module.

If you plot the conversion vs the reactor’s volume, you should get the Figure 9.

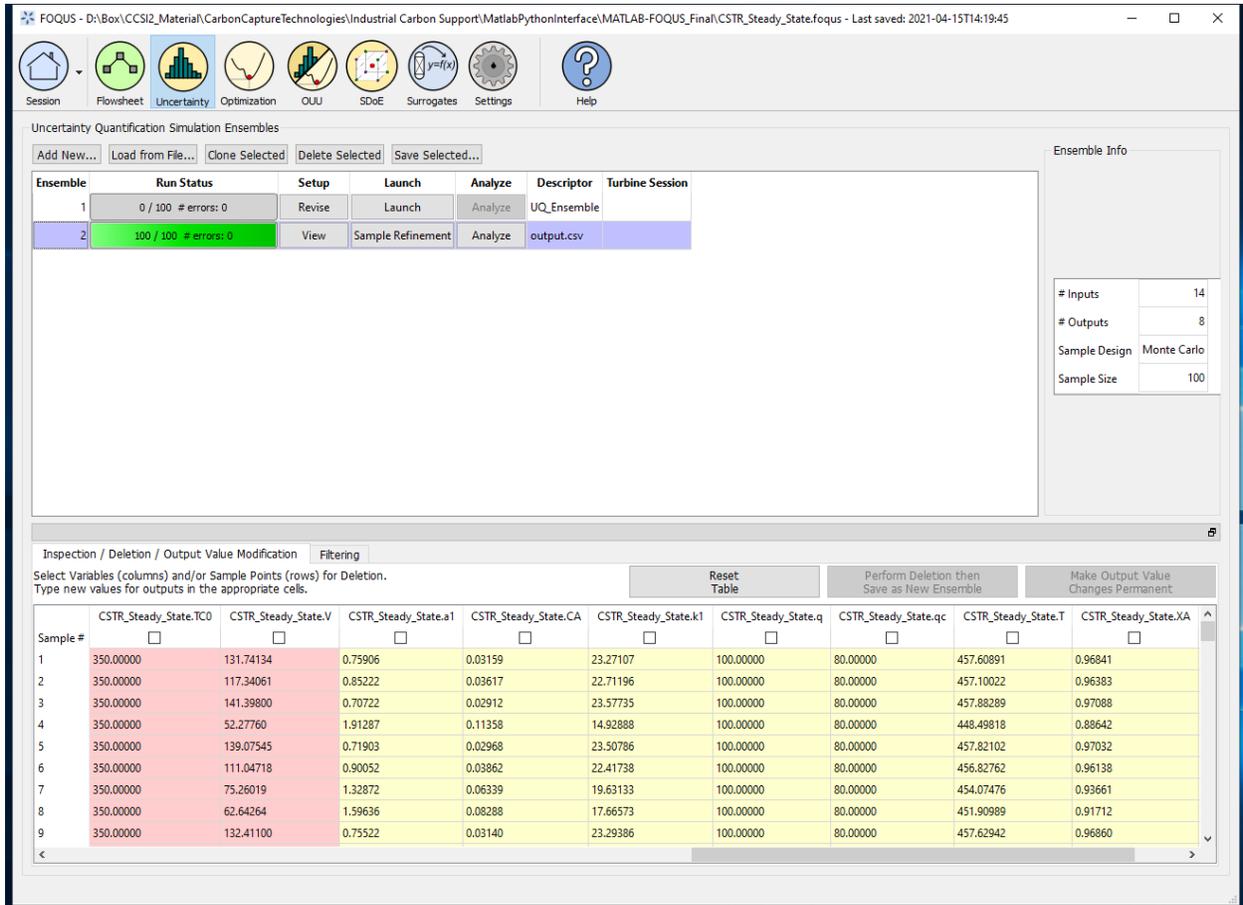


Fig. 12: Figure 11 - New Ensemble with MATLAB Results

SIMULATION STANDARD INTERFACE (SIMSINTER)

14.1 Contents

14.1.1 SimSinter

Configuration

SimSinter is the standard interface library that FOQUS and Turbine use to drive the target simulation software.

SimSinter currently supports:

- AspenPlus (versions 8, 9, and 10)
- Aspen Custom Modeler (ACM) (versions 8, 9, and 10)
- gPROMS
- Microsoft Excel

SimSinter is used to: (1) open the simulator, (2) initialize the simulation, (3) set variables in the simulation, (4) run the simulation, and (5) get resulting output variables from the simulation.

To drive a particular simulation, SimSinter must be told which input variables to set and which output variables to read when the simulation is finished (there are generally far too many variables in a simulation to set and read them all). Each simulation must have a “Sinter Config File” which records this information. FOQUS keeps the simulation file and the “Sinter Config File” together and sends them to the Turbine gateway when a simulation run is requested.

The configuration is simplified by a GUI included with the SimSinter distribution called, “SinterConfigGUI.”

FOQUS can launch the SinterConfigGUI on simulations that have not been configured. To run the “SinterConfigGUI” the user must have:

1. SimSinter distribution installed. SimSinter is installed by the FOQUS bundle installer.
2. The simulation file the user wants to configure. For example, if the user has an Aspen Custom Modeler simulation called BFB.acmf, that file must be on the user’s computer, and the user should know its location.
3. The application used to execute the simulation file. For example, if the user wants to configure an Aspen Custom Modeler simulation called BFB.acmf, Aspen Custom Modeler must be installed on the user’s machine.

The rest of this section details two step-by-step tutorials on configuring a simulation with “SinterConfigGUI.” The first simulation is an Aspen Custom Modeler simulation and the second, Aspen Plus. Please also see the D-RM Builder tutorials for configuring dynamic ACM models. For more details on SimSinter or a tutorial on how to configure a Microsoft Excel file, please see the “SimSinter Technical Manual,” which is included in the FOQUS distribution. The default location is at C:\Program Files (x86)\foqusfoqusdoc. It is also available on the CCSI website.

14.1.2 Tutorial

Tutorial 1: Aspen Custom Modeler (ACM) Configuration

The files (both the ACM and the JSON files) for this tutorial are located in:
 examples/test_files/Optimization/Model_Files

Note: The **examples/** directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

1. The “SinterConfigGUI” can be launched from FOQUS, via the **Create/Edit** button found in **Session** → **Add/Update Model to Turbine** or “SinterConfigGUI” may be run on its own by selecting **SimSinter** → **SinterConfigGUI** from the Windows Start menu.
2. The splash window displays, as shown in Figure *SinterConfigGUI Splash Screen*. The user may click the splash screen to proceed, or wait ten seconds for it to close automatically.



Fig. 1: SinterConfigGUI Splash Screen

3. The SinterConfigGUI Open Simulation window displays (Figure *SinterConfigGUI Open Simulation Window*). If “SinterConfigGUI” was opened from FOQUS, the filename text box already contains the correct file. To proceed immediately click **Open File and Configure Variables** or click **Browse** to search for the file. For this tutorial, the ACM model (BFBv6.2.acmf) for a bubbling fluidized bed adsorber (located in the examples/test_files/Optimization/Model_Files folder) is selected. Once the file is selected, click **Open File and Configure Variables**. The user can open a fresh ACM simulation (.acmf file) or an existing SimSinter configuration file. For this example, open a fresh simulation.

Note: Opening the simulation may take a few minutes depending on how quickly Aspen Custom Modeler can be opened.

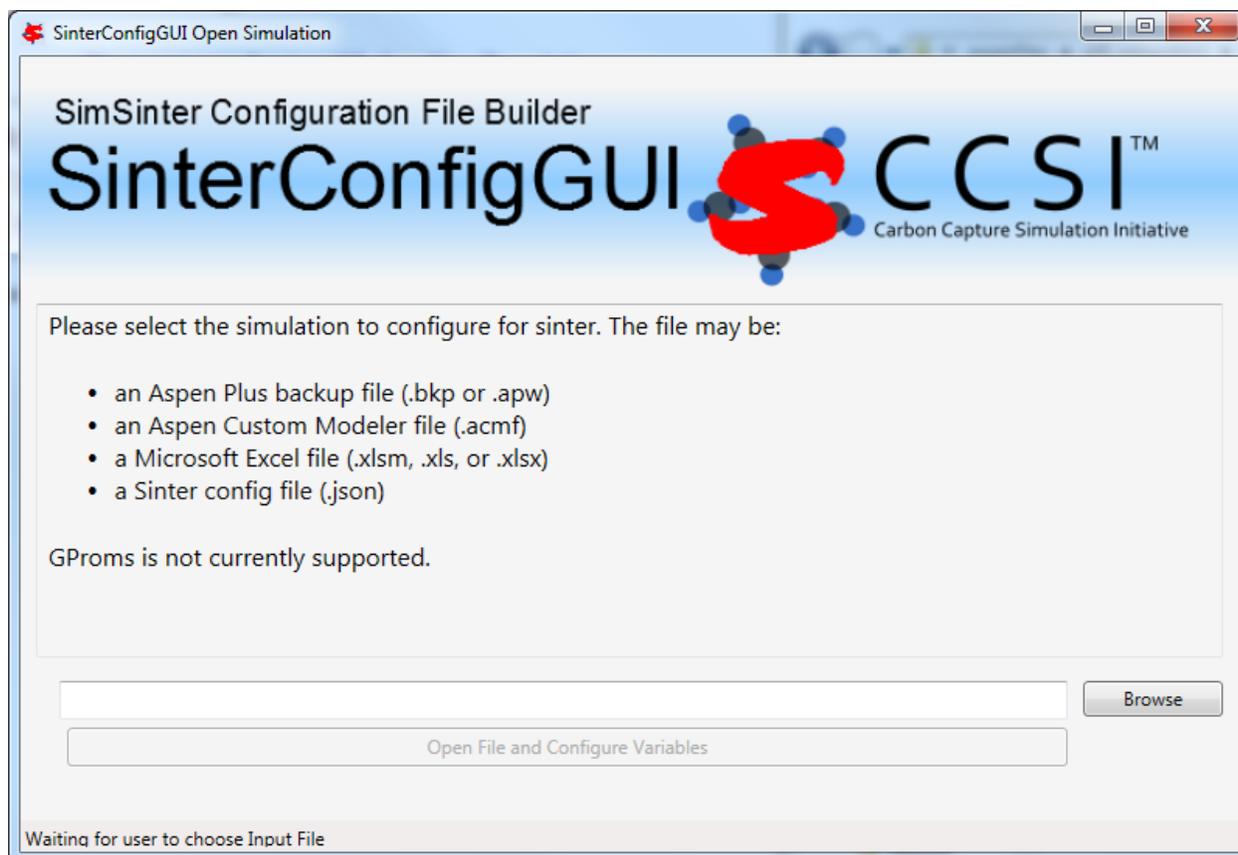


Fig. 2: SinterConfigGUI Open Simulation Window

4. Aspen Custom Modeler starts in the background. This is so the user can observe things about the simulation while working on the configuration file.
5. The SinterConfigGUI Simulation Meta-Data window displays. (Figure *SinterConfigGUI Simulation Meta-Data Page Save Name Text Box*). The first and most important piece of metadata is **SimSinter Save Location** at the top of the window. This is where the sinter configuration file is saved. The system attempts to locate a reasonable file location and file name; however, the user must confirm the correct file location, since it automatically overwrites whatever file name currently exists.
6. Continue to complete the remaining fields and then click **Next** (Figure *SinterConfigGUI Simulation Meta-Data Page with Data Completed*).
7. In the **SinterConfigGUI Variable Configuration Page**, (Figure *SinterConfigGUI Variable Configuration Page before Input*) notice that the ACM **Selected Input Variables: TimeSeries, Snapshot, RunMode, printlevel and homotopy** are already included in the input variables. **TimeSeries** and **Snapshot** are for dynamic simulations. **RunMode** can be either “Steady State” or “Dynamic”. The Dynamic mode requires a dynamic ACM model. For this simulation, the RunMode is Steady State. The **homotopy** variable can be set to “1” so that homotopy is on by default. Notice that the **Dynamic** column (the first column) in each row contains a checkbox, enabling the user to select if the input variable in the row is a dynamic variable. Also notice that a **Variable Type** search box is on the left. This search is exactly the same as Variable Find on the Tools menu in Aspen Custom Modeler. Please refer to the ACM documentation for details on search patterns.

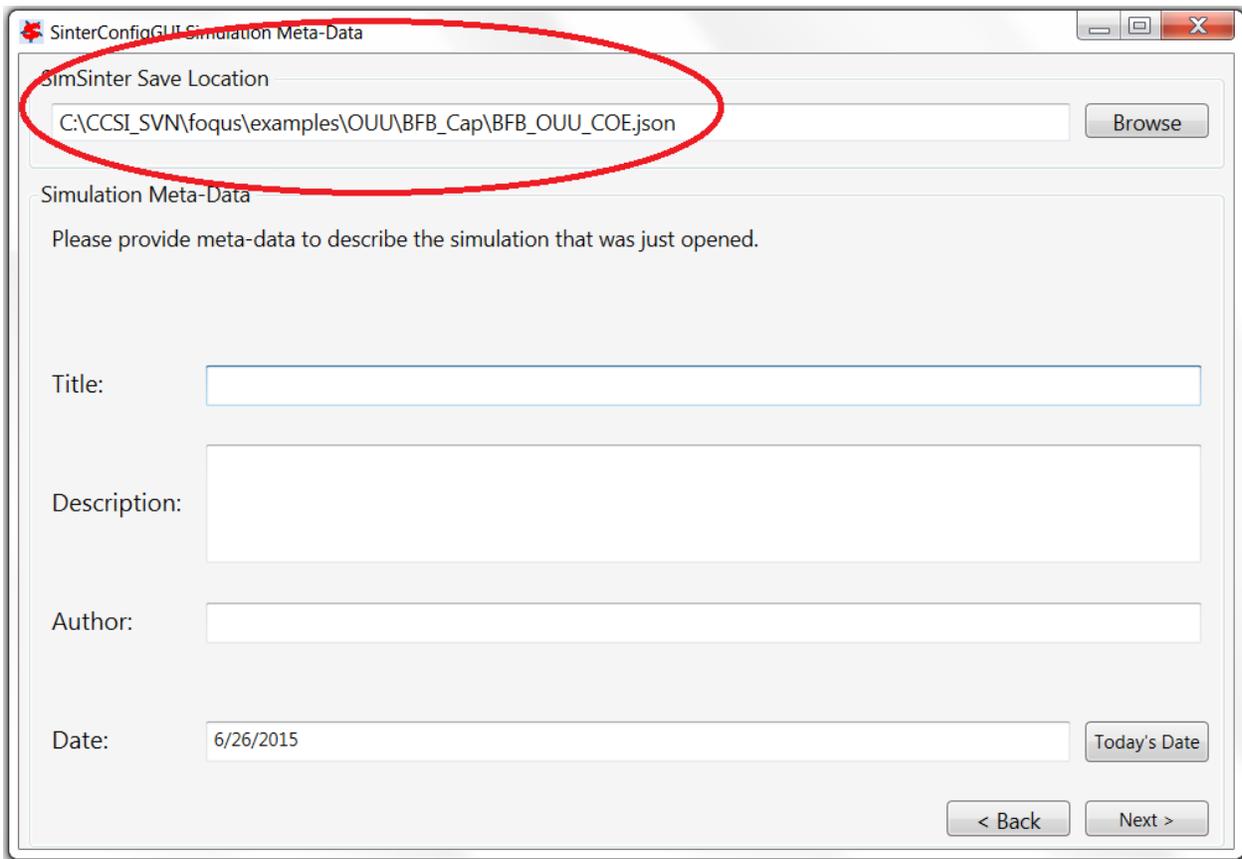


Fig. 3: SinterConfigGUI Simulation Meta-Data Page Save Name Text Box

SinterConfigGUI Simulation Meta-Data

SimSinter Save Location

C:\CCSI_SVN\foqus\examples\OUU\BFB_Cap\BFB_OUU_COE.json

Simulation Meta-Data

Please provide meta-data to describe the simulation that was just opened.

Title:

Description:

Author:

Date:

Fig. 4: SinterConfigGUI Simulation Meta-Data Page with Data Completed

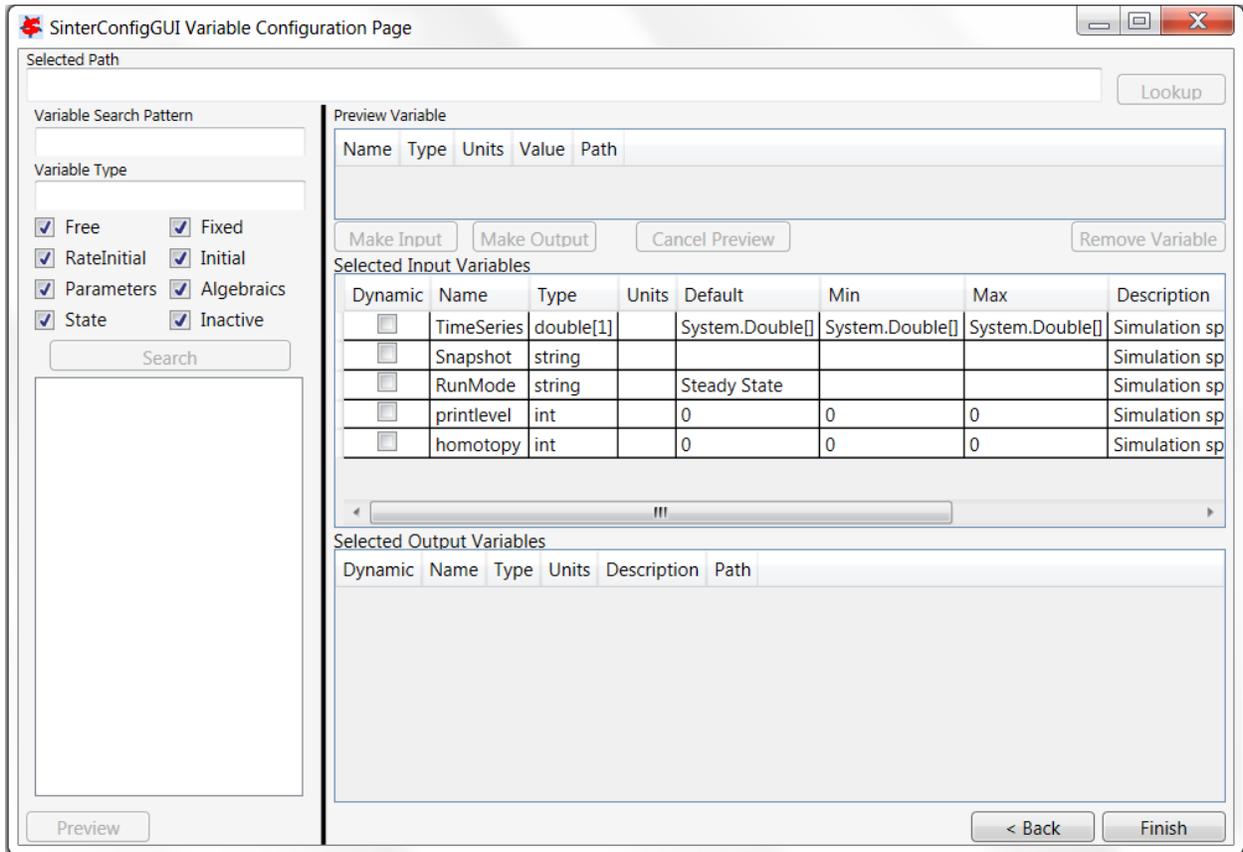


Fig. 5: SinterConfigGUI Variable Configuration Page before Input

8. A search for everything in the “BFBAdsT” block has been selected. The following **Search in Progress** dialog is displayed (Figure *Search in Progress Bar Page*). Sometimes large searches take a while.

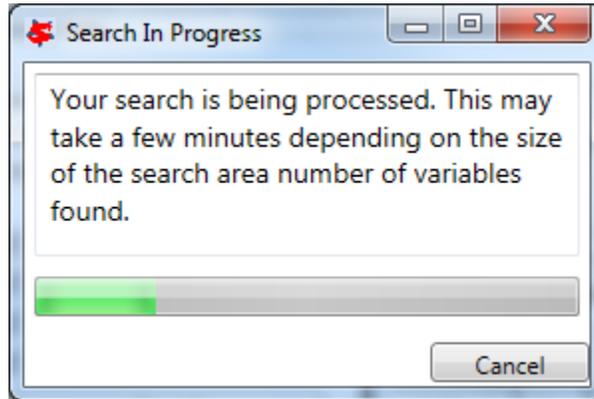


Fig. 6: Search in Progress Bar Page

9. First, select the “BFBadsT.A1” scalar variable in the **Selected Path** field (Figure *SinterConfigGUI Variable Configuration Page BFBadsT.A1 Selected*).
10. If the user double-clicks, presses Enter, or clicks **Preview** or **Lookup**, information displays in the **Preview Variable** section (Figure *SinterConfigGUI Variable Configuration Page BFBadsT.A1 Preview*). Here, the user can verify the variable choices.
11. “BFBadsT.A1” is the correct variable; therefore, click **Make Input**. Information displays in the **Selected Input Variables** section (Figure *SinterConfigGUI Variable Configuration Page BFBadsT.A1 Made Input*).
12. Change the variable name from “BFBadsT.A1” to something more descriptive (e.g., “WaterA”). Set **Name**, **Description** and **Min/Max** as shown in Figure *SinterConfigGUI Variable Configuration Page BFBadsT.A1 Change Name*.
13. One input variable is now displayed (Figure *SinterConfigGUI Variable Configuration Page Vector Preview*). At least one output variable is required. In this example, the vector of calculated bubble sizes is wanted. Scroll down under **Search** and select “BFBadsT.db.Value,” “BFBadsT.db.Value(0),” “BFBadsT.db.Value(1),” etc. If a name with a number in parenthesis at the end is selected, it is a specific entry in the vector. If a basic name is selected (“BFBadsT.db.Value”), the entire vector is displayed. Select the whole vector and click **Preview**.
14. Click **Make Output** if the variable the user wants is selected. Notice that this variable has a unit “m” (Figure *SinterConfigGUI Variable Configuration Page Vector As Output*).
15. Change the **Name** of the variable to “Diameter.” Bubble size is measured in meters; however, meters should be converted to millimeters (mm). Now, the output from the simulation should present bubble diameter in mm (Figure *SinterConfigGUI Variable Configuration Page Output Change Units*). Internal to the simulation, the unit remains “m.”
16. To add a single item in a vector, select “BFBadsT.Ar.Value(1)” and click **Make Input** (See Figure *SinterConfigGUI Variable Configuration Page Removal Demo*). To remove item that was just added, select it and click **Remove Variable**.
17. Select the correct variable vector “BFBadsT.Ar.Value” and make it an input (Figure *SinterConfigGUI Variable Configuration Page Read Input*). Notice that a **Default** or **Min/Max** cannot be set in the GUI for a vector. The correct defaults (from the simulation) are set automatically. To change the **Min/Max** values, the user must edit the JSON file in a text editor.
18. Click **Next** to display the SinterConfigGUI Vector Default Initialization window as shown in Figure *SinterConfigGUI Vector Default Initialization Input Page*. Since the input variable “Value” is a vector, its default values can be modified in the window. In this case there is no need to change the values.

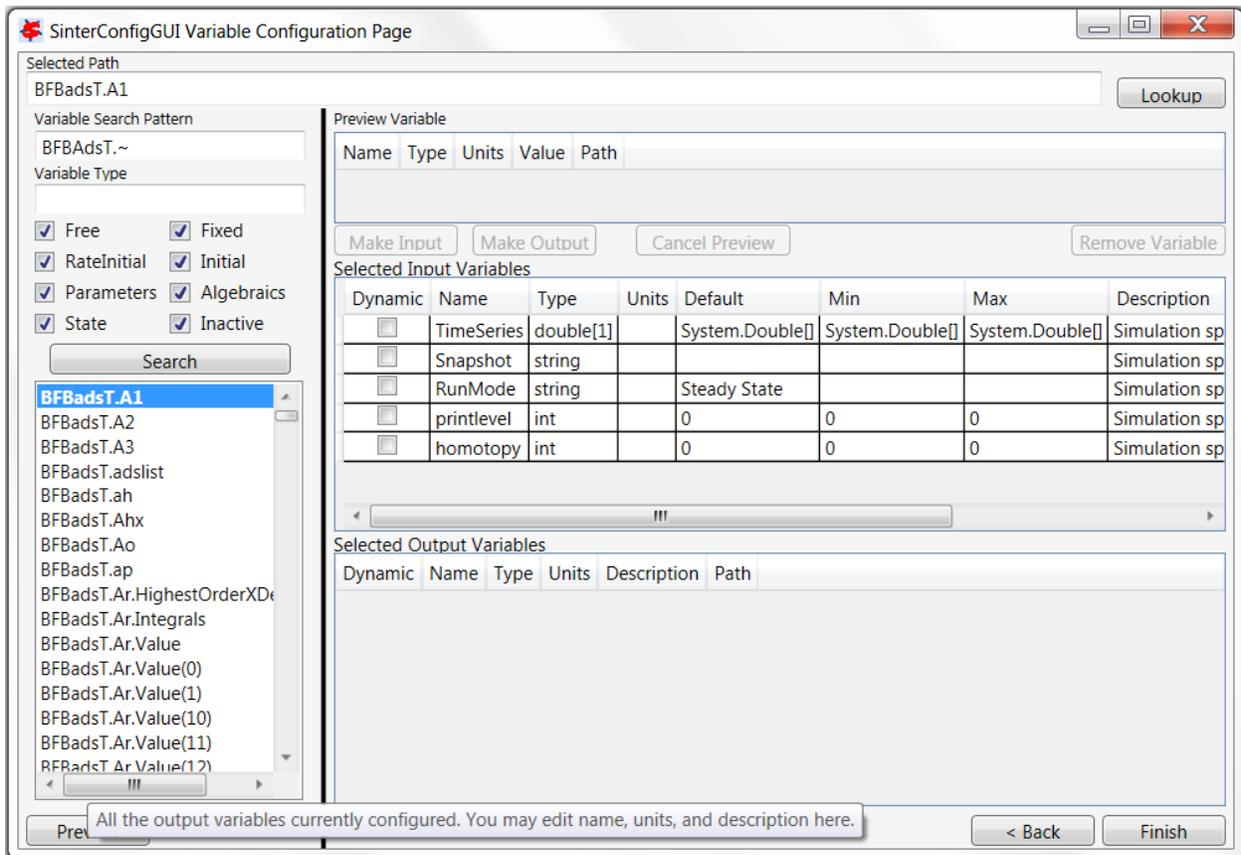


Fig. 7: SinterConfigGUI Variable Configuration Page BFBadsT.A1 Selected

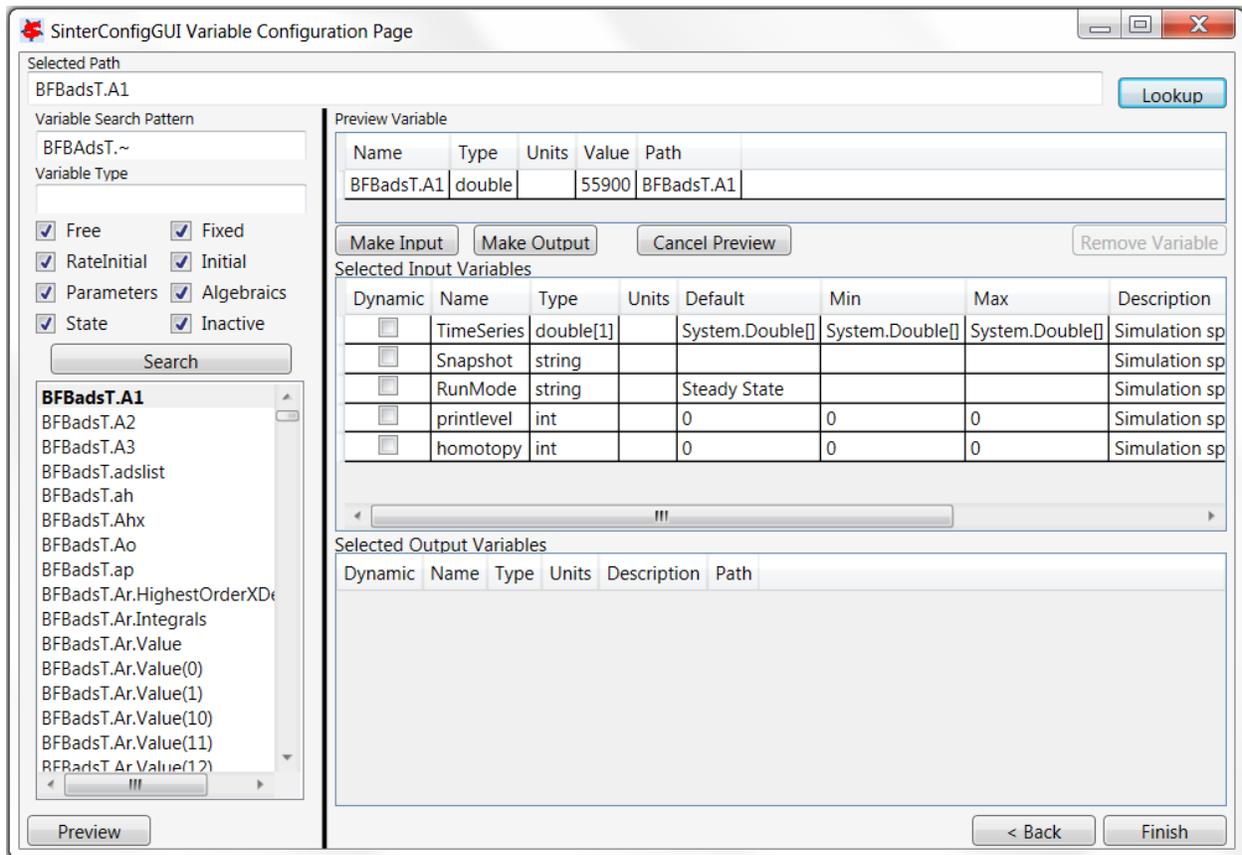


Fig. 8: SinterConfigGUI Variable Configuration Page BFBadsT.A1 Preview

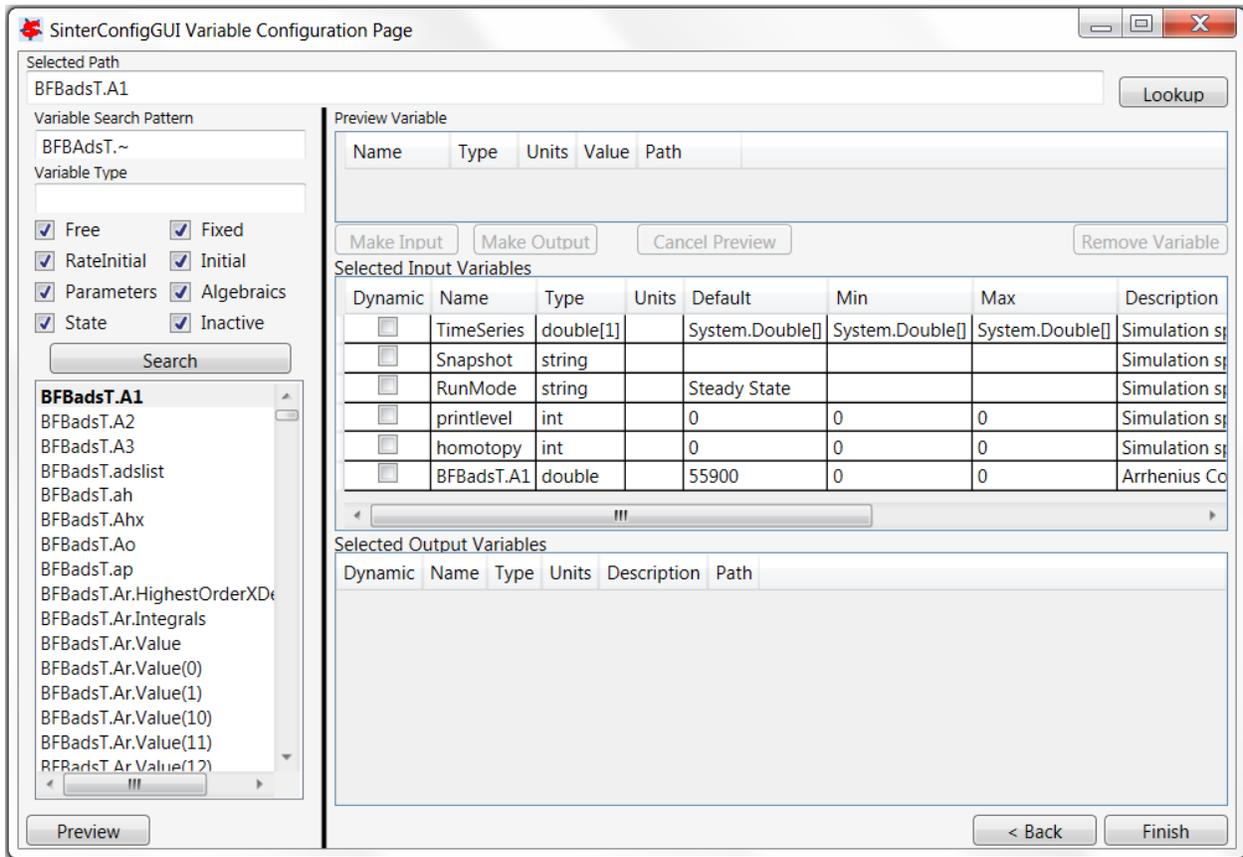


Fig. 9: SinterConfigGUI Variable Configuration Page BFBadsT.A1 Made Input

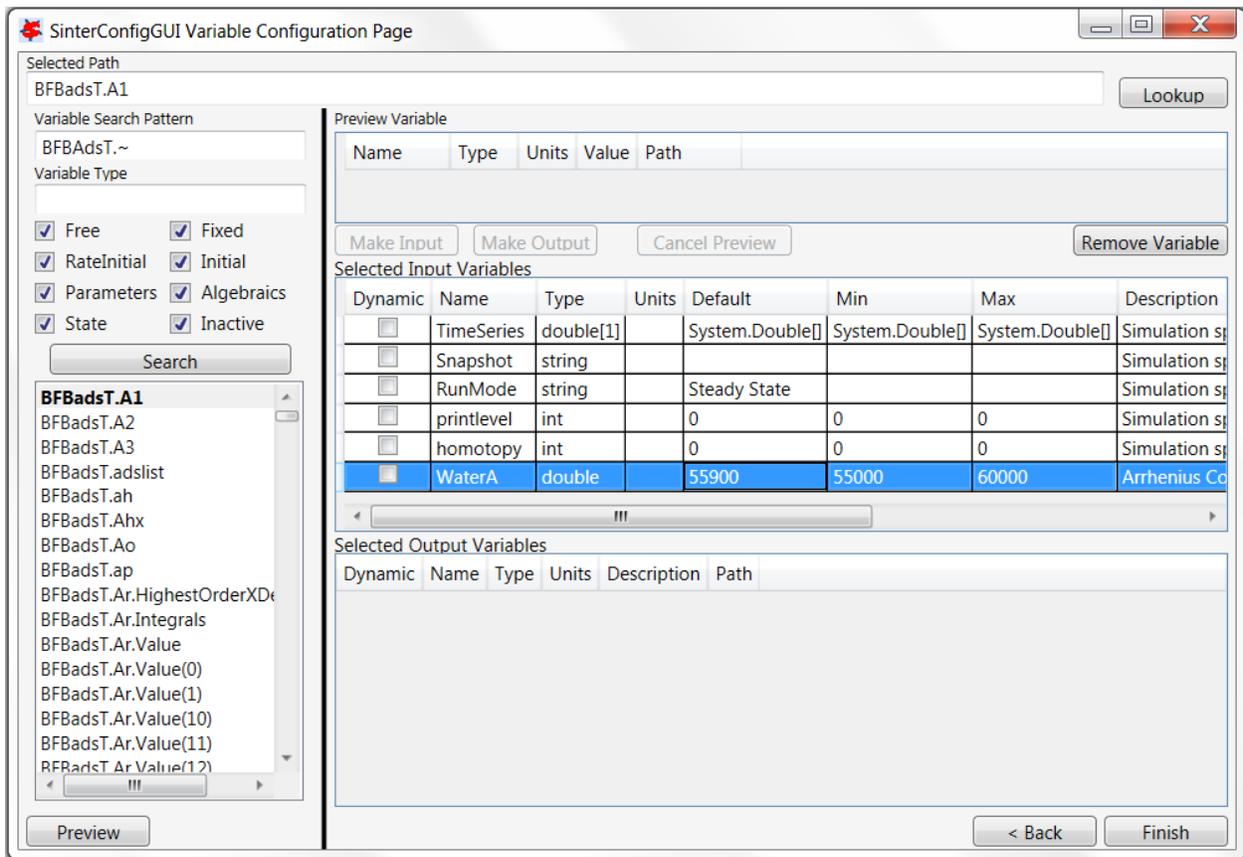


Fig. 10: SinterConfigGUI Variable Configuration Page BFBadsT.A1 Change Name

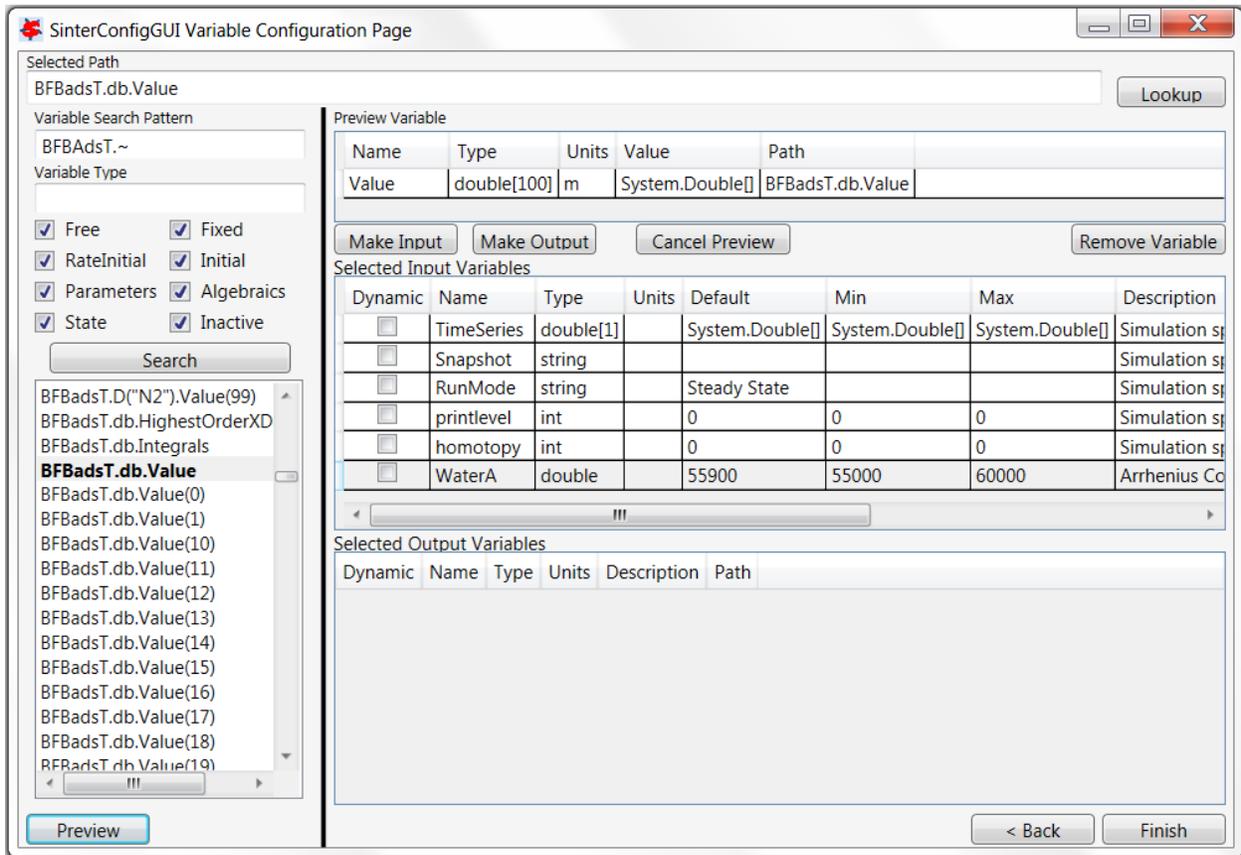


Fig. 11: SinterConfigGUI Variable Configuration Page Vector Preview

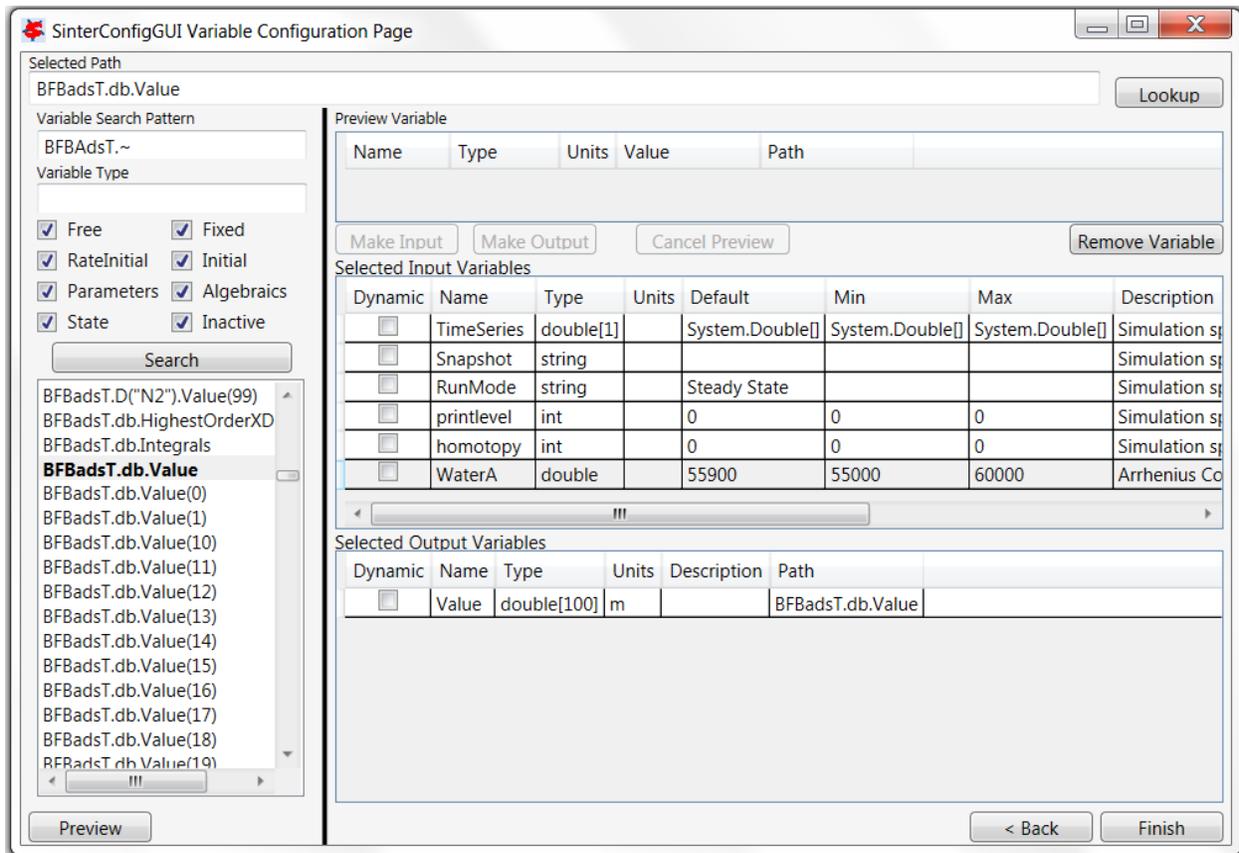


Fig. 12: SinterConfigGUI Variable Configuration Page Vector As Output

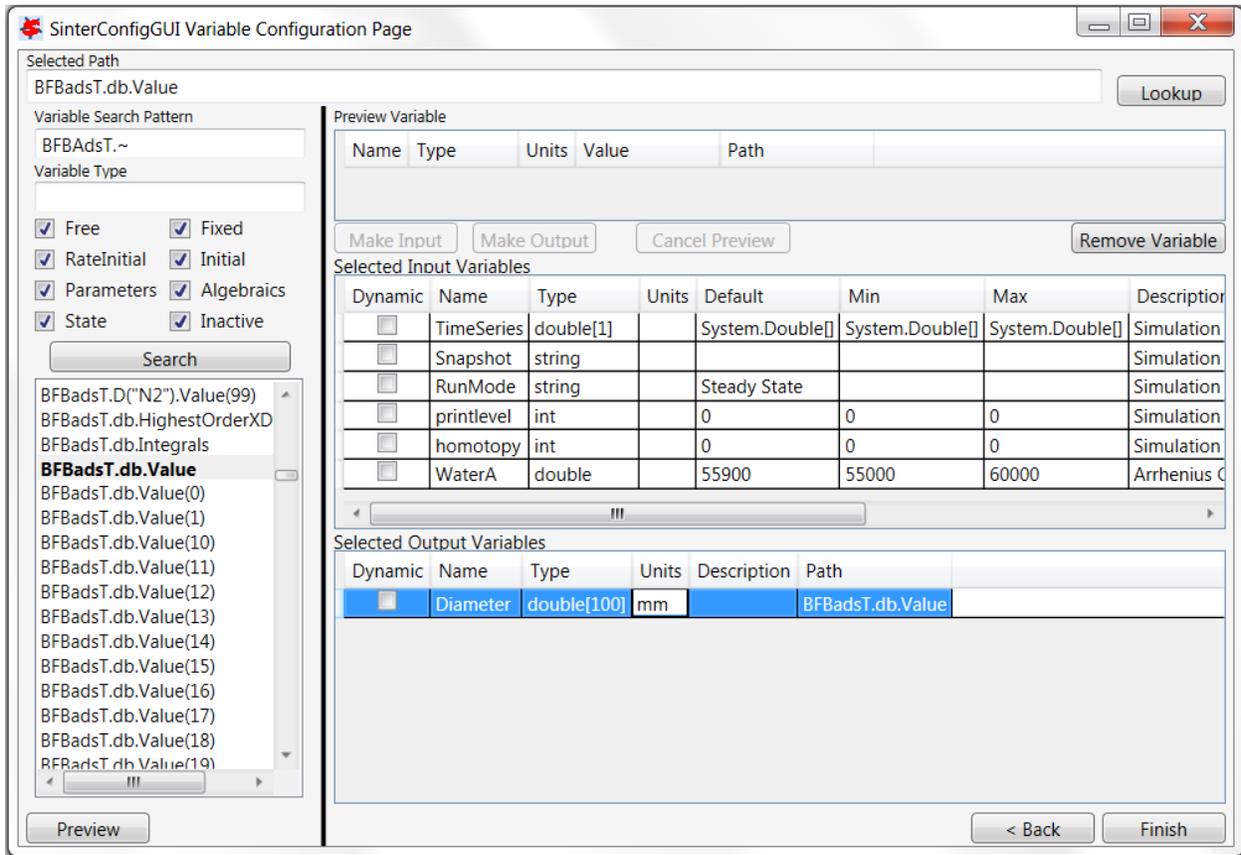


Fig. 13: SinterConfigGUI Variable Configuration Page Output Change Units

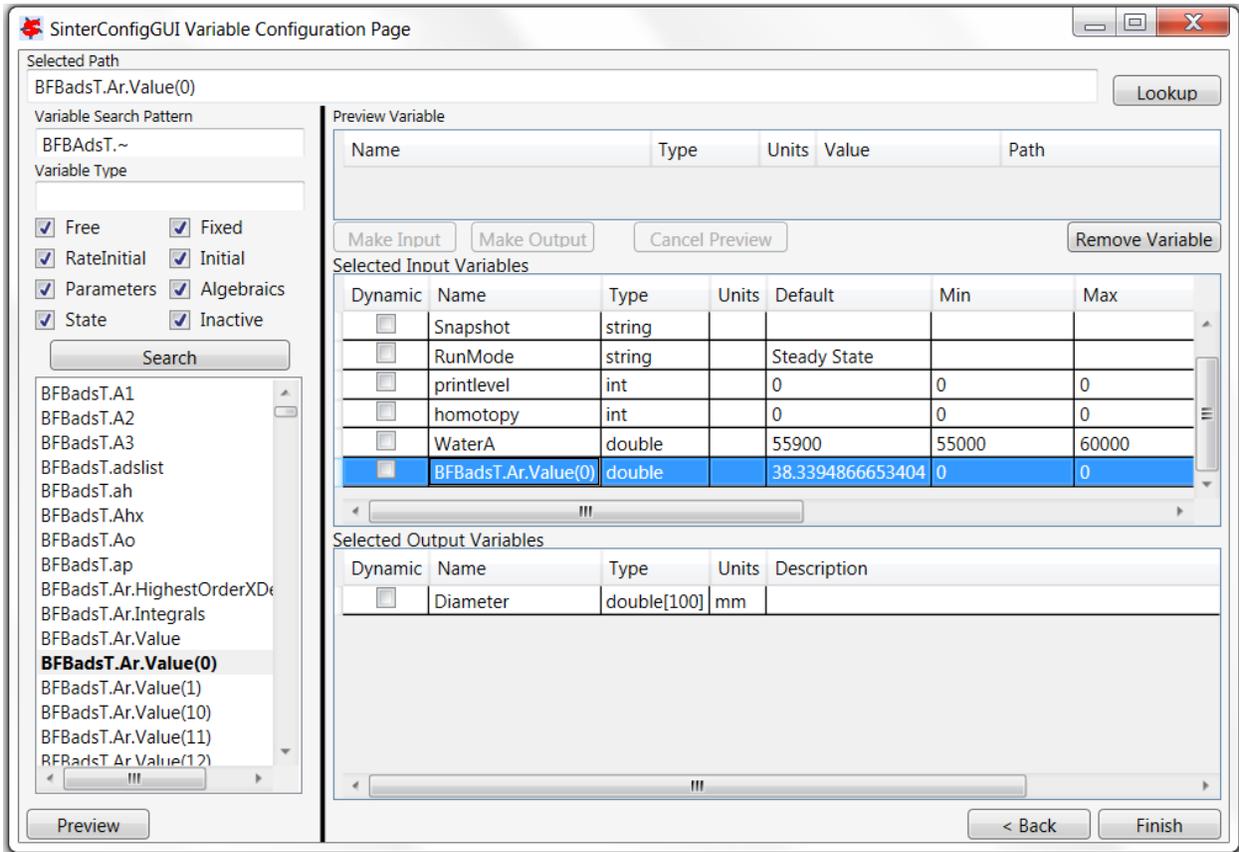


Fig. 14: SinterConfigGUI Variable Configuration Page Removal Demo

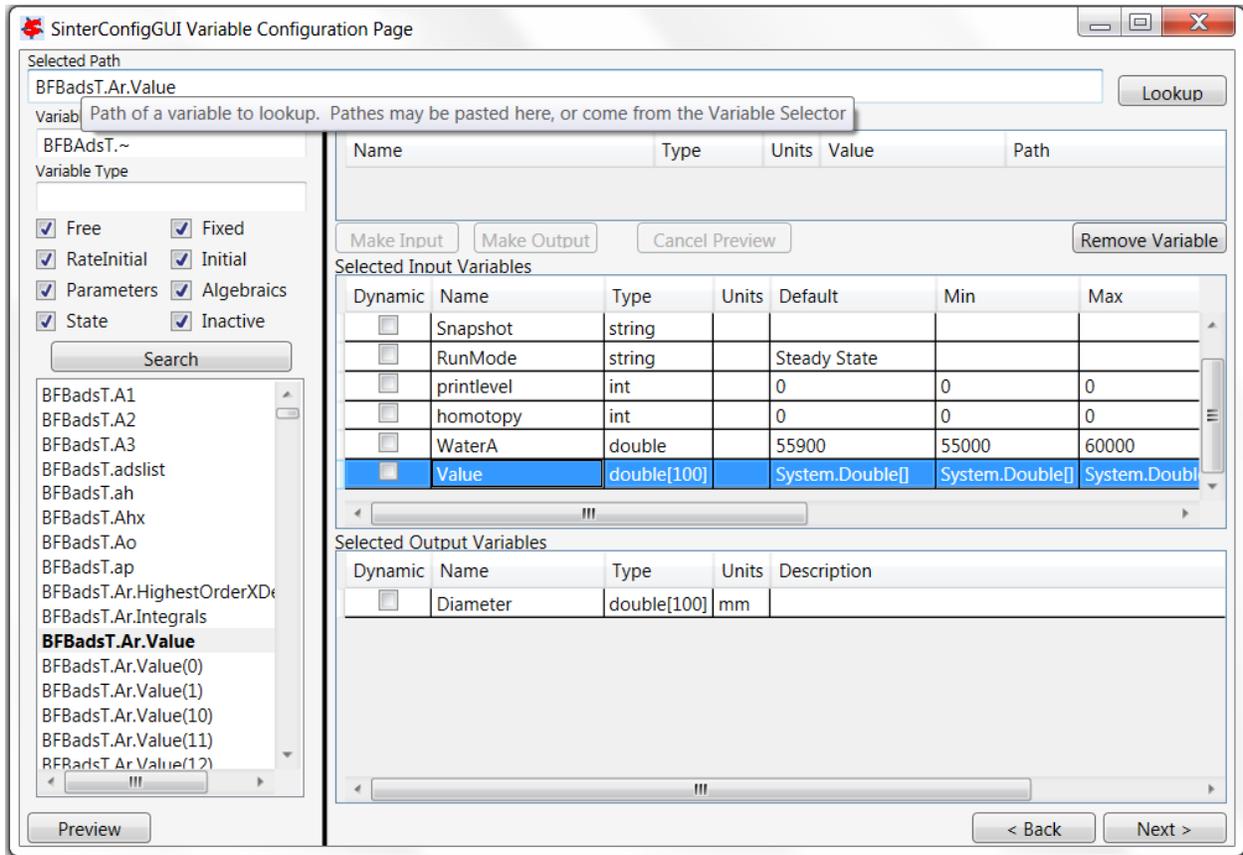


Fig. 15: SinterConfigGUI Variable Configuration Page Read Input

Vector Name	Size	Vector Data
TimeSeries	1	0
Value	75	40.3195002688879 38.2712119914265 38.4850839328382 38.6443788725324 38.6067069861193 38.4092285527587 38.4092285527587 38.4092285527587

Fig. 16: SinterConfigGUI Vector Default Initialization Input Page

19. The simulation is now setup. Save the configuration file by clicking **Finish**. The file is saved to the location specified on the SinterConfigGUI Simulation Meta-Data page. Clicking **Finish** will close the SinterConfigGUI, but NOT Aspen Custom Modeler. The user must close ACM manually.
20. If “SinterConfigGUI” was launched from FOQUS, the path to the configuration file is automatically passed to FOQUS. The next step in FOQUS is to click **OK** in the Add/Update Turbine Model window. FOQUS may then be used to upload it to TurbineLite or AWS FOQUS Cloud. If “SinterConfigGUI” was not launched from FOQUS (e.g., it was launched from the Start menu), the configuration file name must be entered in FOQUS manually.

Tutorial 2: Aspen Plus Configuration

The files (both the Aspen Plus file and the JSON file) for this tutorial are located in:
 examples/tutorial_files/SimSinter/Tutorial_2

Note: The **examples/** directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

1. The initial steps for opening a simulation and entering metadata for an Aspen Plus simulation are similar to ACM. Refer to the SimSinter ACM tutorial *Tutorial 1: Aspen Custom Modeler (ACM) Configuration*. In this tutorial, a flash model “Flash_Example.bkp” (located in the above-mentioned folder) is used as an example. Open the Aspen Plus file and enter the metadata as shown in Figure *SinterConfigGUI Simulation Meta-Data with Data Completed*.

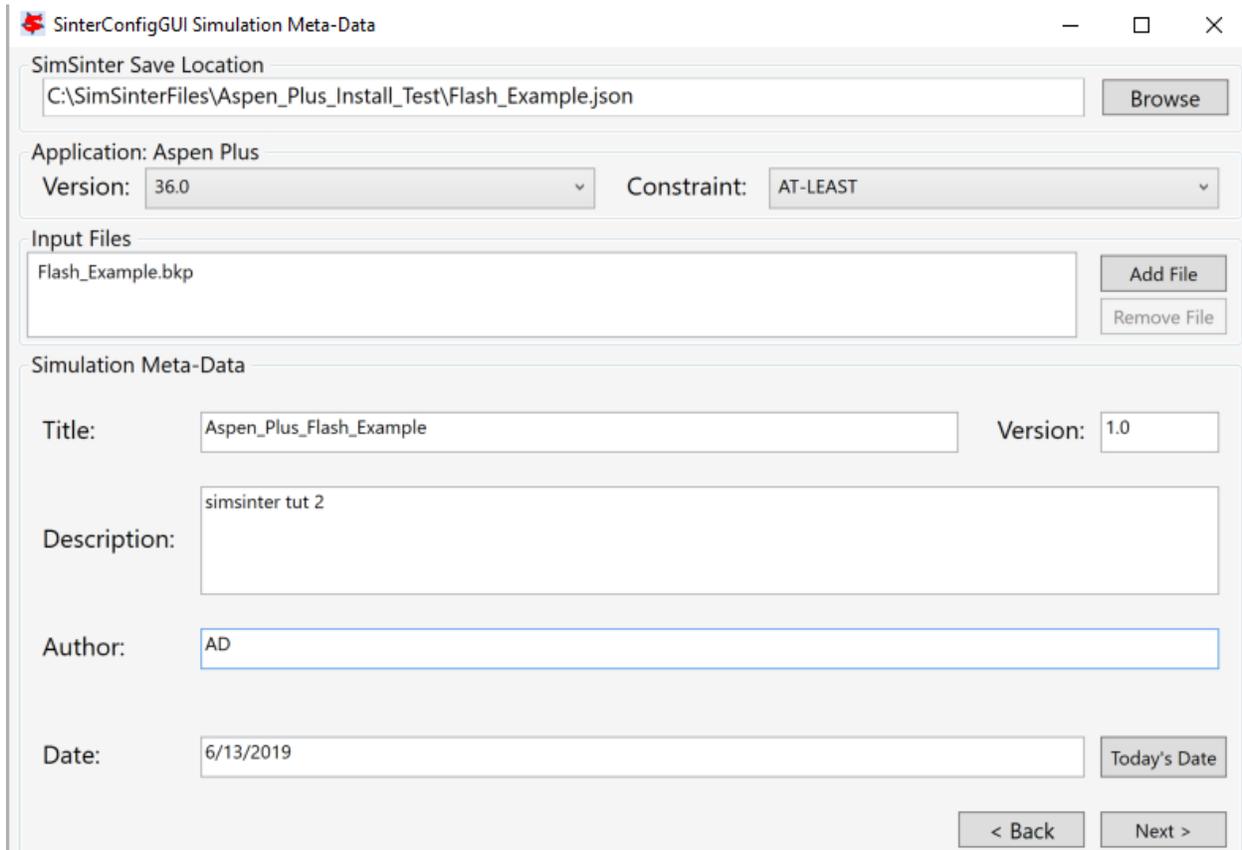


Fig. 17: SinterConfigGUI Simulation Meta-Data with Data Completed

- The **SinterConfigGUI Variable Configuration Page** displays as illustrated in Figure *SinterConfigGUI Variable Configuration Page Empty Variables*. Aspen Plus has no settings, so there are no setting variables in the input section. Unlike ACM, AspenPlus displays the **Variable Tree** on the left side, so the user can explore the tree as is done in Aspen Plus Tools → Variable Explorer. Unfortunately, searching is not possible.

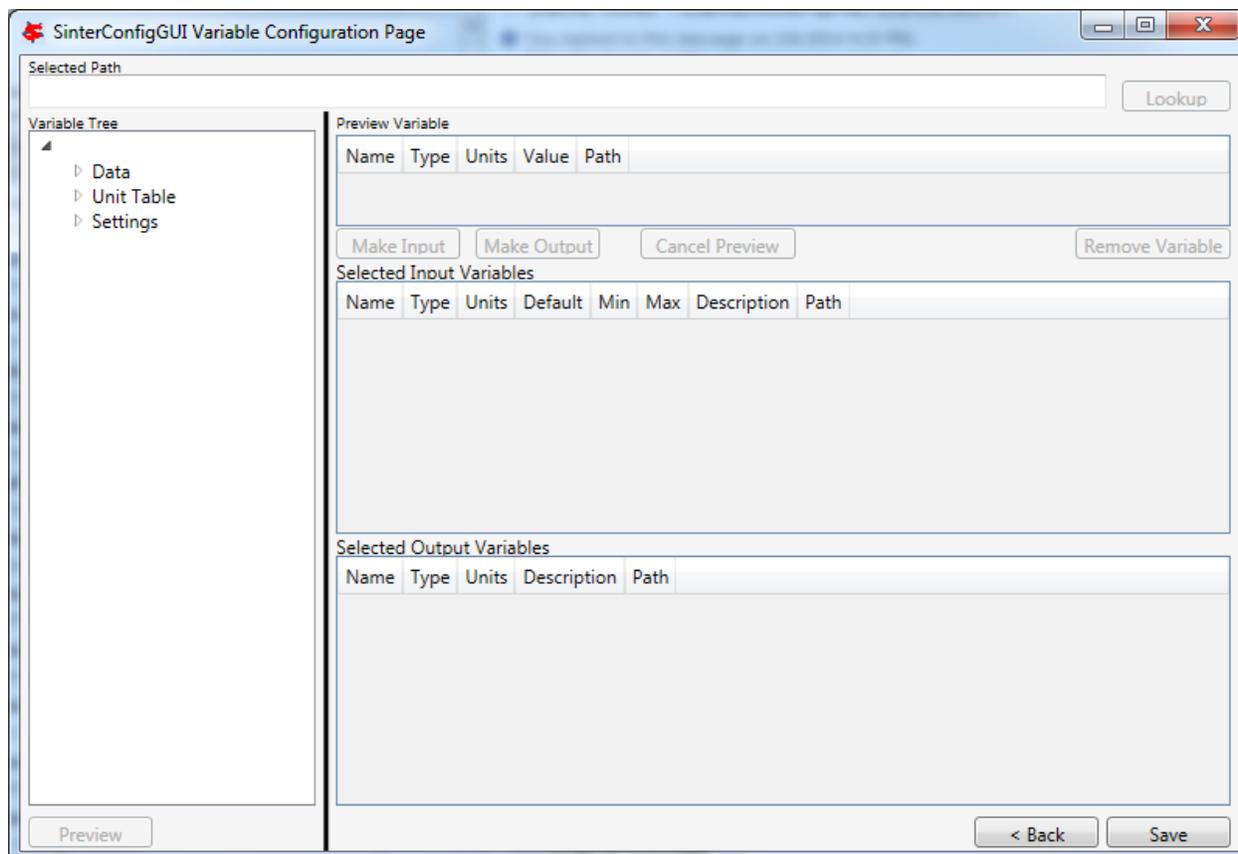


Fig. 18: SinterConfigGUI Variable Configuration Page Empty Variables

- Variable Tree** nodes can be expanded for searching (Figure *SinterConfigGUI Variable Configuration Page Expanded Aspen Plus Variable Tree*).
- The user can type the node address directly into the **Selected Path** field (this is useful for copy/paste from Aspen Plus' Variable Explorer) (Figure *SinterConfigGUI Variable Configuration Page Aspen Plus Variable Selected*). Click **Lookup** or **Preview** (which automatically causes the tree to expand and selects selected variables in the **Variable Tree**).
- To make the temperature of the Flash chamber an **Input Variable**, click **Make Input**. Additionally, the user can **Name** the variable, fix the **Description**, and enter the **Min/Max** fields by clicking on the appropriate text and entering it.
- Select an **Output Variable**, **Preview** it, and click **Make Output**. Next, update the fields as with the **Input Variable** to give a better **Name** and **Description**.
- The task is complete. Save it by clicking **Save** or CTRL+S. The file is saved to the location specified in the SinterConfigGUI Simulation Meta-Data page. If the user wishes to save a copy under a different name, navigate back to the SinterConfigGUI Simulation Meta-Data page and change the name.

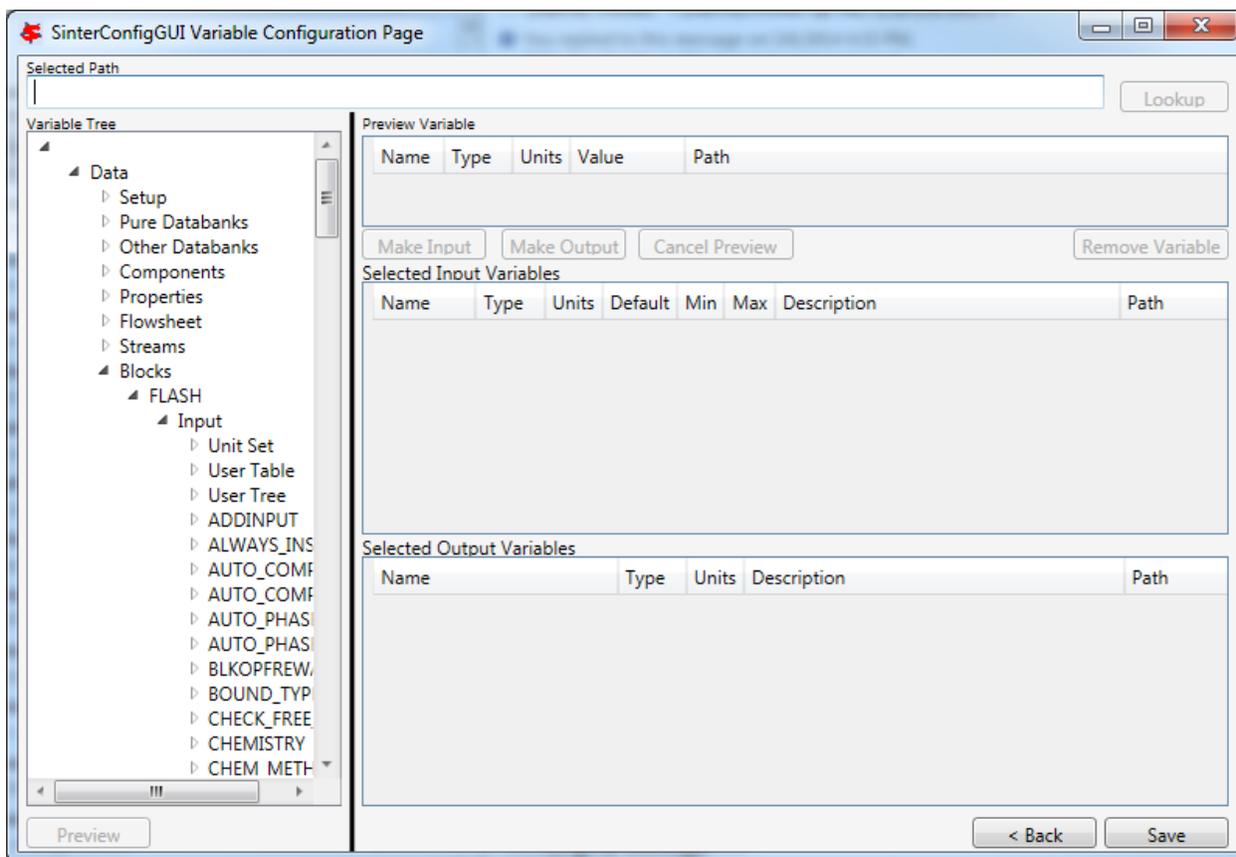


Fig. 19: SinterConfigGUI Variable Configuration Page Expanded Aspen Plus Variable Tree

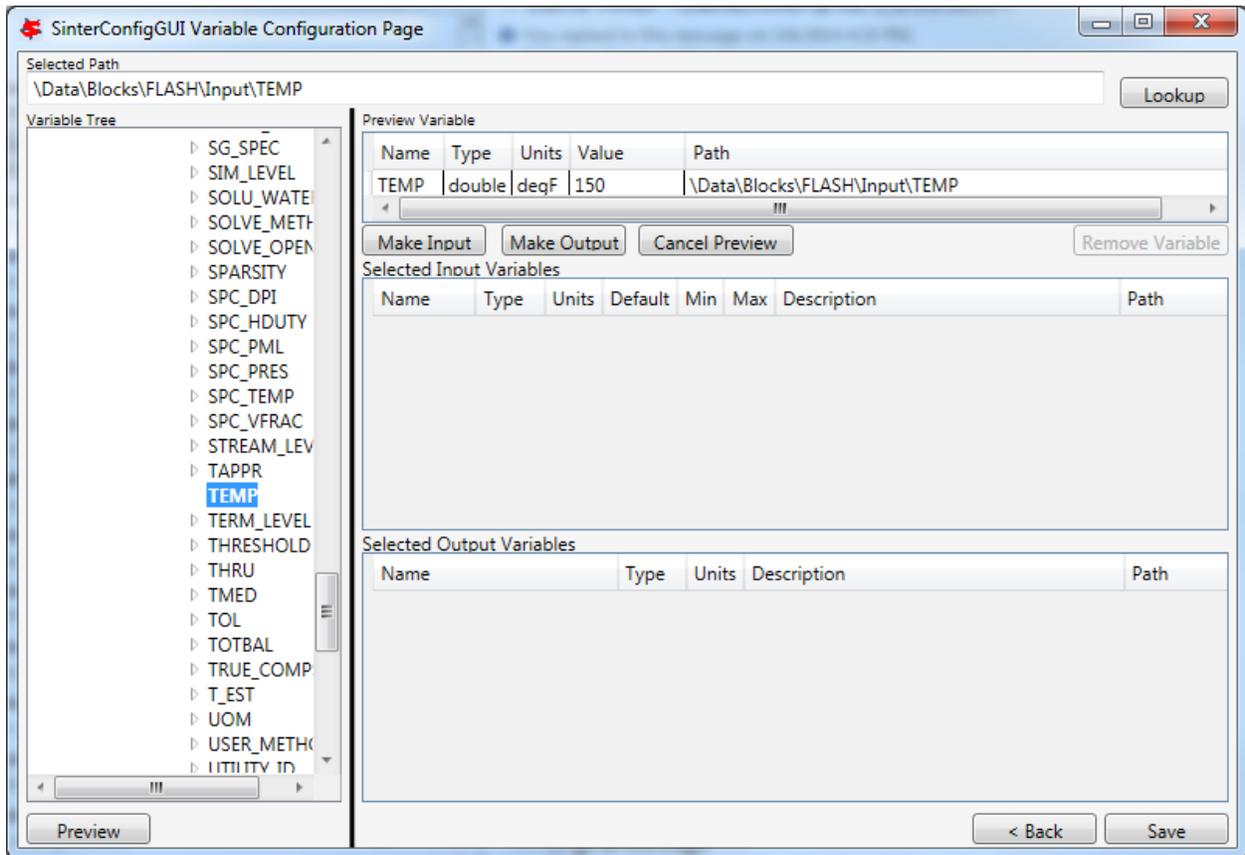


Fig. 20: SinterConfigGUI Variable Configuration Page Aspen Plus Variable Selected

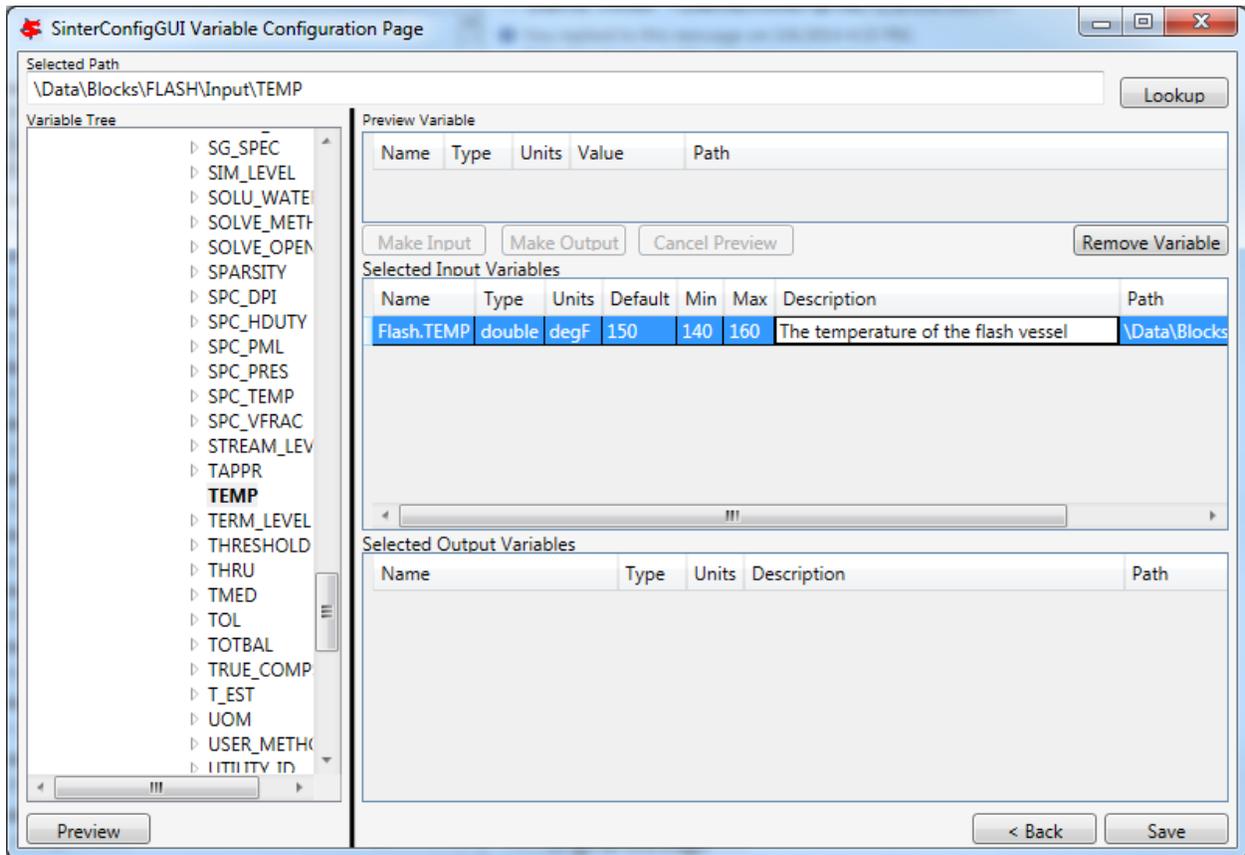


Fig. 21: SinterConfigGUI Variable Configuration Page Input Variable

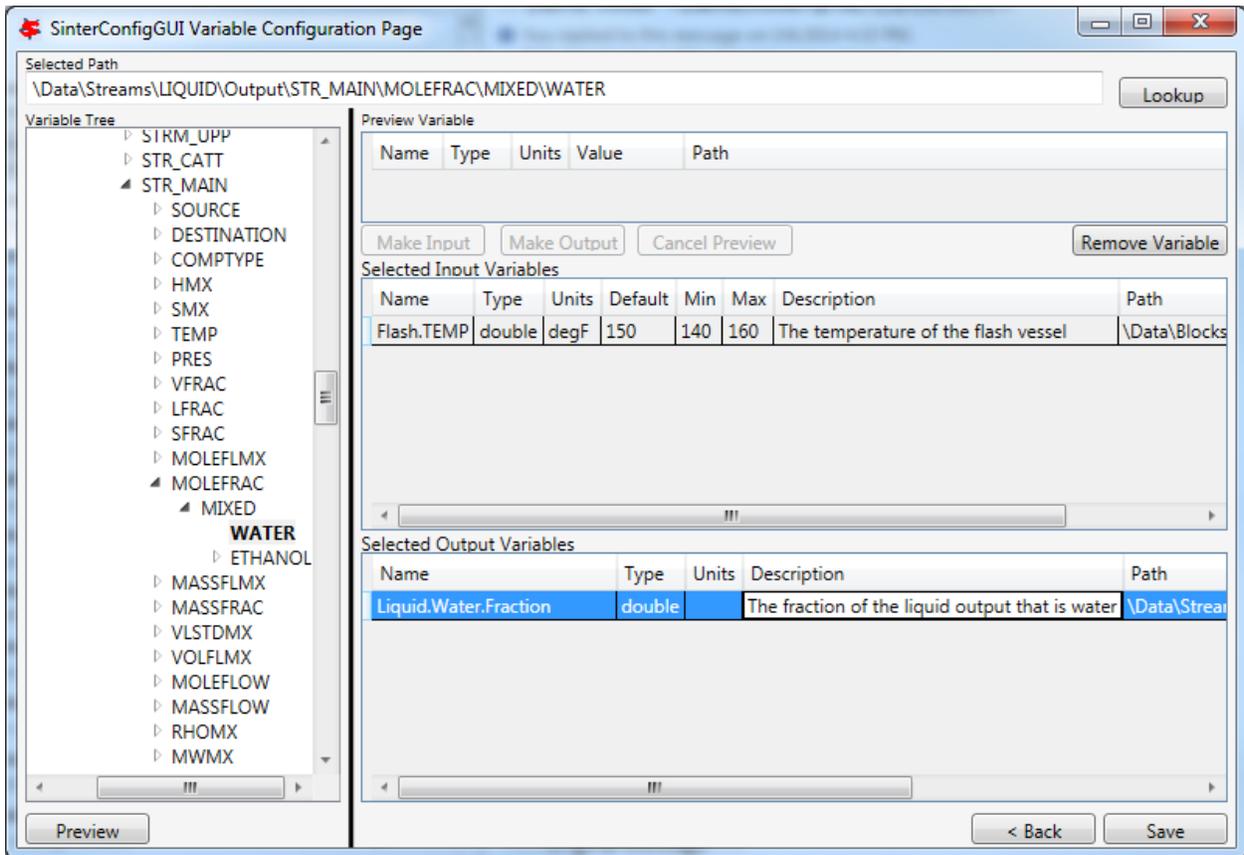


Fig. 22: SinterConfigGUI Variable Configuration Page Output Variable

Tutorial 3: Microsoft Excel Configuration

The files (both the Excel and the JSON files) for this tutorial are located in:
 examples/tutorial_files/SimSinter/Tutorial_3

Note: The **examples/** directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

1. The “SinterConfigGUI” can be launched from FOQUS, via the **Create/Edit** button found in **File**→ **Add/Update Model to Turbine** or “SinterConfigGUI” may be run on its own by selecting **CCSI Tools** → **FOQUS** → **SinterConfigGUI** from the Start menu.
2. The splash window displays, as shown in Figure *SinterConfigGUI Splash Screen*. The user may click the splash screen to proceed, or wait 10 seconds for it to close automatically.

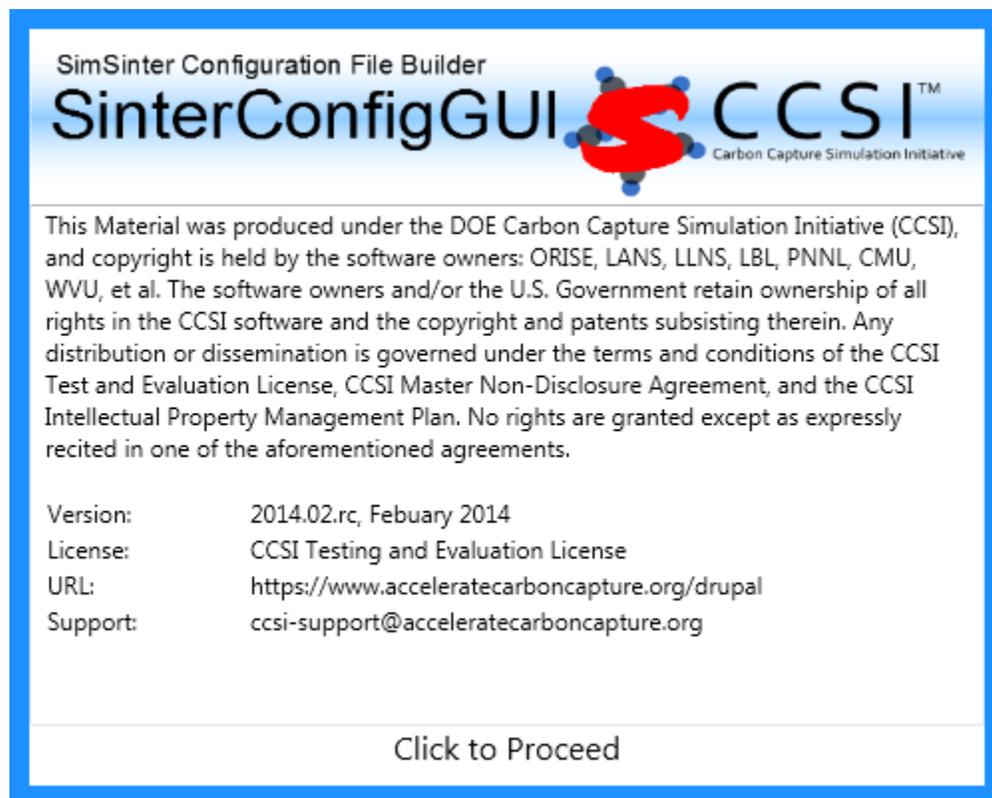


Fig. 23: SinterConfigGUI Splash Screen

3. The SinterConfigGUI Open Simulation window displays (Figure *SinterConfigGUI Open Simulation Window*). If “SinterConfigGUI” was opened from FOQUS, the filename text box already contains the correct file. To proceed immediately click **Open File and Configure Variables** or click **Browse** to search for the file. For this tutorial, a fresh copy of the BMI (body mass index) test is opened (exceltest.xlsm). It is located in:
 examples/tutorial_files/SimSinter/Tutorial_3
4. Microsoft Excel starts in the background. This is so the user can observe things about the worksheet while working on the configuration file.
5. In the “SinterConfigGUI” the SinterConfigGUI Simulation Meta-Data page is now displayed (Figure *SinterConfigGUI Simulation Meta-Data Save Text Box*). The first and most important piece of metadata is **Save Location**

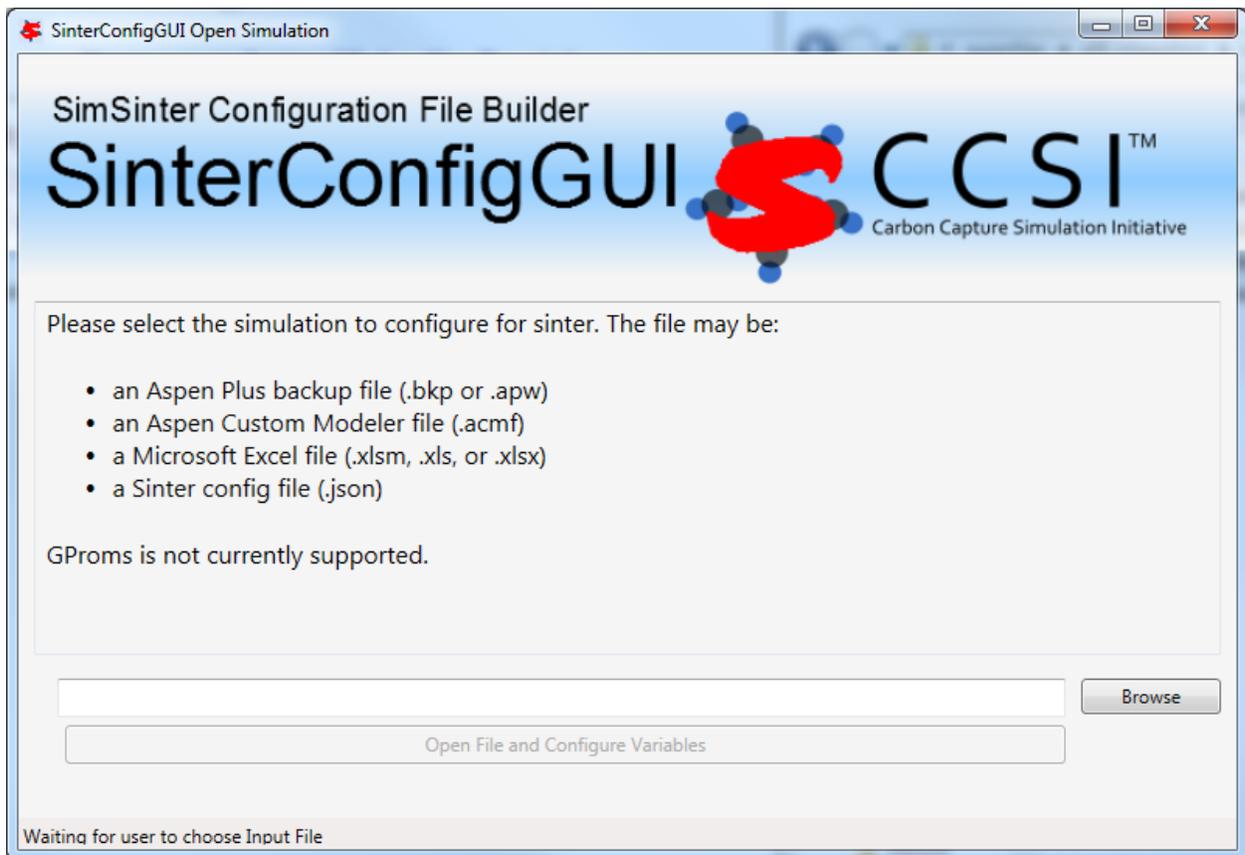


Fig. 24: SinterConfigGUI Open Simulation Screen

at the top of the window. This is where the sinter configuration file is saved. The system attempts to locate a reasonable file location and file name; however, the user must confirm the correct file location, since it automatically overwrites whatever filename currently exists.

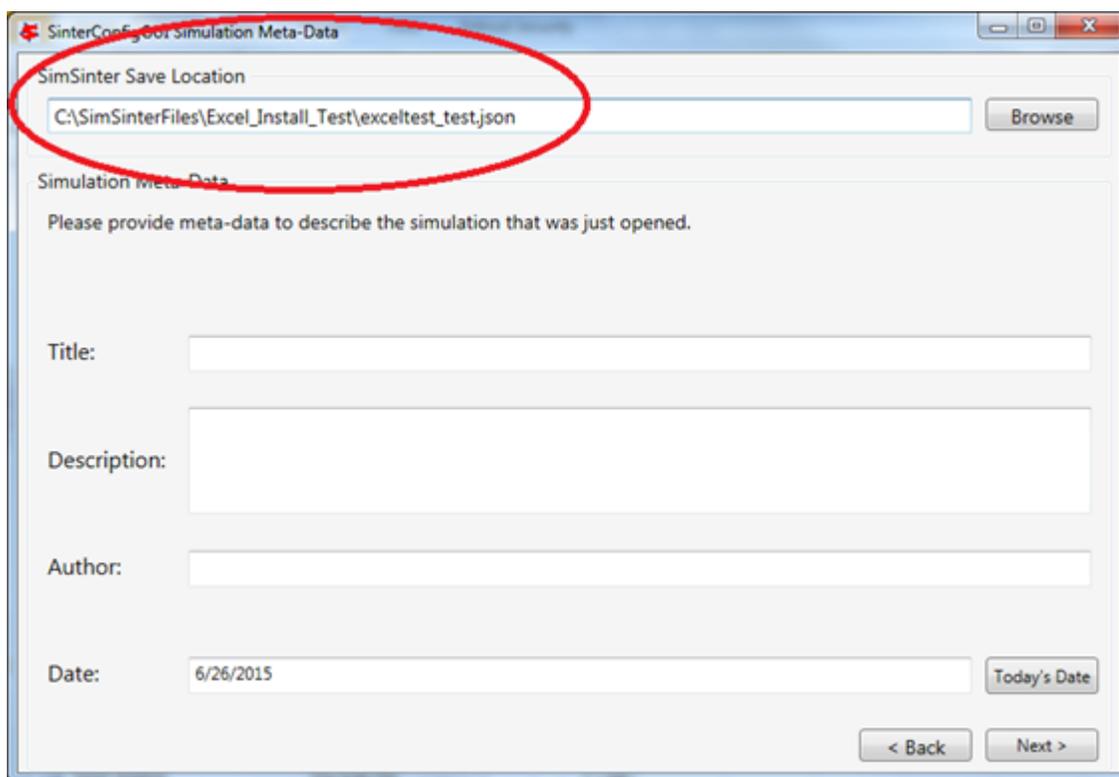


Fig. 25: SinterConfigGUI Simulation Meta-Data Save Text Box

6. Continue to complete in the remaining fields and click **Next**.
7. In the SinterConfigGUI Variable Configuration Page, (Figure *SinterConfigGUI Variable Configuration Page before Input*) notice that the Excel setting variable **macro** is already included in the **Selected Input Variables**. If the Excel spreadsheet has a macro that should be run after SimSinter sets the inputs, but before SimSinter gets the outputs, enter the macros name in the **Default** text box. If the Default box is left blank, no macro is run (unless a name is supplied in the input variables when running the simulation). If the user needs to run multiple macros (e.g., Macro1 and Macro2), we recommend that the user create a “Master” macro in the Excel file that automatically runs Macro1 and Macro2 using the Call statement. Let’s suppose that the “Master” macro is named MasterMacro. Then, in SimSinter, the user will need to type in MasterMacro in the **Default** text box under the Excel setting variable **macro**.
8. The Excel simulation has the same **Variable Tree** structure as Aspen Plus, as shown in (Figure *SinterConfigGUI Variable Configuration Page Selecting a Variable from the Excel Variable Tree*). Only the variables in the active section of the Excel spreadsheet appear in the **Variable Tree**. If, for some reason, a cell does not appear in the tree, the user may manually enter the cell into the **Selected Path** text box. In this case, select the “height\$C\$4” variable.

Note: Row is first in the **Variable Tree**, yet column is first in the **Path**.
9. If the user double-clicks, presses enter, clicks **Preview**, or clicks **Lookup**, the variable will be displayed in the **Preview Variable** frame. Click the **Make Input** button to make the variable an input variable. Now the variable is in the **Selected Input Variables** section, and its meta-data may be edited (Figure *SinterConfigGUI Variable Configuration Page Description “Joe’s Height”*).

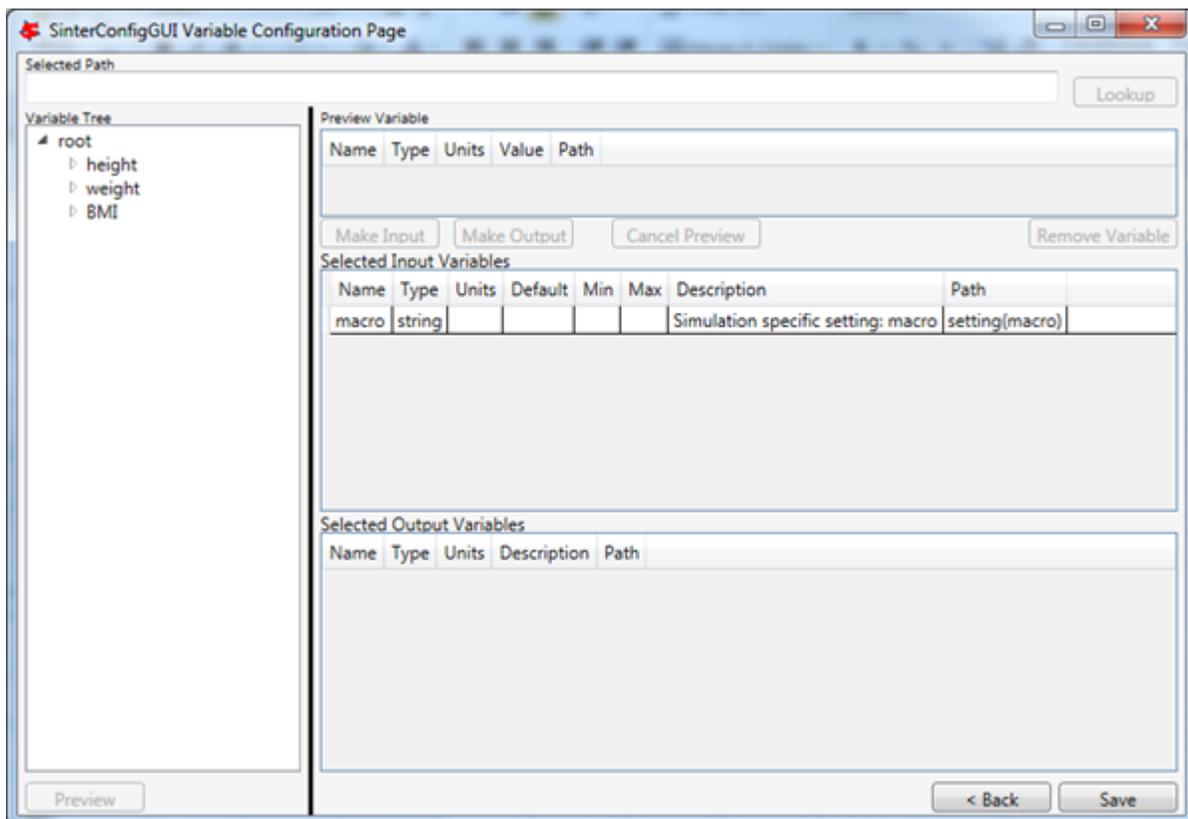


Fig. 26: SinterConfigGUI Variable Configuration Page before Input

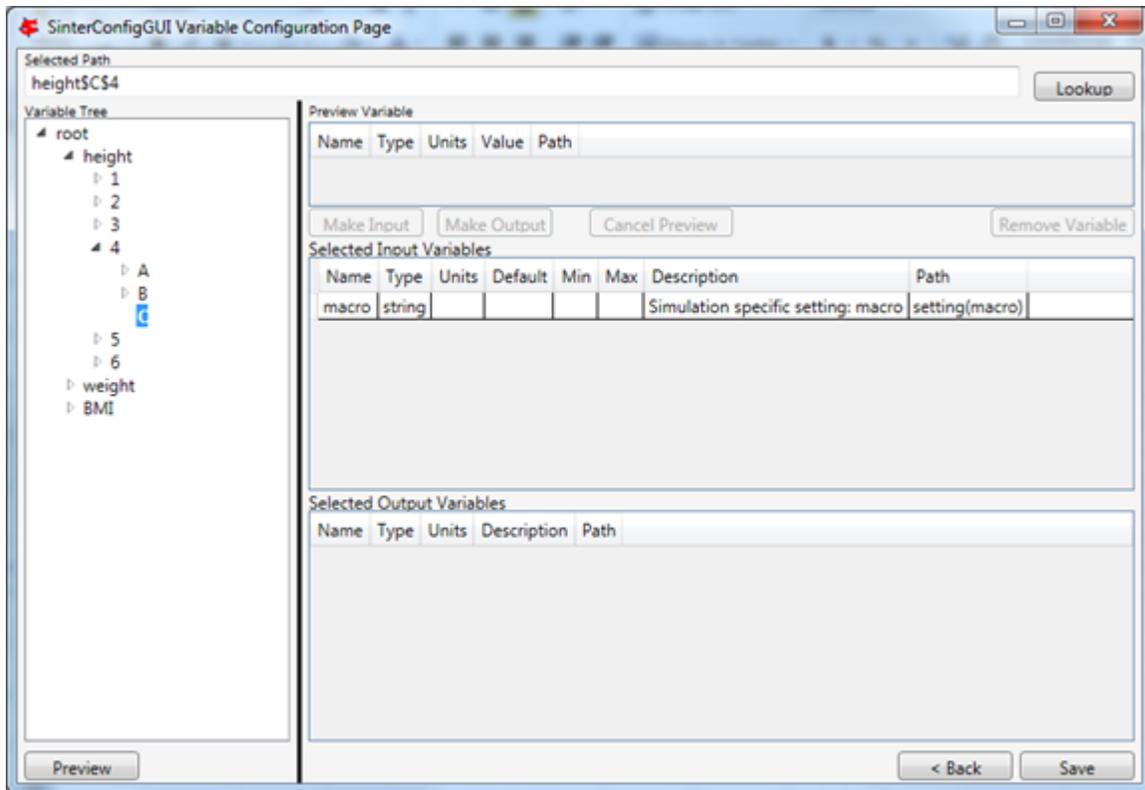


Fig. 27: SinterConfigGUI Variable Configuration Page Selecting a Variable from the Excel Variable Tree

10. Enter an output variable (such as, “BMI\$C\$3”), by selecting the variables in the **Variable Tree**, clicking **Preview**, and then clicking **Make Output** (Figure *SinterConfigGUI Variable Configuration Page Selecting Excel Output Variables*).
11. The simulation is now set up. To save the configuration file, click **Finish** or press CTRL+S. The file is saved to the location that was set on the SinterConfigGUI Simulation Meta-Data window. A user can save a copy under a different name, by navigating back to the SinterConfigGUI Simulation Meta-Data window using **Back**, and then changing the name. This creates a second version of the file.

Tutorial 4: gPROMS Configuration

gPROMS is significantly different from the other simulators SimSinter supports, and the workflow is also significantly different. If you plan to use gPROMS simulations with FOQUS, the CCSI team strongly encourages you to read the [SimSinter gPROMS Technical Manual](<https://github.com/CCSI-Toolset/SimSinter/blob/master/docs/SimSinter%20gPROMS%20Technical%20Manual.pdf>).

[//github.com/CCSI-Toolset/SimSinter/blob/master/docs/SimSinter%20gPROMS%20Technical%20Manual.pdf](https://github.com/CCSI-Toolset/SimSinter/blob/master/docs/SimSinter%20gPROMS%20Technical%20Manual.pdf)).

Unlike Aspen, changes must be made to the gPROMS simulation process in order to work with SimSinter. Therefore, this section consists of a series of tutorials for every step of configuring gPROMS and SimSinter to work together. All the tutorials are required in order to have a gPROMS simulation be runnable with SimSinter. They are divided up to make later reference easier.

The files for this tutorial are located in: `examples/tutorial_files/SimSinter/Tutorial_4`

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

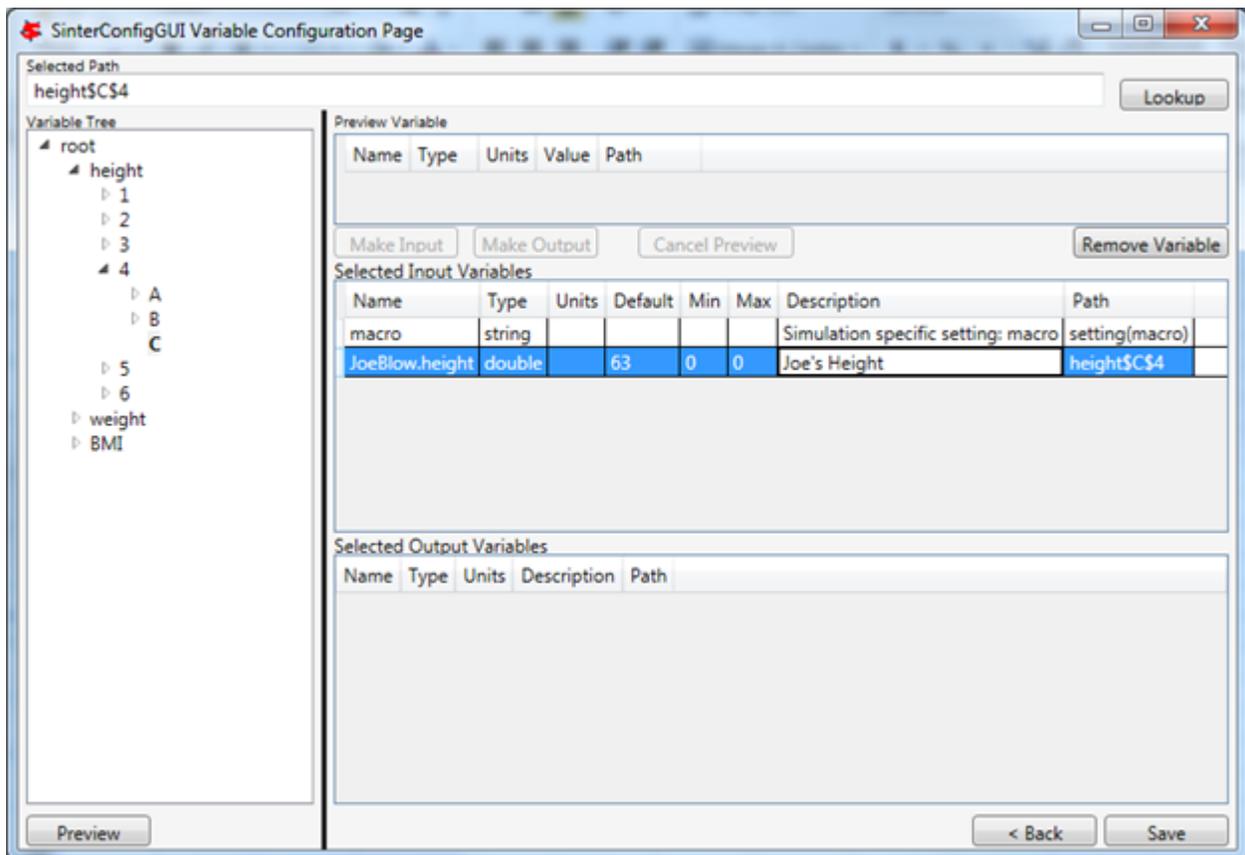


Fig. 28: SinterConfigGUI Variable Configuration Page Description "Joe's Height"

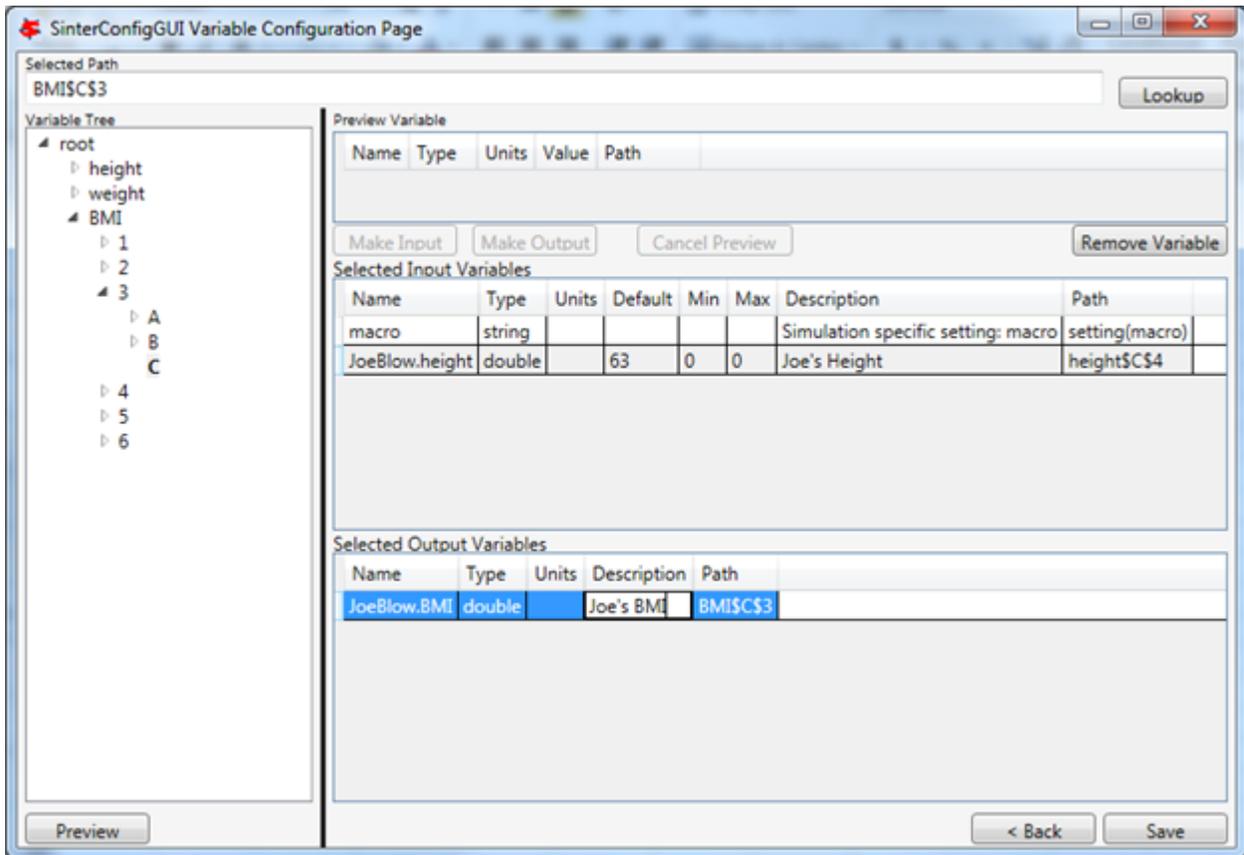


Fig. 29: SinterConfigGUI Variable Configuration Page Selecting Excel Output Variables

Configuring gPROMS to Work with SimSinter

Unlike Aspen, changes have to be made to the gPROMS simulation process in order to work with SimSinter. In fact, SimSinter does not define the inputs to the simulation, gPROMS does. On the other hand, gPROMS does not determine the outputs, SimSinter does. This odd and counter-intuitive situation is the result of how gPROMS `gO:Run_XML` is designed.

The modification to the gPROMS simulation must be done by a developer with an intimate understanding of the simulation, usually the simulation writer. In some cases additional variables may need to be added to handle an extra step between taking the input and inserting it into the variable where gPROMS will use the data.

1. Open the gPROMS simulation file (ends in .gPJ) in ModelBuilder 4.0 or newer. For this example, use the gPROMS install test file “BufferTank_FO.gPJ”, found in:

examples/tutorial_files/SimSinter/Tutorial_4

Double-click on the .gPJ file to open ModelBuilder, as shown in Figure *Opening BufferTank in gPROMS Model Builder*.

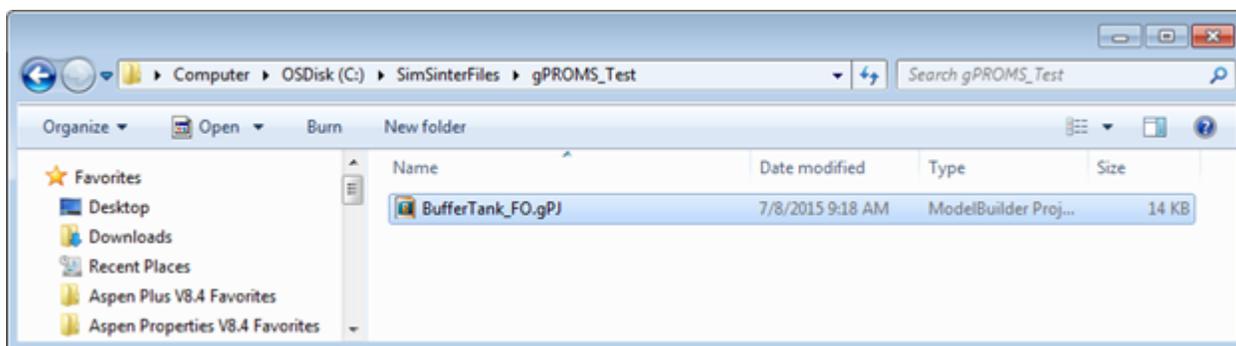


Fig. 30: Opening BufferTank in gPROMS Model Builder

2. This simulation was originally a simple BufferTank simulation. However, it was modified into an example of all the different kinds of variables the user can pass into gPROMS via SimSinter. Therefore, it has a lot of extra variables that do not really do anything, with very generic names, like “SingleInt.” The simulation consists of a single model, “BufferTank”, that contains all the simulation logic, and most of the parameter and variable declarations. The SimSinter simulation will change some of these PARAMETERS and VARIABLES to change the output of the simulation.
3. The example file contains two Processes. SimSinter can only run gPROMS Processes, so any gPROMS simulation must be driven from a Process. “SimulateTank” is the original BufferTank example with hardcoded values, “SimulateTank_Sinter” contains the example of setting values with Sinter. The “SimulateTank_Sinter” example will be recreated in this tutorial.
4. First copy the existing hard-coded Process “SimulateTank”.
5. Right-click on Processes and select **Paste** to make a new process.
6. The new process will be named “SimulateTank_1”. Rename the process by right-clicking on it and selecting **Rename**.
7. Now open up the new “SimulateTank_tutorial” Process. It has the same hard-coded values as “SimulateTank”.
8. First, the user needs to add a FOREIGN_OBJECT named “FO” in the PARAMETER section. Then the user needs to set that FOREIGN_OBJECT to “SimpleEventFOI::dummy” in the SET section. This FOREIGN_OBJECT is how inputs are received from SimSinter.
9. This particular simulation has a large number of input variables that simply demonstrate how to set different types. These are named based on their type. Any variable named similarly to “SingleInt” or “ArraySelector” can

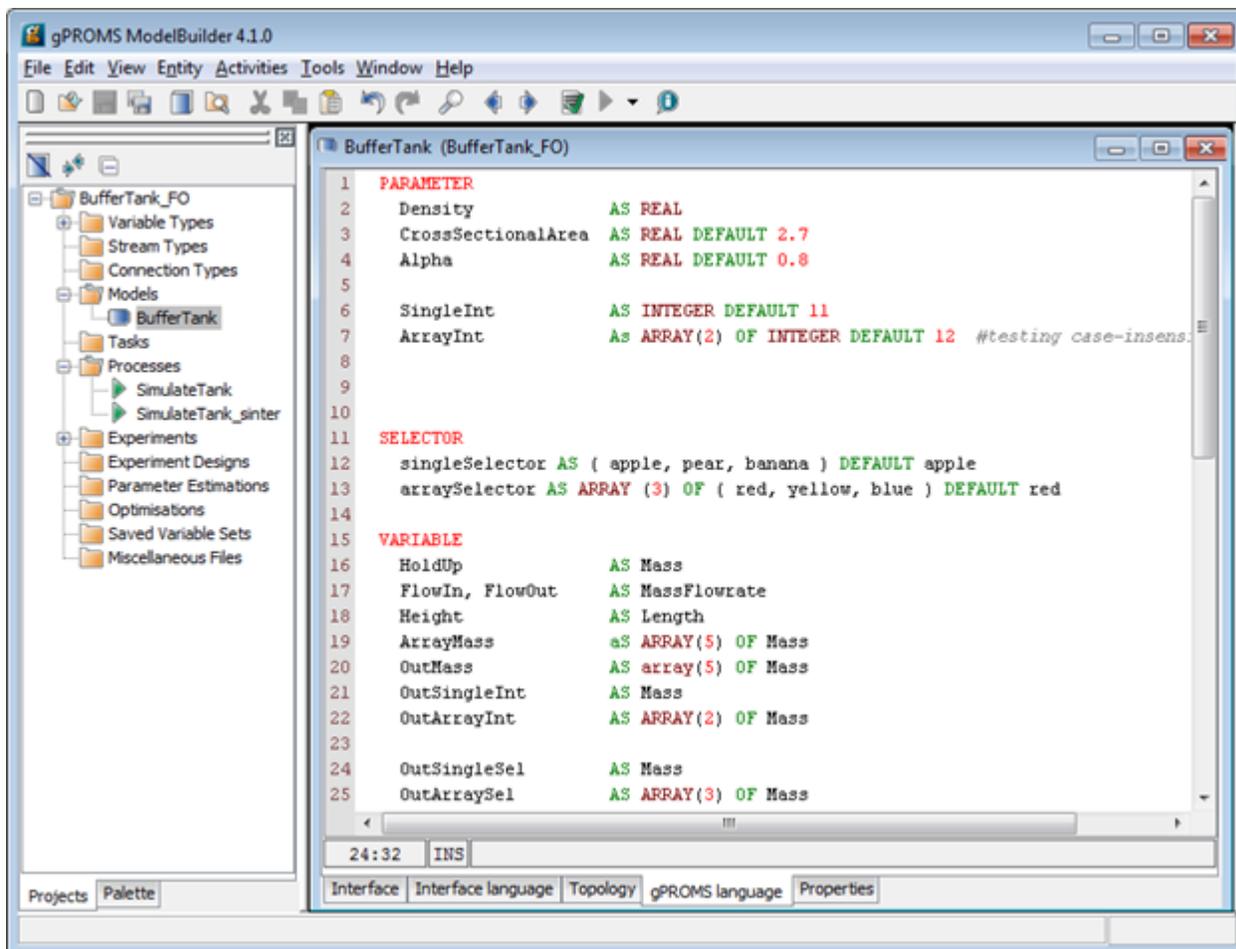


Fig. 31: Viewing BufferTank in gPROMS Model Builder

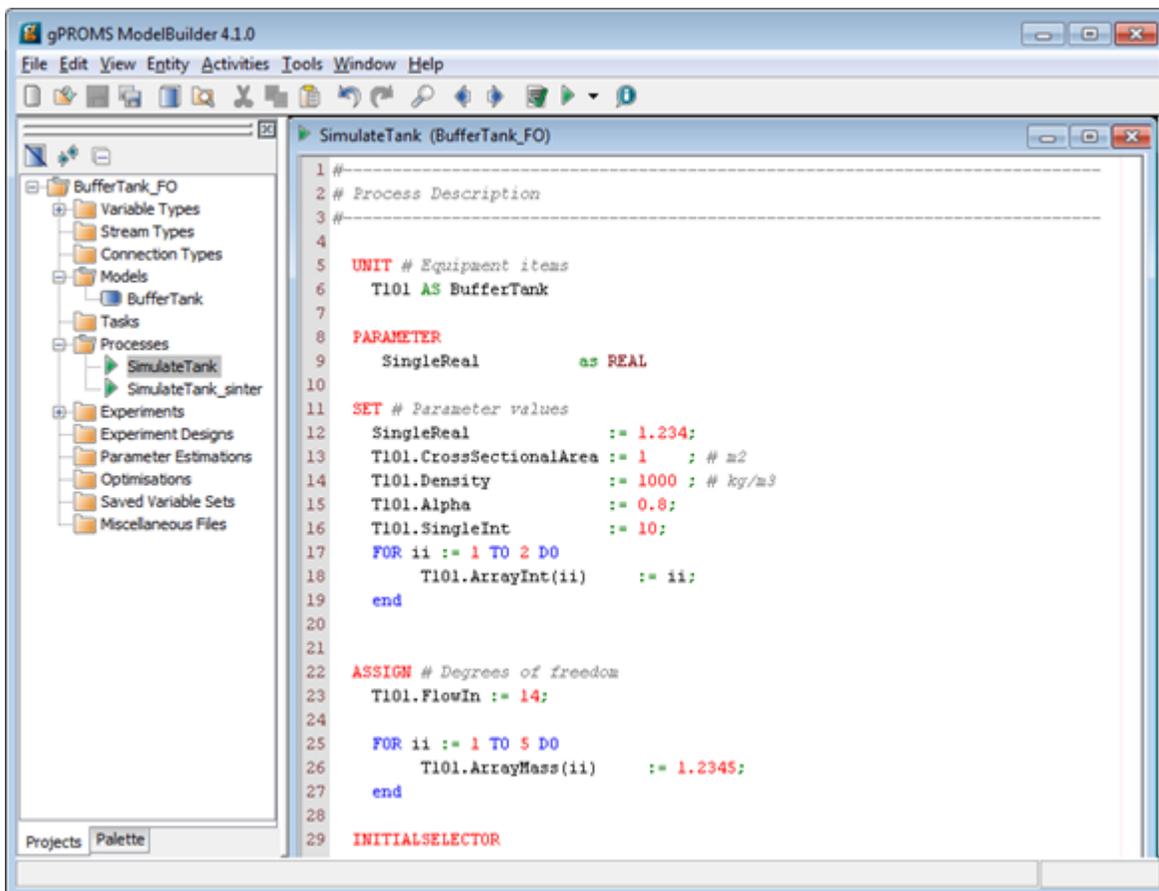


Fig. 32: Viewing SimulateTank in gPROMS Model Builder

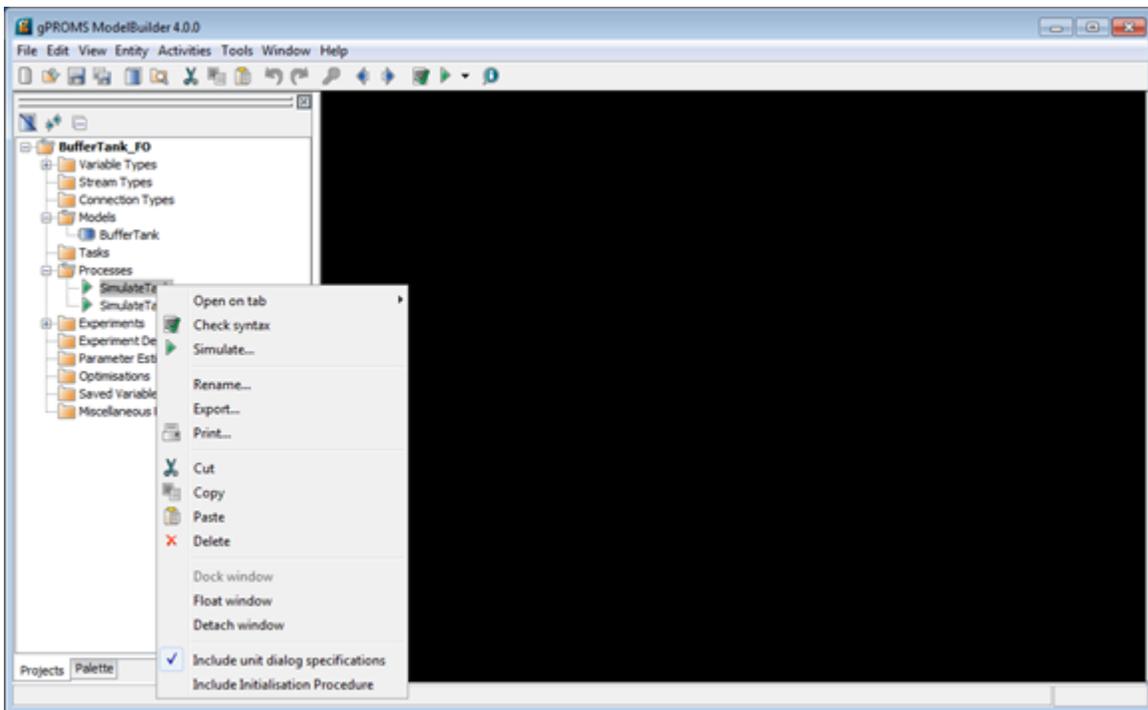


Fig. 33: Copying SimulateTank

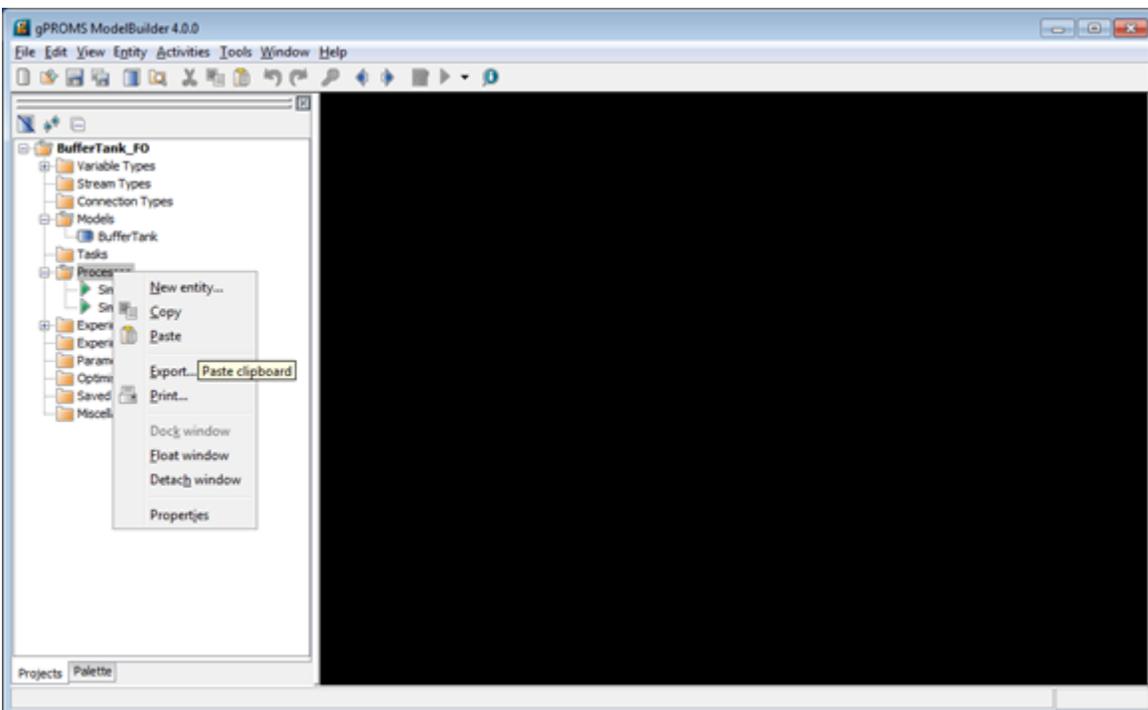


Fig. 34: Paste SimulateTank

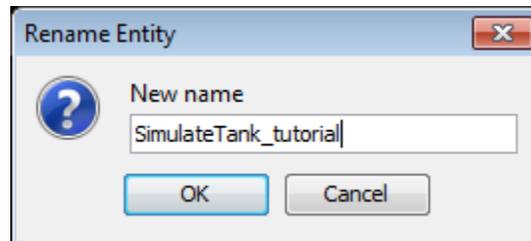


Fig. 35: Rename SimulateTank

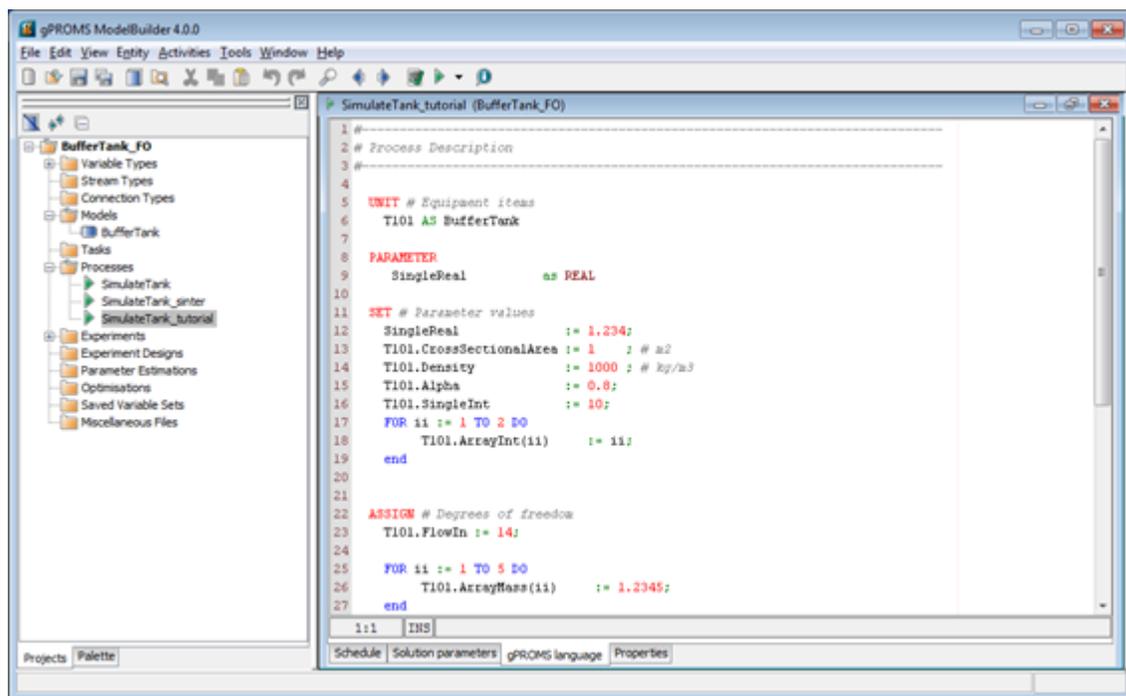


Fig. 36: Opening SimulateTank_tutorial

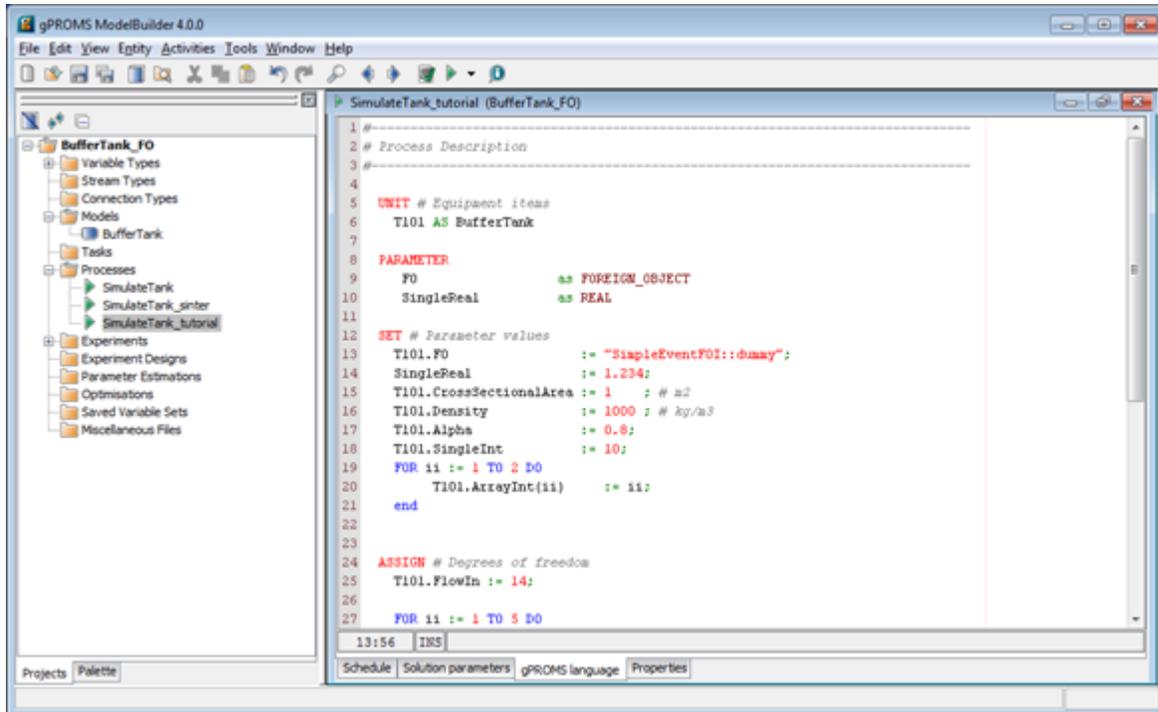


Fig. 37: Adding the FOREIGN_OBJECT

be safely ignored for this tutorial. For a full list of the methods for setting different types see the later section specifically for covering that. Any variable in the simulation can be an input, whether it is defined in the Process or one of the models referenced by the process, or in a model referenced by a model, etc. All inputs take their values from the FOREIGN_OBJECT defined, followed by the type name, two underscores, the input variable name, an open parenthesis, an optional index variable (for arrays), and closed with a close parenthesis and a semicolon. For a scalar:

```
FO.<Type>__<InputName>();
```

SimSinter can only handle arrays inputted in FOR loops such as:

```
FOR ii := 1 TO <array size> DO
  <ArrayName>(ii) := FO.<Type>1__<InputName>(ii);
END
```

For this example the user only really needs to set “T101.Alpha” in PARAMETER, “T101.FlowIn” in ASSIGN, and “T101.Height” in INITIAL.

- Now test “SimulateTank_tutorial” by selecting it and clicking the green **Simulate** triangle. When the simulation runs it will ask for every input variable the user has defined. For the example variables that do not effect the simulation, such as “SingleInt”, any valid value will work. For the values that do effect the simulation, these values work:

```
REAL__AlphaFO = .08
REAL__FlowInFO = 14
REAL__HeightFO = 7.5
```

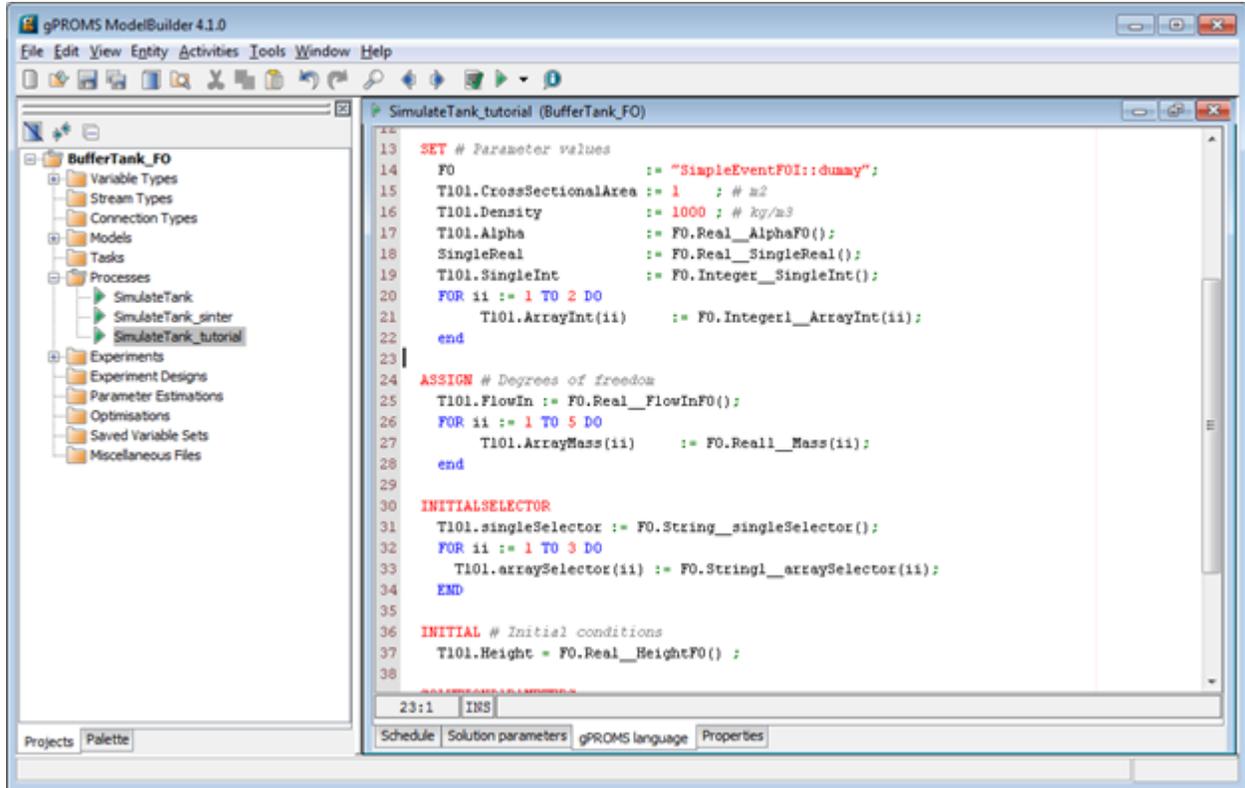


Fig. 38: Setting up Input Variables

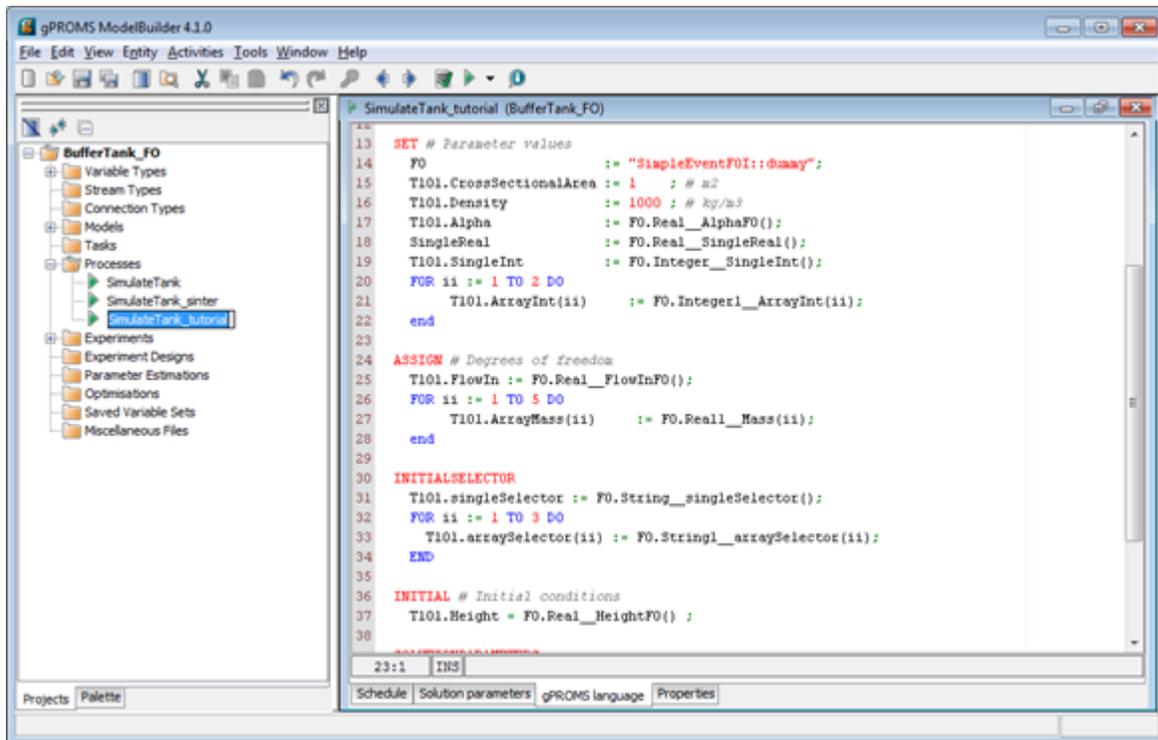


Fig. 39: Testing SimulateTank_Tutorial

Exporting an Encrypted Simulation to Run with SimSinter

SimSinter can only run encrypted gPROMS simulations. These files have the .gENCRYPT extension. If the additions to the simulation for reading input variables ran correctly in the previous section, the user is ready to export that process for use by SimSinter.

1. Right-click on the Process to export (“SimulateTank_tutorial”) and select **Export**.

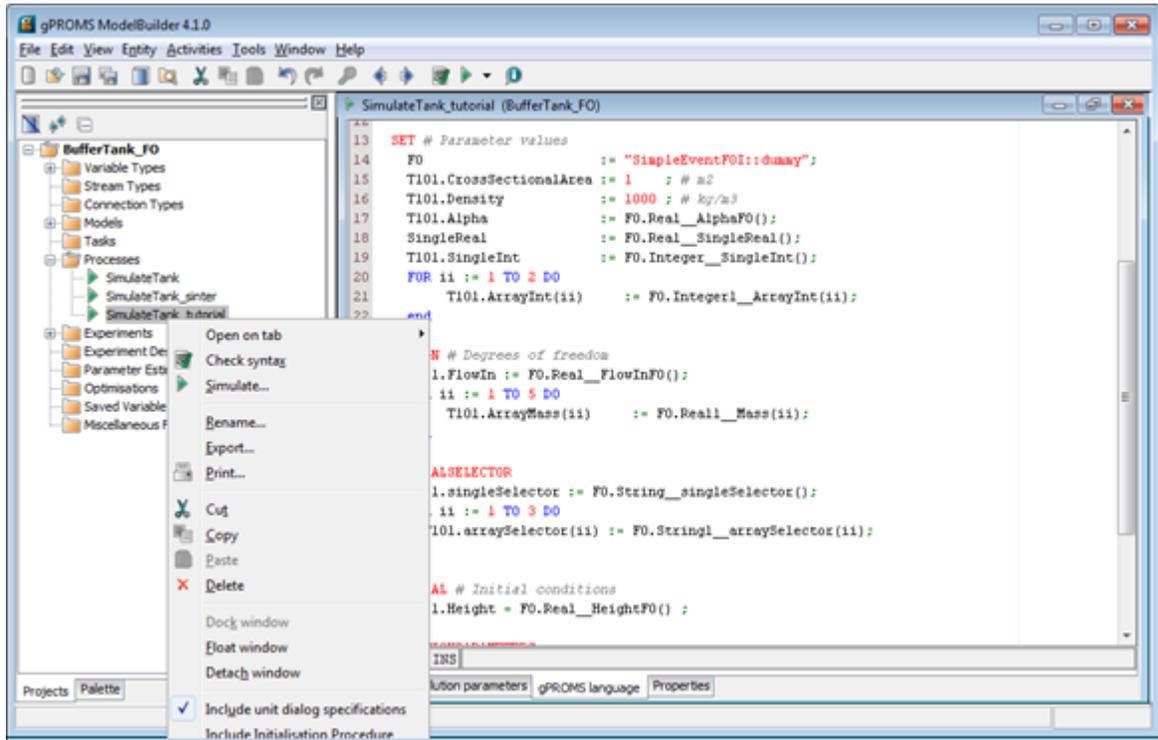


Fig. 40: Select “Export”

2. In the resulting Export window, select **Encrypted input file for simulation by gO:RUN** and click **OK**.
3. On the second page, set the **Export directory** to a directory the user can find. Preferably one without any other files in it so the user will not be confused by the output. If the filename or the **Encryption password** are not changed, SimSinter will be able to guess the password. If either of those values are changed, the user will have to set the correct password in the SinterConfigGUI password setting. A Decryption password is probably unnecessary, as the user has the original file. SimSinter does not use it. When the user has finished setting up these fields, click **Export Entity**.
4. The resulting .gENCRYPT file will be saved to a subdirectory named “Input” in the save directory specified in Step 3. The first part of the name will be identical to the .gPJ filename. The user should not rename it because the SinterConfig file will guess this name, and currently changing it requires editing the SinterConfig file.

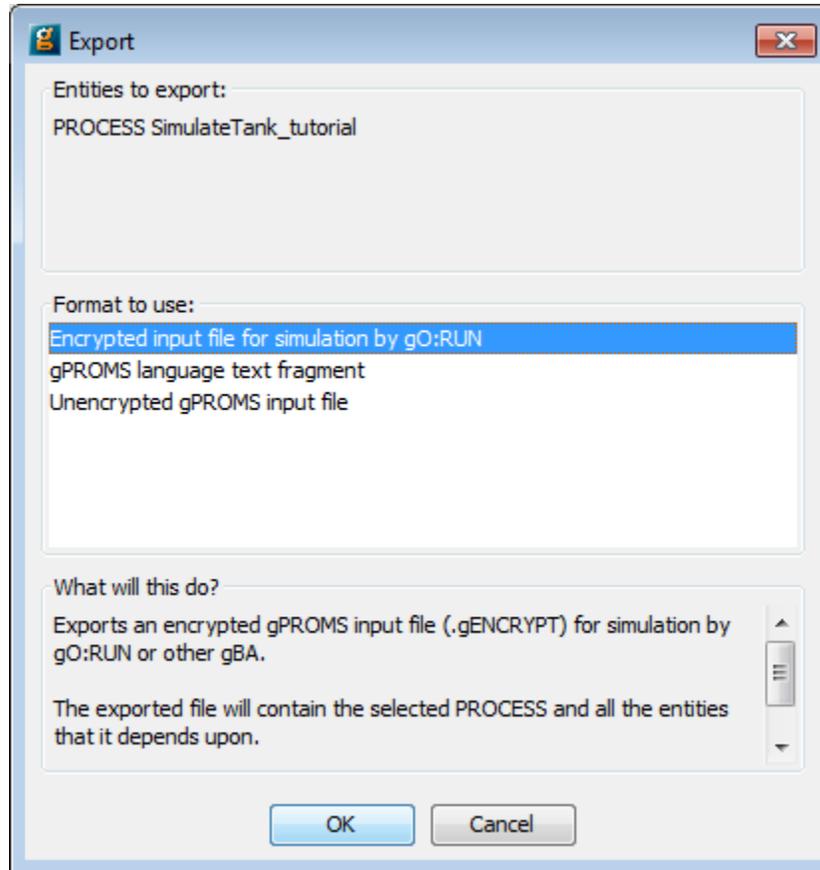


Fig. 41: Select “Encrypted Input File”

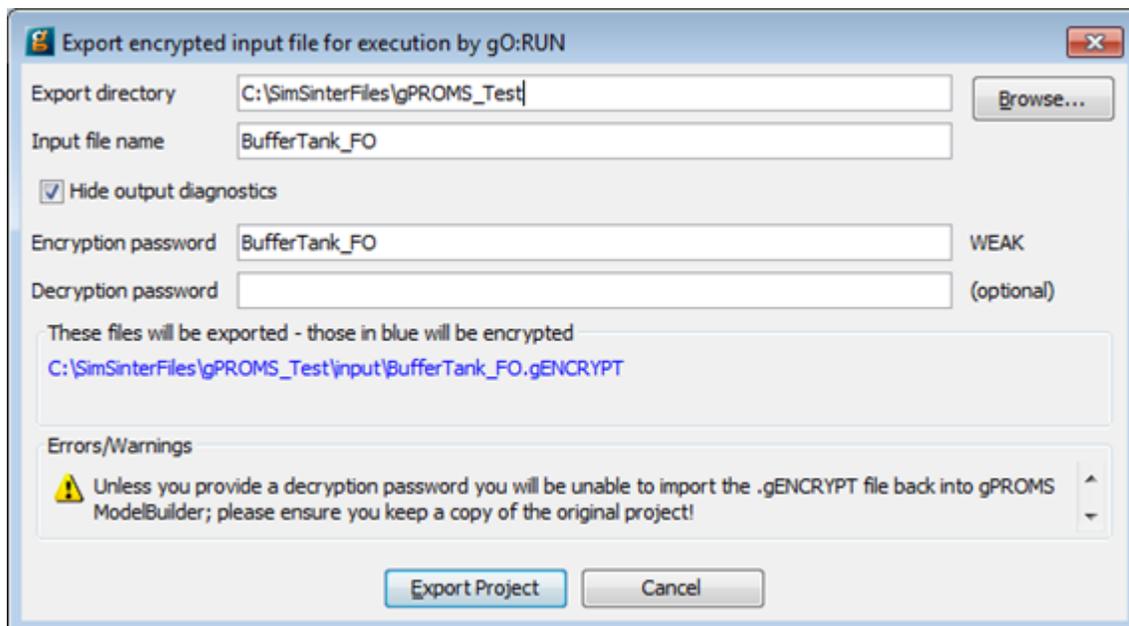


Fig. 42: Export Entity Page

Configuring SimSinter to Work with gPROMS

Now that the gPROMS process has been prepared, the SimSinter configuration can be done.

1. The “SinterConfigGUI” can be launched from FOQUS, via the **Create/Edit** button found in **File** → **Add/Update Model to Turbine** or “SinterConfigGUI” may be run on its own by selecting **CCSI Tools** → **FOQUS** → **SinterConfigGUI** from the Start menu.
2. The splash window displays, as shown in Figure *SinterConfigGUI Splash Screen*. The user may click the splash screen to proceed, or wait 10 seconds for it to close automatically.



Fig. 43: SinterConfigGUI Splash Screen

3. The SinterConfigGUI Open Simulation window displays (Figure *SinterConfigGUI Open Simulation Screen*). If “SinterConfigGUI” was opened from FOQUS, the filename text box already contains the correct file. To proceed immediately click **Open File and Configure Variables** or click **Browse** to search for the file.

This tutorial will use the .gPJ file edited in Section 1.1. Remember that SinterConfigGUI cannot read the .gEN-CRYPT file that is actually run by SimSinter. Instead, the user must open the .gPJ file the ModelBuilder uses.

Once the file is selected, click **Open File and Configure Variables**.

4. The SinterConfigGUI Simulation Meta-Data window displays as shown in (Figure *SinterConfigGUI Simulation Meta-Data Save Text Box*). Unlike the other simulations, gPROMS has not started up in any way. SinterConfigGUI does not get information from gPROMS directly, it must parse the .gPJ file instead.
5. The first and most important piece of meta-data is the **SimSinter Save Location** at the top of the window. This is where the Sinter configuration file is saved. The system suggests a file location and name. The user should confirm this is the intended location of the files to not accidentally overwrite other files. Enter the remaining fields to provide the meta-data to describe the simulation that was just opened and then click **Next**.

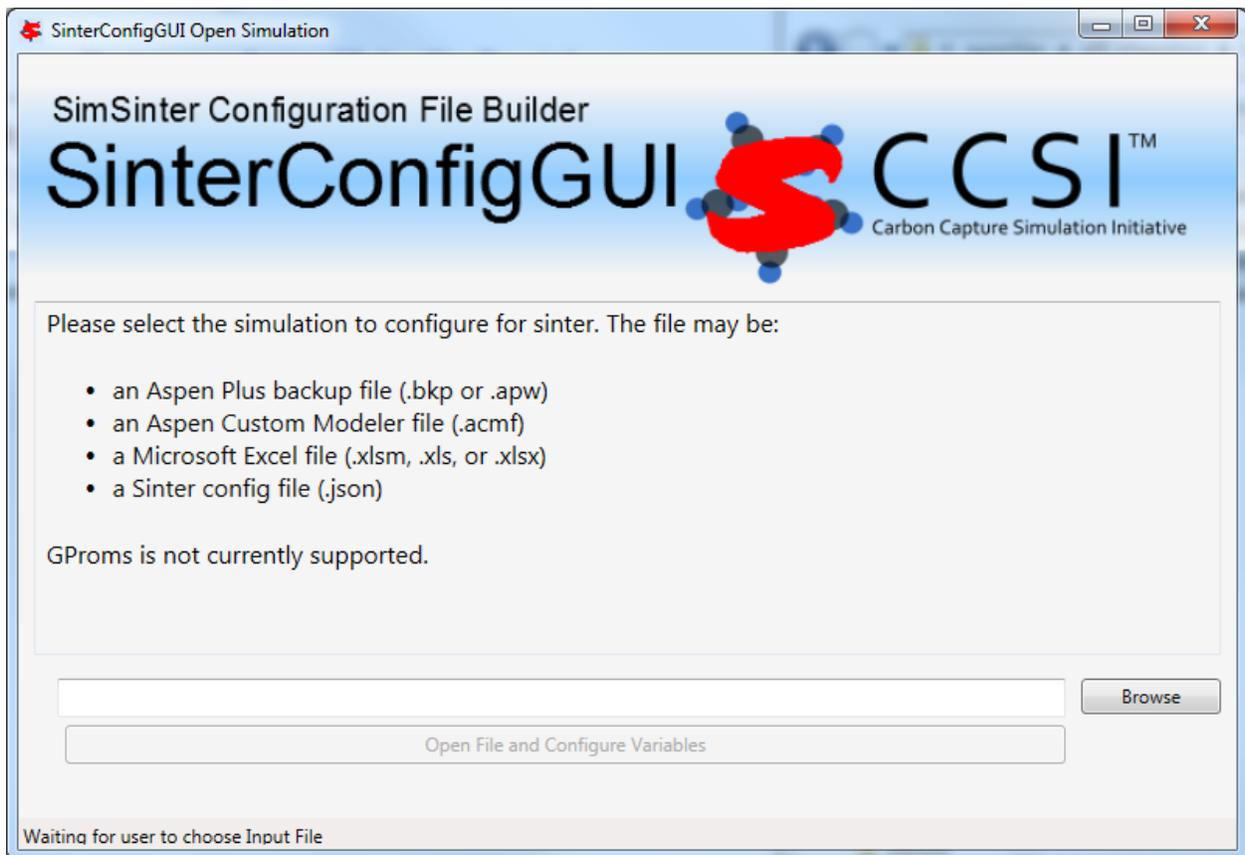


Fig. 44: SinterConfigGUI Open Simulation Screen

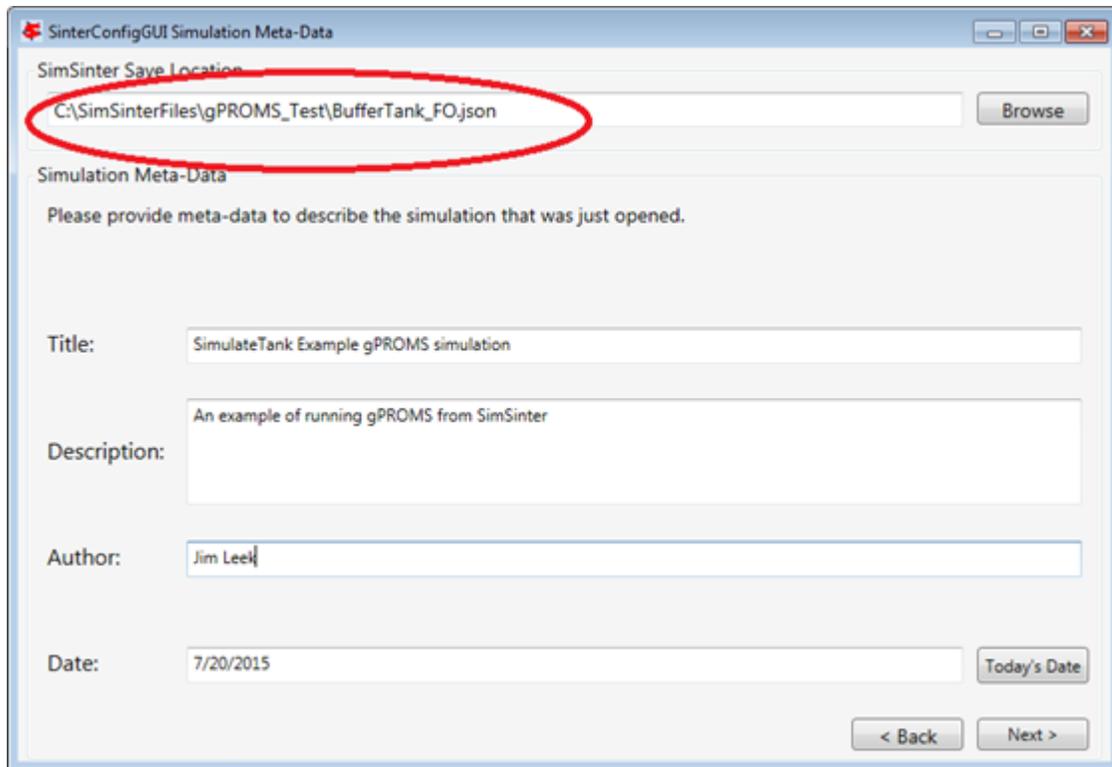


Fig. 45: SinterConfigGUI Simulation Meta-Data Save Text Box

6. The SinterConfigGUI Variable Configuration Page window displays as shown in Figure *SinterConfigGUI gPROMS Settings Configuration*. gPROMS has two settings, **ProcessName** and **password**. SimSinter has guessed at both the **ProcessName** and the **password**. For this example the **password** is correct, but the **ProcessName** is incorrect. SimulateTank is the process that isn't configured to work with SimSinter. On the left side we can see the **Variable Tree**. The root is connected to the three processes defined in this .gPJ file. First, change the **ProcessName** to "SimulateTank_tutorial".
7. After changing the **ProcessName**, click Enter (or clicks away). The **Selected Input Variables** will automatically display all of the available input variables. This is because the input variables have been configured in gPROMS, and SimSinter has parsed them out of the .gPJ file, as long as you have the **ProcessName** set correctly. This also means that the user cannot add new input variables in SinterConfigGUI, only in gPROMS. SimSinter also does its best to identify the **Default** values, **Min**, and **Max** of the variables. The default can only be calculated from the file if it was defined purely in terms of actual numbers. SimSinter cannot evaluate other variables or functions. Therefore,

```
DEFAULT 2 * 3.1415 * 12
```

will work. However,

```
DEFAULT 2 * PI * radius
```

will not work, because SimSinter does not know the value of either PI or radius, and SimSinter will just set the default to 0.

Min and **Max** values are taken from the variable type, if there is one.

8. Now the output values can be entered. Expand the "SimulateTank_tutorial" Process on the Variable Tree, expand the "T101" model, and then double-click on "FlowOut" to make it the Preview Variable. Notice that the **Make**

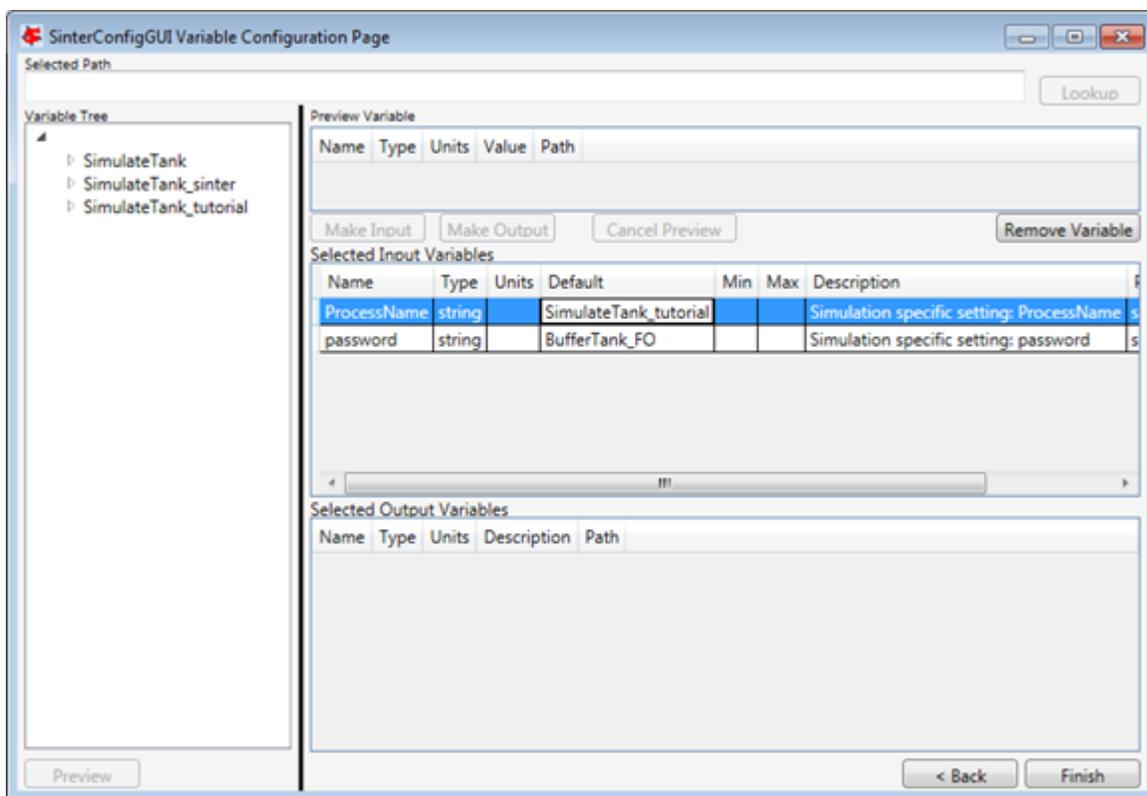


Fig. 46: SinterConfigGUI gPROMS Settings Configuration

Input button is disabled. As stated above, the user cannot make new Input Variables in SinterConfigGUI. Only **Make Output** is allowed.

9. If **Make Output** is clicked, “FlowOut” will be made an Output Variable as shown in Figure *FlowOut as an Input Variable*. The Description can be updated, but SimSinter made a good guess in this example; therefore, there is no need to change the description.
10. By the same method, make Output Variables “HoldUp” and “Height.”
11. The variables names should be made shorter. Simply click on the **Name** column and change the name to your preferred name.
12. For future testing, make sure the defaults are good values. The only three input variables that matter have the following defaults:

```
AlphaFO = 0.8
FlowInFO = 14
HeightFO = 7.5
```

13. When finished making output variables, click **Next** at the bottom of the variables page. If there were any input vectors, the Vector Default Initialization page will display. Here the default values of the vectors may be edited.
14. Finally, click **Finish** and save your configuration file. Your gPROMS simulation should now be runnable from FOQUS.

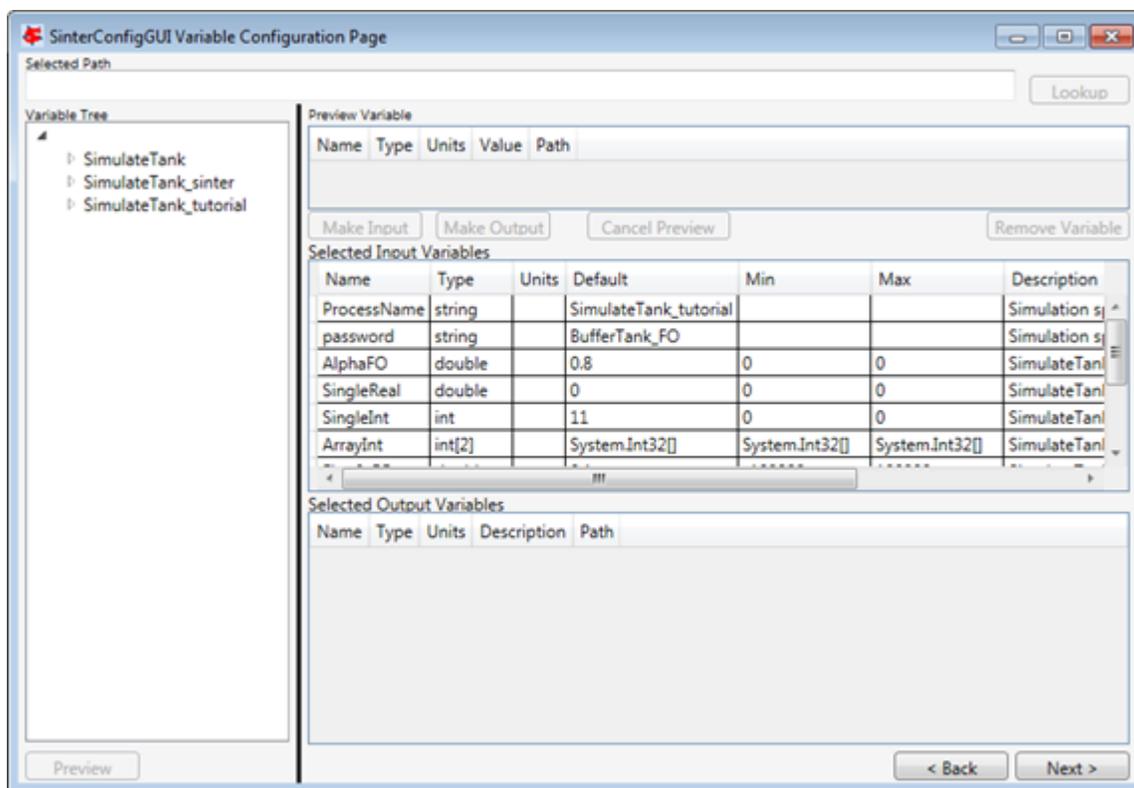


Fig. 47: SinterConfigGUI Automatically Displays Input Variables

14.1.3 Additional ACM Functionality with Excel/VBA

When additional functionality is needed when working with Aspen Custom Modeler (ACM), such as custom initialization routines or executing complicated cyclic models, using excel/VBA execution of the ACM simulation is a viable option. The excel file can then be connected to FOQUS through the documented SimSinter connection.

Below is an example of a VBA macro titled 'runAspen' for executing a temperature swing adsorption (TSA) cycle simulated in ACM.

Code block 1 creates the ACM object using objects installed with Aspen simulation workbook. The full path of the ACM file is needed in the 'ACMApp.OpenDocument' function. The 'ACMSimulation' object is used to read/write variables in the simulation and even run the simulation. Additionally, you can view the aspen simulation by setting 'ACMApp.Visible' to True.

Code block 2 changes the value of a fixed variable in the simulation through the 'ACMSimulation' object. This can be a hardcoded value in the macro script, or it can read from a worksheet cell which is shown here.

Code block 3 runs the custom initialization and simulation routine for this TSA example. The model is first run in initialization mode and then to ensure that the cycle task will correctly run, it is deactivated and then reactivated. The model is again initialized and then run in dynamic mode.

Code block 4 reads results from the model and then closes the simulation.

This is a simple example, but additional examples and more information for scripting and automation can be found in the documentation for ACM. To run the model using FOQUS, connect the excel file which contains the execution macro using SimSinter, and create a flowsheet node for the resulting turbine model. Tutorials for Excel/SimSinter and creating a flowsheet are linked below.

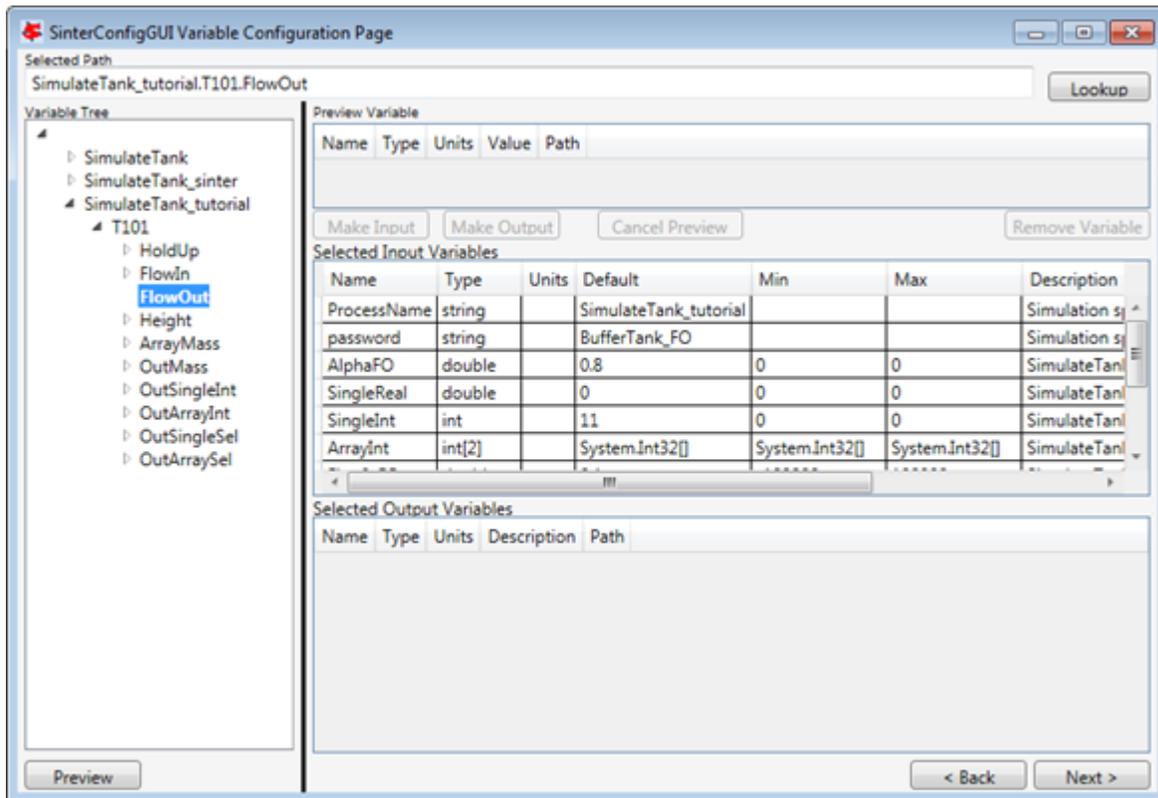


Fig. 48: Preview of the FlowOut Variable

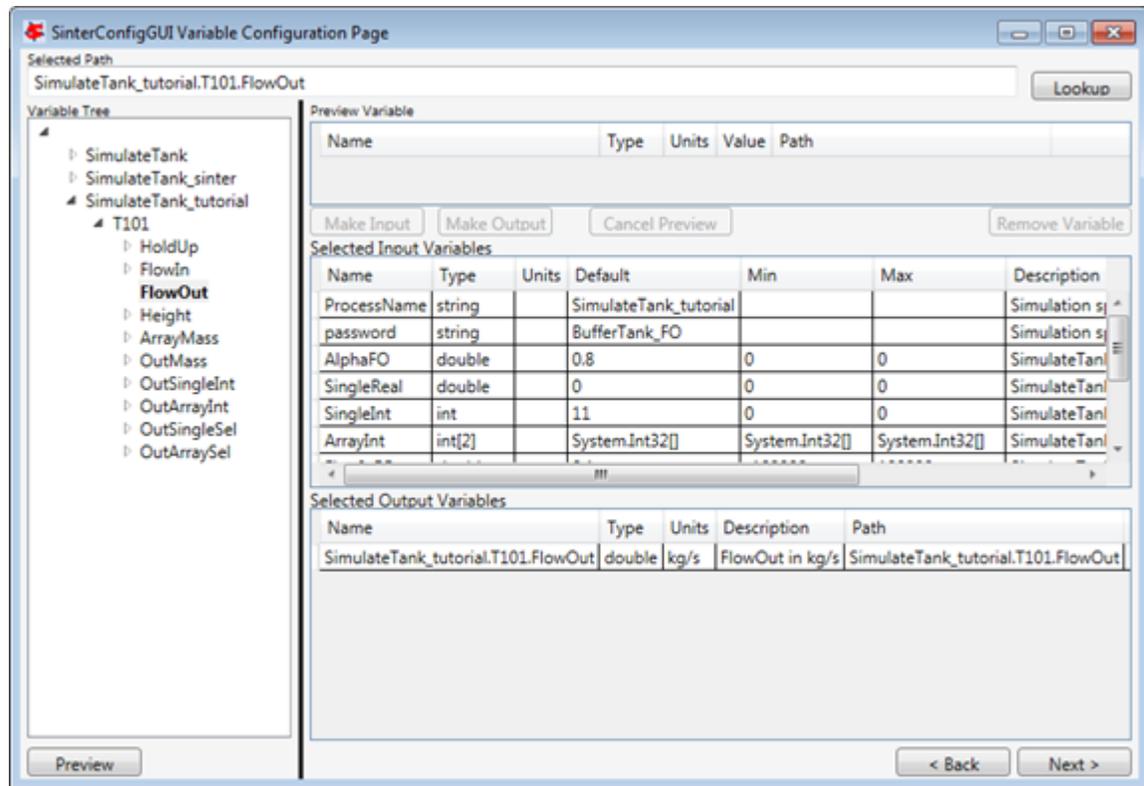


Fig. 49: FlowOut as an Input Variable

https://foqus.readthedocs.io/en/stable/chapt_sinter/tutorial/excel.html

https://foqus.readthedocs.io/en/stable/chapt_flowsheet/tutorial/sim_flowsheet.html

Specifying that the macro should be run can be done during the SimSinter setup step or in the settings tab after the flowsheet node has been created. The node settings for our example setup are shown below in which the 'runAspen' macro is specified.

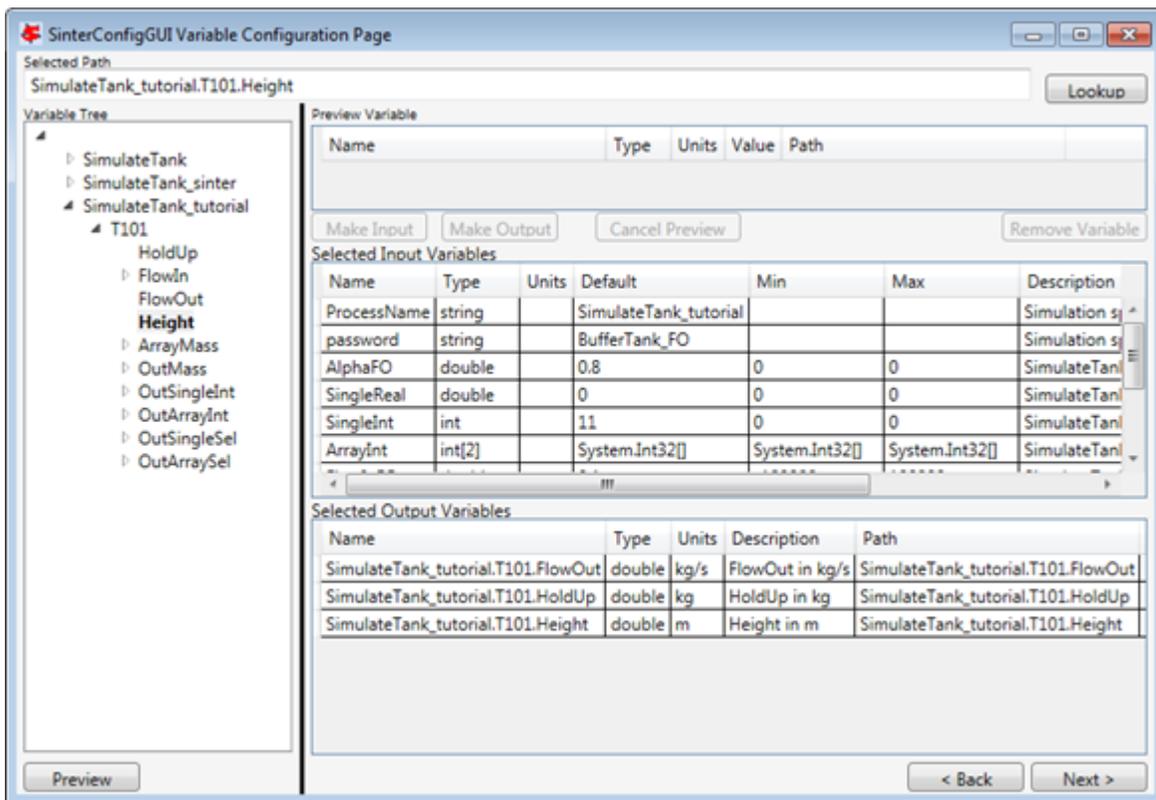


Fig. 50: HoldUp and Height Output Variables

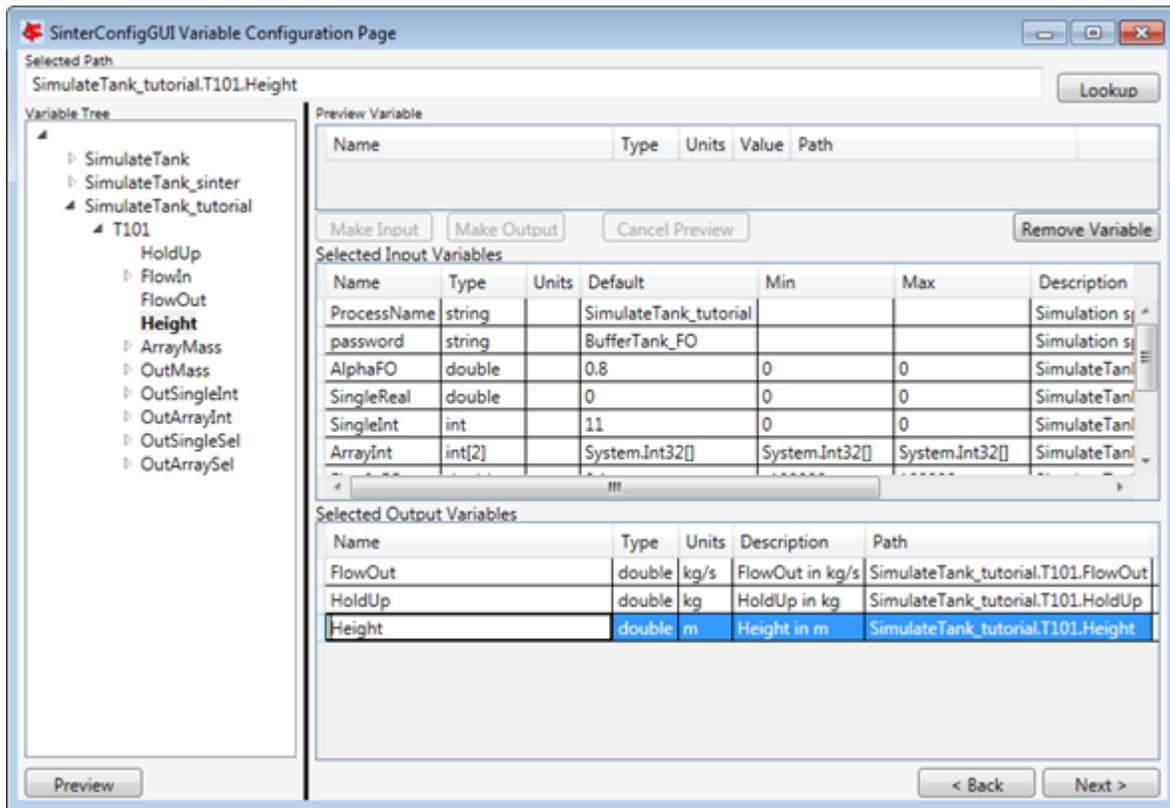


Fig. 51: Editing Variable Names

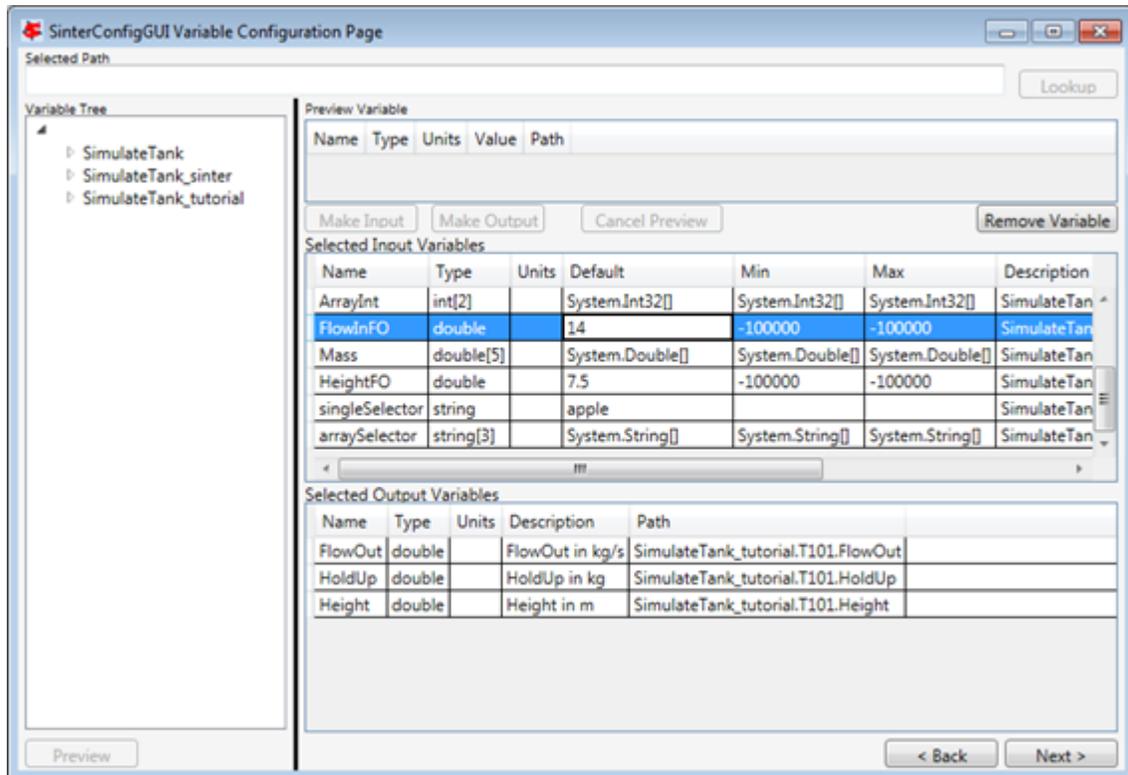


Fig. 52: Editing Defaults

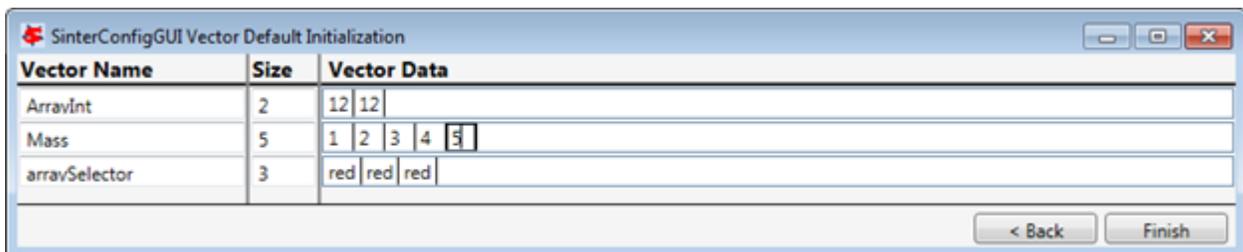


Fig. 53: Editing Vectors

```

Sub runAspen()

    Application.DisplayAlerts = False

    'creating Aspen adsorption object
    Dim ACMApp As Object
    Dim ACMDocument As Object
    Dim ACMSimulation As Object
1   Set ACMApp = CreateObject("ACM Application")
    ACMApp.Visible = False

    'opening simulation file
    Set ACMDocument = ACMApp.OpenDocument(ActiveWorkbook.Path & "\ & "YourModelName.acmf")
    Set ACMSimulation = ACMApp.Simulation

2   'writing simulation inputs
    ACMSimulation.Flowsheet.LL target.Value = Range("A1").Value

    'initializing model
    ACMSimulation.RunMode = "Initialization"
    ACMSimulation.Run (True)

3   ' resetting cycle and initializing again
    ACMSimulation.Flowsheet.Cycle1.Active = False
    ACMSimulation.Flowsheet.Cycle1.Active = True
    ACMSimulation.Run (True)

    'running in dynamic mode
    ACMSimulation.RunMode = "Dynamic"
    ACMSimulation.Run (True)

4   'reading results from simulation
    Range("B1").Value = ACMSimulation.Flowsheet.integralcapture.Value

    'closing simulation
    ACMApp.Quit

End Sub

```

Fig. 54: VBA Example Code

Node Edit

Apply
 Revert

Variables Position Node Script

Name: TSA Visible

Error Status

Code: 0

Message: Finished Normally

Model

Type: Turbine Model: ba_cycle

Input Variables

Output Variables

Settings

	Name	Value	Description
8	Maximum Wait Time (s)	1440.0	This is the ammount of time in seconds that FOQUS should wait for results to come back from Turbine.
9	Maximum Run Time (s)	840.0	This is the ammount of time in seconds that FOQUS should wait for results to come back from Turbine once the simulation s
10	Min Status Check Interval	4.0	This is the minimum amount of time between job status checks.
11	Max Status Check Interval	5.0	This is the maximum ammount of time between job status
12	Override Turbine Configuration	--	Optional, provide a path to a Turbine config to submit models for this node to a alternative Turbine gateway. This can be use
13	macro	"runAspen"	Simulation specific setting: macro

Fig. 55: Node Settings for Running Excel Macro

SURROGATE MODEL BASED OPTIMIZER

15.1 Contents

15.1.1 Surrogate model-based optimizer - overview

Introduction

As part of the improvements and new capabilities of FOQUS, the Surrogate Model-based Optimizer is an automated framework for hybrid simulation-based and mathematical optimization. The SM-based Optimizer in FOQUS leverages the direct link with commercial simulators, generation of surrogate models, access to algebraic modeling systems for optimization, and implements a modified trust region approach for the optimization of advanced process systems.

The motivation behind developing this framework was to combine the advantages of both, simulation based, and pure mathematical optimization. Pure mathematical optimization directly leverages the equations describing the physical system to be optimized. Such models are the most accurate and complete representation of the system, and thus provide the most accurate optimization results. This approach encounters challenges, however, when large sets of PDE's and highly complex, nonlinear representations are required to sufficiently characterize the process of interest. The mathematical model can then become intractable. Simulation-based optimization, on the other hand, considers the system model to be a black box and is based on a heuristic algorithm that uses the results from process simulations to obtain the relationship between the relevant system input and output variables. Although this approach can be used to obtain satisfactory results for large scale, complex systems, it can often be computationally expensive and hence, time consuming, due to multiple simulation runs.

The SM-based optimization algorithm involves generating a simplified representation of the rigorous process model (i.e. built using advanced commercial simulators like ASPEN, gPROMs, Python, etc.) via surrogate models that are more amenable to gradient-based optimization methods and nonlinear programming (NLP) solvers. This approach can overcome the difficulties associated with complex process models in terms of intractability and multiple evaluation requirement, without significantly compromising solution quality and speed, provided that the surrogate modeling method is accurate.

Additional	python	packages	required
-------------------	---------------	-----------------	-----------------

1. Surrogate modeling toolbox - smt: pip install smt
2. Experimental design package for python - pyDOE: pip install pyDOE
3. Pyomo package for optimization: pip install pyomo
4. Mathematical optimization solver ipopt: conda install -c conda-forge ipopt (preferred installation method for Windows users)

Note: smt package is required to access its Latin hypercube sampling method, which is required to generate samples and re-build surrogate models in each iteration of the algorithm. pyDOE package is a requirement within the smt package, which makes its installation necessary.

Framework

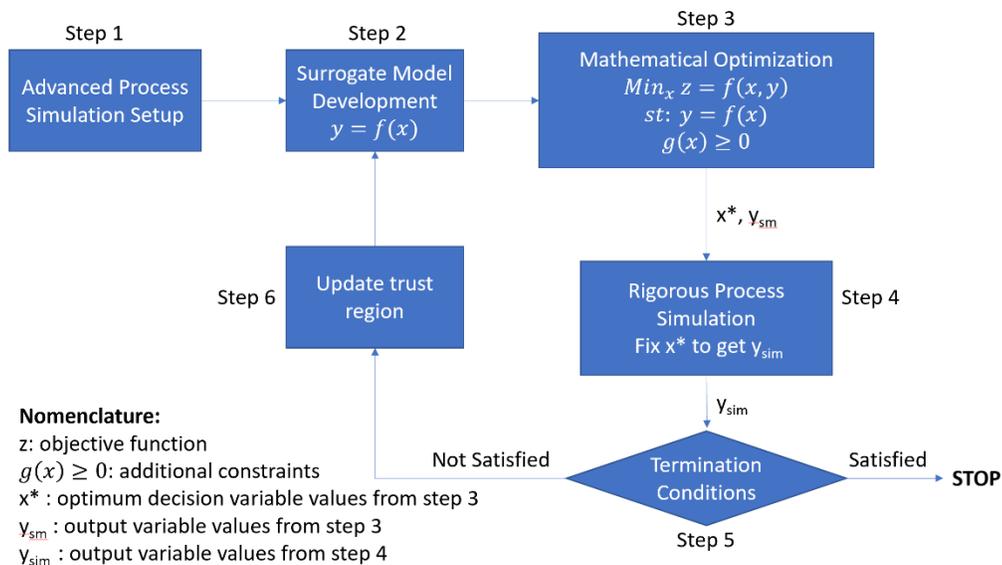


Fig. 1: Figure 1 - Framework for surrogate model-based optimization algorithm

As shown in figure 1, the framework consists of 6 main steps, in which the first 2 steps require the user interaction, while the rest of the algorithm will be performed automatically. The detailed description of each Step is provided here:

Step 1 – Flowsheet set up: First, the user must provide a rigorous process simulation to the FOQUS flowsheet, then select the input and output variables of interest. Once, the simulation node has been tested and the user provided input variables with their default values, upper and lower bounds, the user needs to generate simulation samples using the UQ module in FOQUS for a given input space. At this point, the upper and lower variable bounds will be considered as the initial trust region, and the samples will be used to develop the initial surrogate model.

Step 2 – Surrogate Model Development: This step is simple, but critical to minimize the number of iterations required in the algorithm. The user must select the number of samples, and alamo settings to generate the best surrogate possible. Finally, The user generates a surrogate model based on the simulation samples using FOQUS-ALAMO module.

Step 3 – Mathematical Optimization: In the Optimization module, setup the problem by selecting the decision variables, providing the objective function, and additional constraints. Since, FOQUS Optimization module allows multiple derivative free optimizers (DFO), user must select the surrogate model-based optimizer as the solver, with appropriate settings for the algorithm (detailed description of the settings is provided in the tutorial). The SM-based

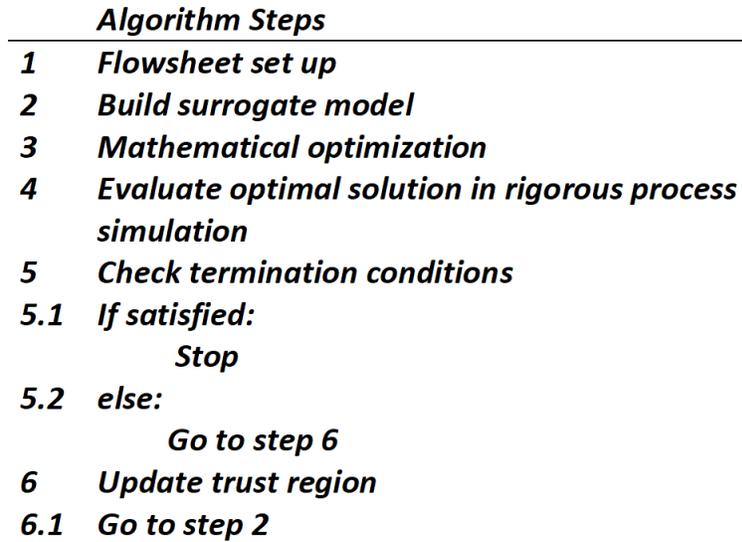


Fig. 2: Algorithm Steps

optimizer formulates and solves the optimization problem by creating a Pyomo model (Concrete Model), adding the input and output variables (as Pyomo variables with bounds – trust region), adding the surrogate models as Pyomo constraints, and adding additional constraints provided by the user ($g(x)>0$ or $h(x)=0$). In this step, to avoid eliminating feasible solutions due to local optimums, a multi-start approach has been implemented, in which the optimization problem is solved for different initialization points. A combination of initial values is used based on the variable bounds, mid-point, and user provided values for the decision variables. The optimal solution chosen corresponds to that case which gives best value of objective function (minimum or maximum). Note, if a solution returns infeasible it will be eliminated. The solution is called x^* and ysm , for optimum decision variables and output variables, respectively.

Step 4 – Rigorous Process Simulation: In this step, the process simulation is run at the optimal point obtained in step 3 (x^*), then evaluating the optimal solution using the rigorous model, we obtain the corresponding output variable values $ysim$.

Step 5 – Termination Condition Check: The algorithm includes three termination conditions to determine if the optimal solution has been obtained:

$$\frac{|z_{sim} - z_{sm}|}{|z_{sim}|} \leq \epsilon \dots (1)$$

$$\frac{|y_{sim} - y_{sm}|}{|y_{sim}|} \leq \epsilon \dots (2)$$

$$g(x^*) \geq 0 \dots (3)$$

First, Equation 1 checks if the objective function from the surrogate model (zsm) minus the one obtained evaluating the rigorous model ($zsim$) meet the tolerance. Secondly, the relative error between the output variables from the optimization problem (ysm) and the rigorous simulation ($ysim$) in Equation 2. Finally, Equation 3 checks that the additional constraint is satisfied at the optimum point. If the conditions in step 5 are satisfied, the algorithm is terminated, otherwise, step 6 is implemented.

Step 6 – Update Trust Region: In this step, the input variable upper and lower bounds (xub and $xlub$) are adjusted to shrink the trust region. The extent to which the trust region shrinks ($difk$) depends on the fractional multiplier α . The updated upper and lower bounds ($xub,k+1$ and $xlub,k+1$) are calculated around x^* , based on $difk$:

$$0 \leq \alpha \leq 1$$

$$difi_k = (x_{ub,k} - x_{lb,k}) * \alpha$$

$$x_{lb,k+1} = x^* - \frac{difi_k}{2} \dots (x_{lb,k+1} = x_{lb,k=0} \dots difi, x_{lb,k+1} < x_{lb,k=0})$$

$$x_{ub,k+1} = x^* + \frac{difi_k}{2} \dots (x_{ub,k+1} = x_{ub,k=0} \dots difi, x_{ub,k+1} > x_{ub,k=0})$$

Note that if the ratio of upper and lower bounds is less than or equal to a set value of bound ratio, the trust region is not updated further, and the algorithm terminates.

If

$$\frac{x_{ub,k+1}}{x_{lb,k+1}} \leq boundratio$$

Stop

Further, Latin hypercube samples are generated in the updated trust region. This sampling method ensures that the sample points are uniformly spaced out and cover the entire trust region without any skewness. Once the samples are generated, step 2 is repeated using this new data set and the original ALAMO settings.

15.1.2 Surrogate model-based optimizer - tutorial

Flash Optimization

Problem Statement: An Ethanol-CO₂ mixture at 50 mol %, enters a flash column at 100 kg/hr, 25 OC and 100 bars. The optimum flash column pressure needs to be determined such that maximum revenue can be obtained based on the CO₂ obtained in the vapor stream, and Ethanol obtained in the liquid stream. The optimization is subject to a purity constraint, specifying that the CO₂ mass % in the vapor phase should be at least 98.5 %. The system is shown in Figure 1.

Instructions

Step 1 - Flowsheet Setup

Step 1.1 - Setup the Aspen model for flash column as FOQUS simulation node : To setup the Aspen model in the FOQUS flowsheet, first, create and add the SimSinter json file to turbine. Then, create a node named 'FLASH', and load the simulation in the node. The Aspen and json files (along with the FOQUS file) can be found in the folder: `examples/tutorial_files/SM_Optimizer/Flash_Optimization`.

Note: The `examples/` directory refers to the location where the FOQUS examples were installed, as described in *Install FOQUS Examples*.

Figures 2 and 3 represent the FOQUS node with loaded simulation. Finally, run the flowsheet simulation.

Step 1.2 - Generate a simulation ensemble by selecting 'FLASH.PRES' as a variable with bounds 1-10 bar (in this case, keep the other variables fixed). Select Latin Hypercube Sampling with 20 points, and after the samples are generated, launch the simulations. Figure 4 represents the simulation ensemble generation.

For more details on this, refer to the documentation: https://foqus.readthedocs.io/en/latest/chapt_uq/tutorial/sim.html

Step 2 - Surrogate Model Development

Step 2.1 - Select Data Set: In the surrogate modeling module, select ALAMO as the tool and under 'Data' tab, ensure that the dataset corresponds to the correct UQ Simulation Ensemble. If there are multiple data sets, add filters to select the appropriate set. Figure 5 represents the data selection in the surrogates tab.

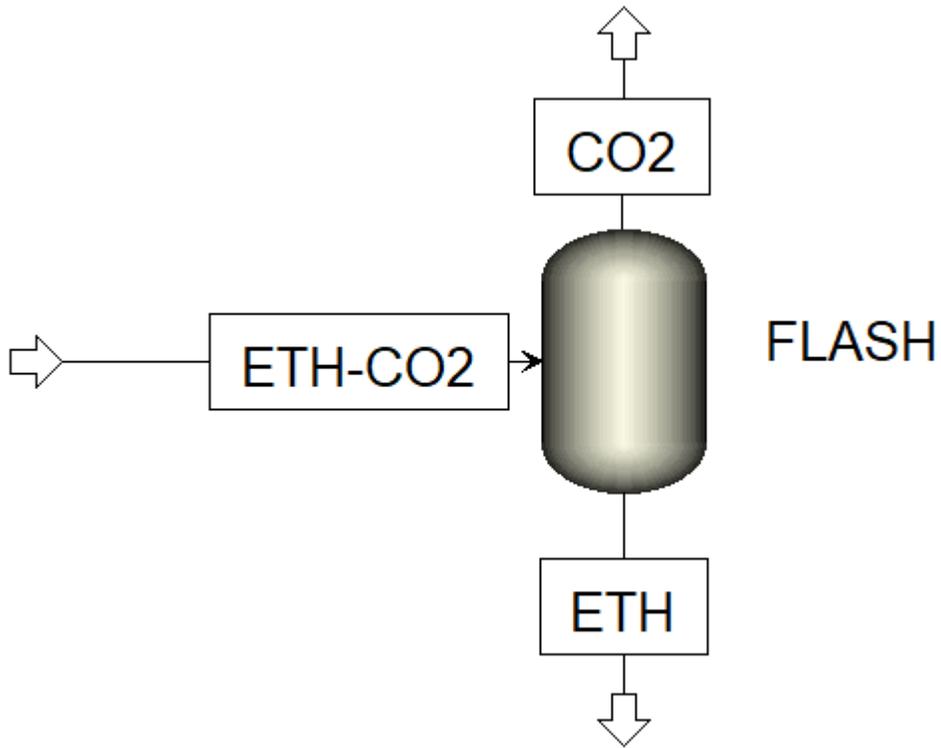


Fig. 3: Figure 1: Ethanol-CO2 Flash System

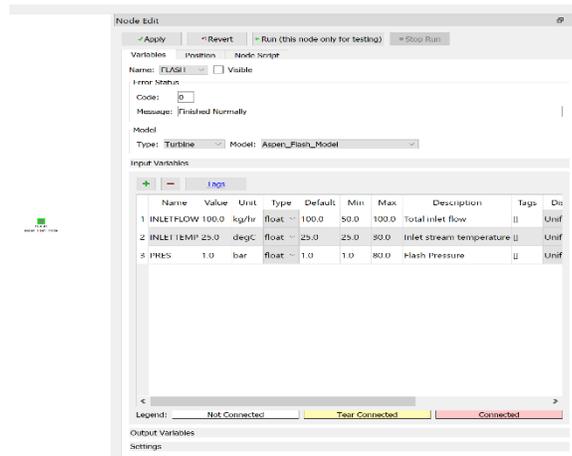


Fig. 4: Figure 2: Input variables of the Ethanol-CO2 Flash Simulation Node in FOQUS

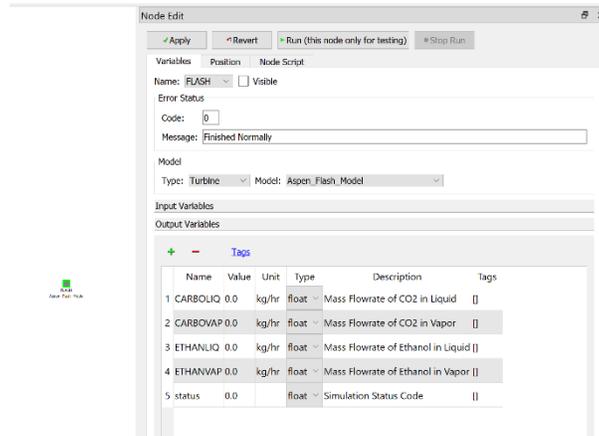


Fig. 5: Figure 3: Output variables of the Ethanol-CO2 Flash Simulation Node in FOQUS

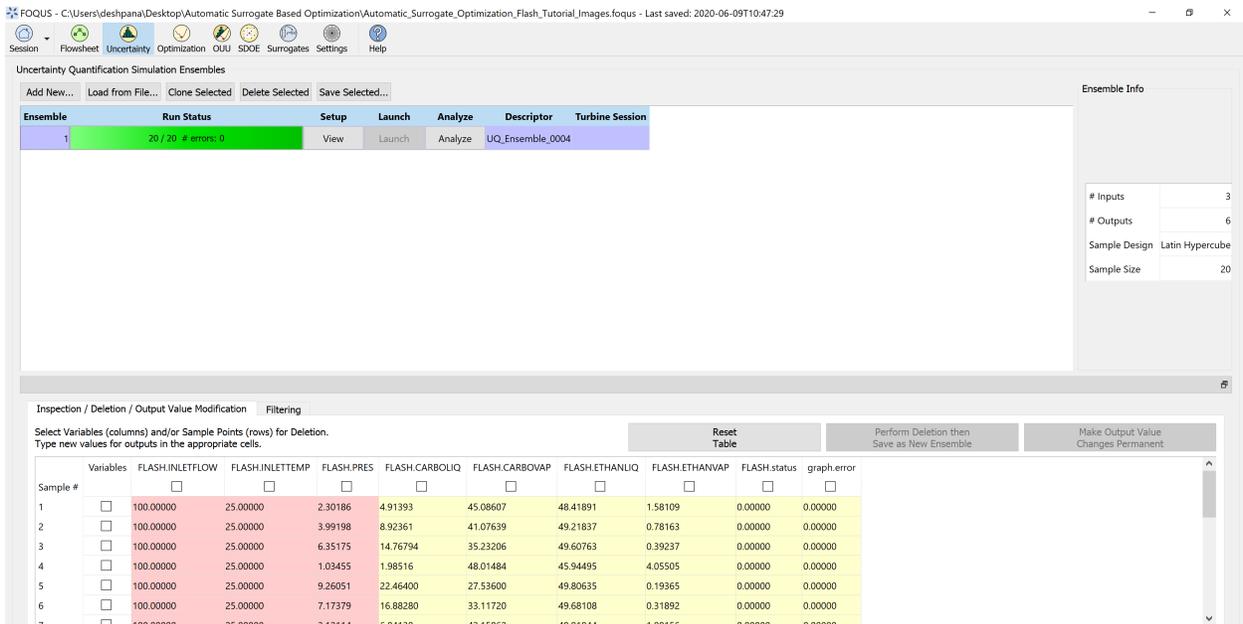


Fig. 6: Figure 4: Simulation ensemble generation

Method Description | Data | Variables | Method Settings | Execution

Tool: ALAMO

Method Description | Data | Variables | Method Settings | Execution

Add Samples...

Flowsheet Results

Menu | Current Filter: uq2 | Add/Edit Filters | Number of Rows: 20

	set	result	time	solution_time	err	input.FLASH.INLETFLOW	ut.FLASH.INLETTE	input.FLASH.PRES	output.graph.ero	ut.FLASH.ETHAN	ut.FLASH.CARBO	put.FLASH.ETHAN	ut.FLASH.C
0	"UQ_Ensemble_0004"	"uq_000000"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	2.000000	0.0	1.58109353	45.0860679	48.4189065	4.91393213
1	"UQ_Ensemble_0004"	"uq_000001"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	3.000000	0.0	0.781631779	41.0763855	49.2183682	8.92361446
2	"UQ_Ensemble_0004"	"uq_000002"	"2019-12-25T21:00:00.000000"	9.000000	0.0	100.0	25.0	6.000000	0.0	0.392373696	35.2320561	49.6076263	14.7679439
3	"UQ_Ensemble_0004"	"uq_000003"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	1.000000	0.0	4.05504969	48.0148352	45.9449503	1.98516477
4	"UQ_Ensemble_0004"	"uq_000004"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	9.2605093965123	0.0	0.193647919	27.5359972	49.8063521	22.4640028
5	"UQ_Ensemble_0004"	"uq_000005"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	7.000000	0.0	0.31892271	33.1172031	49.6810773	16.8827969
6	"UQ_Ensemble_0004"	"uq_000006"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	3.000000	0.0	1.08156013	43.1586218	48.9184399	6.84137819
7	"UQ_Ensemble_0004"	"uq_000007"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	8.000000	0.0	0.218753981	28.9476907	49.781246	21.0523093
8	"UQ_Ensemble_0004"	"uq_000008"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	5.000000	0.0	0.562540954	38.5279808	49.437459	11.4720192
9	"UQ_Ensemble_0004"	"uq_000009"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	2.000000	0.0	1.31616915	44.2194309	48.6838308	5.78056912
10	"UQ_Ensemble_0004"	"uq_000010"	"2019-12-25T21:00:00.000000"	9.000000	0.0	100.0	25.0	8.000000	0.0	0.237789895	29.89946	49.7622101	20.10054
11	"UQ_Ensemble_0004"	"uq_000011"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	4.000000	0.0	0.671495395	39.9554316	49.3285046	10.0445684
12	"UQ_Ensemble_0004"	"uq_000012"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	7.000000	0.0	0.293260696	32.2216702	49.7067393	17.7783298
13	"UQ_Ensemble_0004"	"uq_000013"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	6.000000	0.0	0.384431731	35.0300478	49.6155683	14.9695922
14	"UQ_Ensemble_0004"	"uq_000014"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	3.000000	0.0	1.01430418	42.7801612	48.9856958	7.21983881
15	"UQ_Ensemble_0004"	"uq_000015"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	1.000000	0.0	2.41701089	46.6849346	47.5829891	3.31506539
16	"UQ_Ensemble_0004"	"uq_000016"	"2019-12-25T21:00:00.000000"	5.000000	0.0	100.0	25.0	9.000000	0.0	0.179576051	26.6529098	49.8204239	23.3470902

Fig. 7: Figure 5: Select data for surrogate model generation

Note: If a particular simulation ensemble needs to be used from the UQ module for generating the surrogate model, add a data filter, referring to the instructions in the documentation: https://foqus.readthedocs.io/en/latest/chapt_uq/tutorial/data.html

Step 2.2 - ALAMO input/output variables: Under 'Variables' tab, select 'FLASH.PRES' as the surrogate model input variable, and 'FLASH.CARBOLIQ', 'FLASH.CARBOVAP', 'FLASH.ETHANLIQ', 'FLASH.ETHANVAP' as the surrogate model output variables. Figure 6 represents surrogate model variables selection.

Step 2.3 - ALAMO Settings: Under 'Method Settings', to select the data set to be used to develop the surrogate models, an Initial Data Filter can be applied to the full data set, if there are no filters, simply select "all". In this case, we select "uq2" filter. In Figures 7 and 8, the settings 3 to 9 values are default in FOQUS. The settings 10 to 22 have been selected to explore several basis functions and obtain the best model possible, while minimizing the size of the model (selecting Bayesian Inference Criteria as the modeler). The rest of the settings are kept as their default values. For more information about the best settings to be used in ALAMO, please see the following documentation: https://foqus.readthedocs.io/en/latest/chapt_surrogates/tutorial/alamo.html

Note that setting number 42 is the name of the python file that gets created after ALAMO runs. It contains the Pyomo model for optimization, based on the ALAMO generated surrogate model. This python file is accessed by the SM based optimizer.

Step 2.4 - Under 'Execution', run ALAMO, as shown in Figure 9:

Step 3 - Mathematical Optimization:

Step 3.1 - Problem Setup - select optimization variables: In the Optimization module, select 'FLASH.PRES' as the decision variable. Keep the other input variables fixed, as shown in Figure 10.

Step 3.2 - Problem Setup - objective function and additional flowsheet constraints declaration: The objective is to maximize the separation process, therefore, we assume that the selling price of the vapor and liquid are \$5/kg and \$30/kg, respectively. Additionally, the CO2 vapor stream

FOQUS - C:\Users\deshpana\Desktop\Automatic Surrogate Based Optimization\Automatic_Surrogate_Optimization_Flash_Tutorial_Images.foqus - Last saved: 2020-06-09T10:47:29

Session | Flowsheet | Uncertainty | Optimization | OUJ | SDOE | Surrogates | Settings | Help

Tool: ALAMO

Method Description | Data | Variables | Method Settings | Execution

Input Variables

Select All | Select None | Set XFACTOR from range

	Name	Min	Max	XFACTOR	EXTRAPXMIN	EXTRAPXMAX
1	<input type="checkbox"/> FLASH.INLETFLOW	50.0	100.0	1.0	0.0	1.0
2	<input type="checkbox"/> FLASH.INLETTEMP	25.0	30.0	1.0	0.0	1.0
3	<input checked="" type="checkbox"/> FLASH.PRES	1.0	10.0	1.0	0.0	1.0

Output Variables

Select All | Select None | Set ZMIN and ZMAX from current data filter

	Name	MAXTERMS	IGNORE	TOLMEANERROR	TOLRELMETRIC	ZMIN	ZMAX	CUSTOMCON
1	<input checked="" type="checkbox"/> FLASH.CARBOLIQ	-1	0	1e-6	1e-6	0.0	1.0	[]
2	<input checked="" type="checkbox"/> FLASH.CARBOVAP	-1	0	1e-6	1e-6	0.0	1.0	[]
3	<input checked="" type="checkbox"/> FLASH.ETHANLIQ	-1	0	1e-6	1e-6	0.0	1.0	[]
4	<input checked="" type="checkbox"/> FLASH.ETHANVAP	-1	0	1e-6	1e-6	0.0	1.0	[]
5	<input type="checkbox"/> FLASH.status	-1	0	1e-06	1e-06	0.0	1.0	[]
6	<input type="checkbox"/> graph.error	-1	0	1e-06	1e-06	0.0	1.0	[]

Fig. 8: Figure 6: Select variables for surrogate model generation

FOQUS - C:\Users\deshpana\Desktop\Automatic Surrogate Based Optimization\Automatic_Surrogate_Optimization_Flash_Tutorial_Images.foqus - Last saved: 2020-06-09T13:06:04

Session | Flowsheet | Uncertainty | Optimization | OUJ | SDOE | Surrogates | Settings | Help

Tool: ALAMO

Method Description	Data	Variables	Method Settings	Execution
Setting Name	Value		Description	
1 Initial Data Filter	"uq2"		Filter to be applied to the initial data set.	
2 Validation Data Filter	"uq2"		Data set used to compute model errors at the validation phase.	
3 MAXTIME	2000		Maximum total execution time in seconds.	
4 MINPOINTS	0		Convergence is assessed only if the simulator is able to compute the output variables for at least MINPOINTS out of the data points requested by ALAMO.	
5 NSAMPLE	0		Number of data points to be generated by sampling before any model is built.	
6 MAXSIM	5		Maximum number of successive simulator failures allowed before quit	
7 SAMPLER	"None"		Adaptive sampling method to be used. If adaptive sampling is used, also set MAXITER below.	
8 MAXITER	1		Maximum number of ALAMO iterations, 1 = no adaptive sampling, 0 = no limit	
9 PRESET	-111111		Value to be used if the simulation fails.	
10 MONOMIALPOWER	[1, 2, 3]		Vector of monomial powers considered in basis functions. Use an empty vector for none.	
11 MULTI2POWER	[1, 2, 3]		Vector of powers to be considered for pairwise combinations in basis functions. Empty vector for none.	
12 MULTI3POWER	[1, 2, 3]		Vector of three variables combinations of powers to be considered as basis functions.	
13 RATIOPOWER	[1, 2, 3]		Vector of ratio combinations of powers to be considered in the basis functions.	
14 EXPCFNS	<input checked="" type="checkbox"/>		Use or not of exponential functions as basis functions in the model.	
15 LOGFCFNS	<input checked="" type="checkbox"/>		Logarithmic functions are considered as basis functions if true; otherwise, they are not considered.	
16 SINFNS	<input type="checkbox"/>		Sine functions are considered as basis functions if true; otherwise, they are not considered.	
17 COSFCFNS	<input type="checkbox"/>		Cosine functions are considered as basis functions if true; otherwise, they are not considered.	
18 LINFNS	<input checked="" type="checkbox"/>		Linear functions are considered as basis functions if true; otherwise, they are not considered.	
19 CONSTANT	<input checked="" type="checkbox"/>		A constant will be considered as a basis function if true; otherwise, its not considered.	
20 CUSTOMBAS	[]		A list of user-supplied custom basis functions can be provided by the user.	
21 MODELER	"BIC"		Fitness metric to be used for model building.	

Fig. 9: Figure 7: Select appropriate method settings for surrogate model generation

FOQUS - C:\Users\deshpana\Desktop\Automatic Surrogate Based Optimization\Automatic_Surrogate_Optimization_Flash_Tutorial_Images.foqus - Last saved: 2020-06-10T08:48:20

Session | Flowsheet | Uncertainty | Optimization | OUJ | SDOE | Surrogates | Settings | Help

Tool: ALAMO

Method Description	Data	Variables	Method Settings	Execution
Setting Name	Value		Description	
23 SCREENER	"No Screening"		Regularization method is used to reduce the number of potential basis functions before optimization.	
24 SCALEZ	<input type="checkbox"/>		If used, the variables are scaled prior to the optimization problem is solved.	
25 GAMS	"gams"		GAMS path is needed. GAMS is the software used to solve the optimization problems.	
26 GAMSSOLVER	"BARON"		Name of preferred GAMS solver for solving ALAMO's mixed-integer quadratic subproblems.	
27 SOLVEMIP	<input type="checkbox"/>		GAMS will be used to solve ALAMO's MIPs/MIQPs if checked	
28 FUNFORM	"Fortran"		Format for printing basis functions and models found by ALAMO. Fortran must be selected to generate FOQUS UQ and flowsheet models.	
29 MIPOPTCA	0.05		Absolute convergence tolerance for mixed-integer optimization problems.	
30 MIPOPTCR	0.0001		Relative convergence tolerance for mixed-integer optimization problems.	
31 LINEARERROR	<input type="checkbox"/>		If true, a linear objective is used when solving the mixed-integer optimization problems.	
32 CONREG	<input type="checkbox"/>		Specify whether a constraint regression is used or not.	
33 CRNCUSTOM	<input type="checkbox"/>		If true, constraints need to be entered in variables tab.	
34 CRNINITIAL	0		Number of random bounding points at which constraints are sampled initially.	
35 CRNMAXITER	10		Maximum allowed constrained regressions iterations.	
36 CRNVIOL	100		Number of bounding points added per round per bound in each iteration.	
37 CRNTRIALS	100		Number of random trial bounding points per round of constrained regression.	
38 CRTOL	0.001		Tolerance within which custom constraints must be satisfied. Real greater than 1e-5 is expected	
39 Input File	"alamo.alm"		File name for ALAMO input file.	
40 FOQUS Model (for UQ)	"alamo_surrogate_uq.py"		.py file for UQ analysis.	
41 FOQUS Model (for Flowsheet)	"alamo_surrogate_fs.py"		.py file flowsheet plugin, saved to user_plugins in the working directory.	
42 Pyomo Model for Optimization	"alamo_surrogate_pyomo_optim.py"		.py file, meant for surrogate based optimization to be used within FOQUS optimizer plugin, saved to user_plugins in the working directory.	
43 Standalone Pyomo Model for Optimization	"alamo_surrogate_pyomo_optim_standalone.py"		.py file, meant for surrogate based optimization to be used standalone, saved to user_plugins in the working directory.	

Fig. 10: Figure 8: Select appropriate method settings for surrogate model generation continued

```

FOQUS - C:\Users\deshpana\Desktop\Automatic Surrogate Based Optimization\Automatic_Surrogate_Optimization_Flash_Tutorial_Images.foqus - Last saved: 2020-06-09T13:06:04
Session  Flowsheet  Uncertainty  Optimization  OUU  SDOE  Surrogates  Settings  Help

Tool: ALAMO

Method Description  Data  Variables  Method Settings  Execution

-----
Starting ALAMO

Exec File Path:  C:\ALAMO_052020\alamo.exe
Sub-directory:   alamo
Input File Name: alamo.alm
Output File Name: alamo.lst
SAMPLER:        None
UQ Driver File:  alamo_surrogate_uq.py

-----
ALAMO version 2020.5.2. Built: WIN-64 Sat May 2 11:56:24 EDT 2020

If you use this software, please cite:
Cozad, A., N. V. Sahinidis and D. C. Miller,
Automatic Learning of Algebraic Models for Optimization,
AIChE Journal, 60, 2211-2227, 2014.

ALAMO is powered by the BARON software from http://www.minlp.com/
-----
Licensee: Anuja Deshpande at US Department of Energy, anuja.deshpande@netl.doe.gov.
-----
Reading input data
Checking input consistency and initializing data structures
Warning: powers of 1 will be discarded from the set of basis functions.

Step 0: Initializing data set
User provided an initial data set of 20 data points
We will sample no more data points at this stage
-----
Iteration 1 (Approx. elapsed time 0.16E-01 s)

Step 1: Model building using BIC

Model building for variable FLASH_CARBOLIQ
----
BIC = -30.5 with FLASH_CARBOLIQ = 2.4 * FLASH_PRES
----
BIC = -104. with FLASH_CARBOLIQ = 2.1 * FLASH_PRES + 0.38E-01 * FLASH_PRES**2
----
BIC = -168. with FLASH_CARBOLIQ = 2.3 * FLASH_PRES + 0.18E-02 * FLASH_PRES**3 - 0.45
----
BIC = -389. with FLASH_CARBOLIQ = 2.3 * FLASH_PRES + 0.14E-01 * FLASH_PRES**2 + 0.92E-03 * FLASH_PRES**3 - 0.36
----
BIC = -391. with FLASH_CARBOLIQ = 2.3 * FLASH_PRES + 0.32E-07 * exp(FLASH_PRES) + 0.14E-01 * FLASH_PRES**2 + 0.92E-03 * FLASH_PRES**3 - 0.36
----
BIC = -466. with FLASH_CARBOLIQ = 2.3 * FLASH_PRES + 0.23E-02 * log(FLASH_PRES) + 0.10E-06 * exp(FLASH_PRES) + 0.14E-01 * FLASH_PRES**2 + 0.90E-03 * FLASH_PRES**3 - 0.36

Model building for variable FLASH_CARBOVAP
----
BIC = 77.4 with FLASH_CARBOVAP = 37.
----
BIC = -64.5 with FLASH_CARBOVAP = - 2.5 * FLASH_PRES + 51.
----
BIC = -168. with FLASH_CARBOVAP = - 2.3 * FLASH_PRES - 0.18E-02 * FLASH_PRES**3 + 50.

```

Fig. 11: Figure 9: Run ALAMO to generate surrogate model

FOQUS - C:\Users\deshpana\Desktop\Automatic Surrogate Based Optimization\Automatic_Surrogate_Optim

Session Flowsheet Uncertainty Optimization OUU SDOE Surrogates Settings Help

Variables Samples Objective/Constraints Solver Run

	Variable	Type	Scale	Min	Max	Value
1	FLASH.INLETFLOW	Fixed	None	50.0	100.0	100.0
2	FLASH.INLETTEMP	Fixed	None	25.0	30.0	25.0
3	FLASH.PRES	Decision	Linear	1.0	10.0	1.0

Fig. 12: Figure 10: Select optimization variables

must be at least 98.5% pure. In the Objective/Constraints tab, under the objective function $f(x)$ expression section or box, enter $-5*f.FLASH.CARBOVAP - 30*f.FLASH.ETHANLIQ$ Under the inequality constraints section/box expression, enter $-f.FLASH.CARBOVAP / (f.FLASH.CARBOVAP + f.FLASH.ETHANVAP) + 0.985$

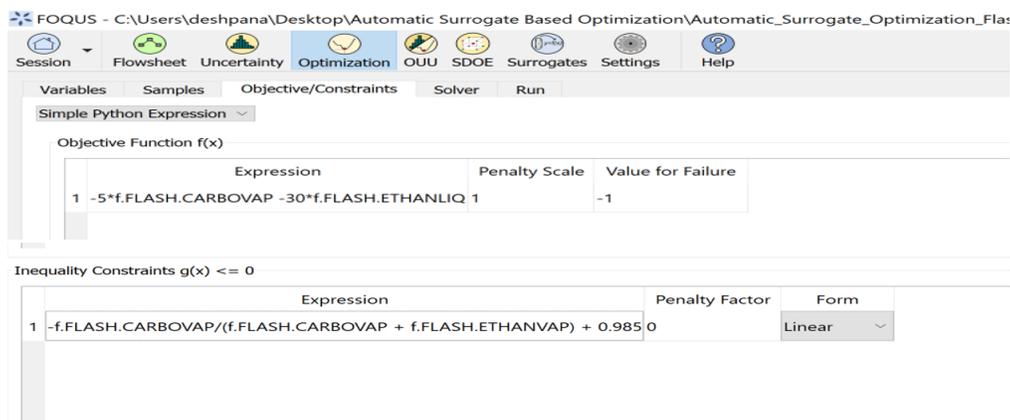


Fig. 13: Figure 11: Add objective function and constraints to the solver

Step 3.3 - Optimization solver settings: Under the solver tab, select “SM_Optimizer”

Figure 12 shows the solver options. solver options 1 to 11 are algorithm specific.

Solver option 1 selects the source of mathematical optimization solver. It can either be “gams” or “pyomo”. It is preferred to keep it at the default setting, “pyomo”.

Solver option 2 selects the mathematical optimization solver which will be used to solve the optimization at each iteration. It is preferred to keep it at the default setting, “ipopt”.

Solver option 3 selects the type of mathematical model that is formulated. This is used when “gams” is selected as the solver source. Depending on the type of problem, it can be non-linear programming “nlp”, linear programming “lp”, or mixed integer non-linear programming “minlp”. The setting would be “nlp” for this case.

Solver option 4 describes the maximum number of iterations that are allowed before the algorithm terminates. It can be set to 10 in this case.

Solver option 5 describes the value of ‘alpha’ which is a fractional multiplier that affects the extent to which the trust region shrinks at each iteration. The smaller this value is, faster is the rate of convergence of the algorithm. However, a very small value might discard the optimal solution. A value of 0.8 is chosen for this case.

Solver option 6 describes the number of Latin hypercube samples for generating the surrogate model in each iteration. Note that more the number of samples, a more accurate surrogate model could be obtained, however, the algorithm would take a longer time to converge. A value of 10 is chosen in this case.

Solver option 7 describes the lower limit of the ratio of upper and lower bounds of the decision variables. This condition is imposed while shrinking the trust region, to ensure that the solver converges. A value of 1 is chosen in this case.

Solver option 8 allows the user to display the mathematical optimization solution at each iteration

Solver options 9, 10, 11 describe the tolerance for the objective value, inequality constraint, and output variable value termination conditions, respectively. A value of 0.001 is chosen in this case.

FOQUS - C:\Users\deshpana\Desktop\Automatic Surrogate Based Optimization\Automatic_Surrogate_Optimization_Flash_Tutorial_Images.foqus - Last saved: 2020-06-10T08:48:20

Session Flowsheet Uncertainty Optimization OUU SDOE Surrogates Settings Help

Variables Samples Objective/Constraints Solver Run

Solver

Select Solver **SM_Optimizer**

Description

This solver provides the option to perform mathematical optimization based on surrogate models developed for the FOQUS flowsheet

Solver Options

	Option	Setting	Description
1	Solver Source	"pyomo"	Source of math optimization solver
2	mathoptsolver	"ipop"	Math Optimization Solver
3	mtype	"nlp"	Type of Math Optimization Model
4	Maxiter_Algo	10	Maximum iterations for surrogate based optimization algorithm
5	Alpha	0.8	Fractional Reduction in Surrogate Modeling Space
6	nLHS	10	Number of latin hypercube samples for generating modified surrogate model
7	Bound_Ratio	1	Ratio of upper and lower bounds of decision variables
8	tee	<input checked="" type="checkbox"/>	Display of solver iterations in terminal/anaconda prompt
9	Objective Value Tolerance	0.001	Tolerance for deviation from objective function evaluated at optimum, from Aspen Simulation
10	Inequality Constraint Tolerance	0.001	Tolerance for constraint satisfaction at optimum, from Aspen Simulation
11	Output Variable Tolerance	0.001	Tolerance for deviation of output variable values at optimum decision variables, calculated from Aspen Simulation, and Surrogate Model
12	Save results	<input checked="" type="checkbox"/>	Save math optimization results
13	Set Name	"PYOMO_SM"	Name of flowsheet result set to store data
14	Pyomo Surrogate File	"alamo_surrogate_pyomo_optim"	Name of python file containing surrogate based pyomo model
15	Surrogate Model Storing File	"SM_stored.txt"	Name of text file storing surrogate model from each algorithm iteration
16	Algorithm Convergence Plots File	"Algo_Convergence_File.py"	Name of python file with algorithm convergence plots
17	Parity Plot File	"parityplot.py"	Name of python file with the parity plot for the final surrogate model

Fig. 14: Figure 12: Select appropriate solver options

Solver option 12: if true, the optimization results will be stored in the FOQUS flowsheet. i.e. input and output variable values.

Since, each Algorithm iteration includes the generation of surrogate models, a call to Pyomo solver, and a call to the rigorous process simulation, the results are stored in the flowsheet results data tab, under the set name provided by the user in option 13. Solver option 14 corresponds to the python file containing the Pyomo model for the initial surrogate model developed in the previous steps. The name should match setting number 42 in the ALAMO settings. User can select the names of text and python files from option 15 to 17. The names should end with the required extension '.txt' for text file and '.py' for python file.

Step 3.4 Under the Run tab, click on 'start'. The main details for each iteration get displayed on the message window as the solver runs, the details are divided by section (i.e. step 3, step 4, step 5, etc.). After the final iteration, once the optimization is successful, the results get displayed as shown in the Figure 13 below:

Result Analysis:

The optimal solution was obtained in 3 iterations, and reported a revenue of \$ 1677.06 /hr and the problem was solved in 4 min 30 seconds. The overall implementation of the algorithm required a total of 23 rigorous simulations (ASPEN), 9 calls to the Ipopt solver, and two calls to ALAMO. Compared with a DFO solver the SM-based obtained the same solution in 6 min 30 seconds. The final optimization result is loaded in the node input and output variables, and gets stored in the flowsheet results data tab.

Solver option 15 corresponds to the file saving the surrogate models generated in each algorithm iteration; Solver option 16 corresponds to the python file containing plots that show termination condition values at each algorithm iteration. These files are useful to track the extent of convergence, as the algorithm proceeds. Finally, Solver option 17 corresponds to the python file that contains data to show the parity plot for the final surrogate model.

Note that these extra text and python files can be found in the "user_plugins" folder of FOQUS working directory.

MEA Carbon Capture System Optimization

Summary: This tutorial demonstrated the implementation of the surrogate model-based optimization. This includes setting up the Aspen model in FOQUS, generating the initial dataset (required for surrogate model development) in the UQ module, generating the surrogate model using ALAMO, and further, using it to solve the required optimization problem. In each iteration, after the optimization is solved, the rigorous model is evaluated at the optimum decision variable values returned by the optimization solver. Note that the final optimal solution reported by the algorithm corresponds to the solution of the rigorous model when evaluated at the optimal decision variable values. In comparison with other optimization tools provided by FOQUS, the SM-based optimizer has an advantage over DFO solvers in terms of total solution time and accuracy. For the flash optimization example, SM based optimizer took total 4 min 30 seconds, while the NLOpt DFO solver took 6 min 30 seconds for obtaining the same solution. For the MEA system example, SM based optimizer took total 48 mins, while the NLOpt DFO solver took 1 hr 5 mins. Overall, the SM based optimizer has expanded the possibility of solving optimization problems involving complex flowsheets within a shorter time frame as compared to DFO solvers, without compromising solution accuracy.

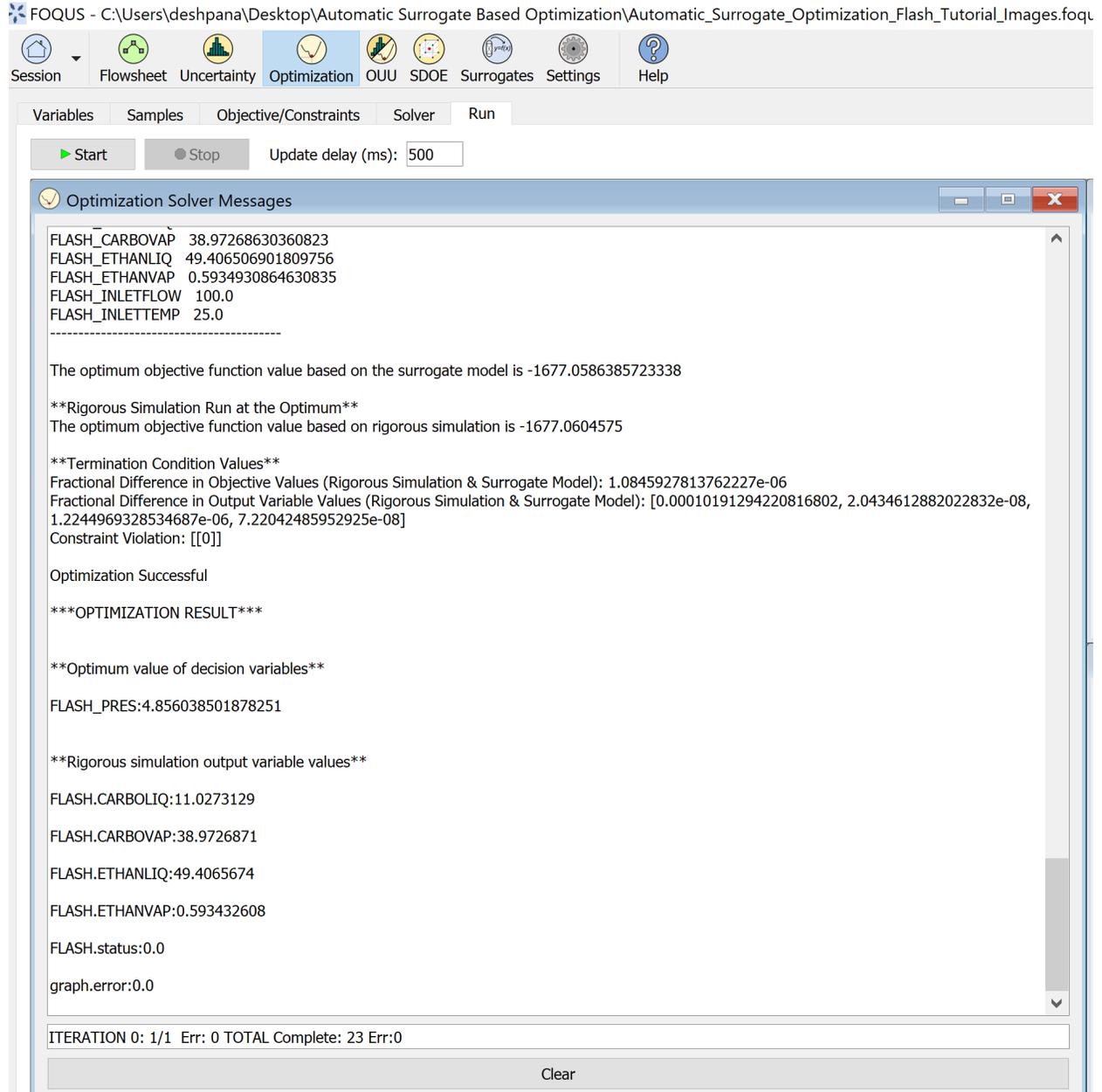


Fig. 15: Figure 13: Start the optimization and check results in the message window

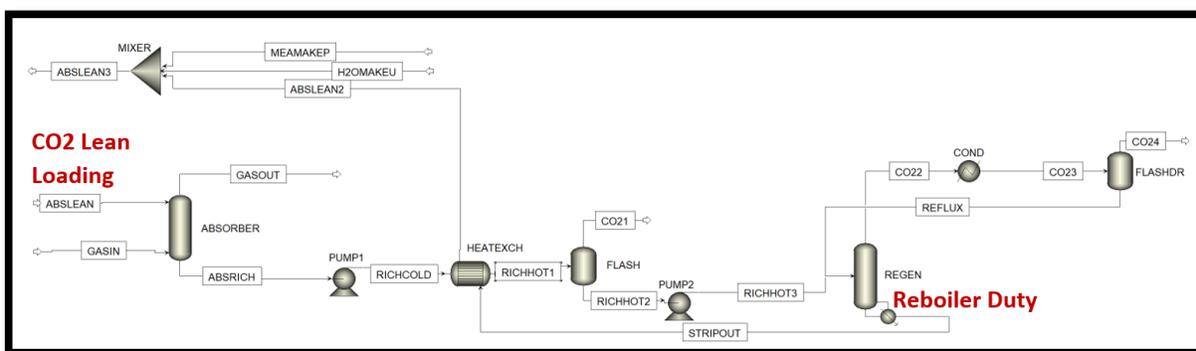


Fig. 16: Figure 14: MEA Carbon Capture System

Problem Statement: An MEA solvent based carbon capture system is set up in Aspen Plus v10, as shown in Figure 14, with a design specification of carbon capture rate 90 %. The flue gas flowrate to the absorber is 2266.1 kg/hr with 17.314 % by mass CO₂. It is sought to minimize the specific reboiler duty associated with the regenerator, by varying the CO₂ loading in the lean solvent entering the absorber.

Note: The Aspen, json, and FOQUS files for this example can be found in the folder:
 examples/tutorial_files/SM_Optimizer/MEA_Optimization

Result: After implementing the SM based optimization solver, the solution is: Optimum CO₂ lean loading = 0.1695 mol CO₂/mol MEA Rigorous model output variable values at optimum: Solvent Flowrate = 5438.703 kg/hr Total CO₂ Capture Rate = 353.1799 kg/hr SRD = 3.6382 MJ/kg CO₂

DEBUGGING

This chapter contains information that may be helpful in resolving a problem or filing a bug report.

16.1 How to Debug

Log files may contain very useful information when reporting problems. The log files are contained in the logs sub-directory of the FOQUS working directory. To change the log message levels in FOQUS go to the FOQUS **Settings** button from the Home window. From there various log settings can be changed. The debugging log level provides the highest level of information.

Almost any error that occurs in FOQUS should be logged. Occasionally, an error may occur that is difficult to find, or causes FOQUS to crash before logging it. In that case the “FOQUS Console” application can be used. All output from FOQUS, including messages that cannot be seen otherwise will be shown in a “cmd” window which will remain open even after FOQUS closes.

When running heat integration, the debugging information can be found in `\gamsHeatIntegration.lst`. This file includes detailed results and errors returned by GAMS.

Most UQ routines interact with PSUADE via Python wrappers. When PSUADE is running, the stdout is written to `psuadelog` in the working directory. (At present, only some PSUADE commands write to this log; however, this will be standardized in the near future so that all PSUADE commands write to this log.) Other errors that are due to the Python wrappers or PySide GUI components are written to the logs subdirectory in the working directory.

16.2 Known Issues

The following are known unresolved issues:

- Calculator blocks that use Excel in Aspen Plus do not work in FOQUS, because they are not supported by the Aspen Plus COM interface, and can only be used in interactive mode.
- The FOQUS flowsheet can be edited while a flowsheet evaluation, optimization, or UQ is running. This should not be allowed, and may cause unexpected behaviors. Currently changes to a flowsheet while running an evaluation will be ignored and reset when the evaluation is completed.
- The `win32com` module generates Python code, which it needs to run. This code is generated in the FOQUS install location “`\distwin32comgen_py`.” In some cases there may be a problem writing to that directory due to permission settings. This will prevent FOQUS from running simulations locally. If this error is encountered the solution is to make the “`gen_py`” directory user writable. So far, in testing, this error seems to occur in Windows 8 and 10, but not 7.

- FOQUS has trouble getting files from Turbine and saving them to the DMF when dealing with files in Turbine involving directories.
- The default port for TurbineLite is 8080. If another program is already using port 8080, there will be an error in FOQUS when connecting to TurbineLite. In the **Turbine** Tab of the Settings window, there is a tool to change the TurbineLite port. If the TurbineLite port is changed the configuration file that FOQUS uses to connect to TurbineLite, must also be changed.

16.3 Turbine

Configuration

Turbine Lite uses a configuration file to set many runtime options for submitted jobs, most of which are pre-set server information based on the software version. However, there are a few options which control the run behavior that users can modify to suit their needs. The “User Settings” section of the file passes timeout values which cap how many minutes Turbine will let a certain action occur before force-stopping.

On a local machine, the file is located at C:\Program Files (x86)\Turbine\Lite\Clients\AspenSinterConsumerConsole.exe.config and requires administrative access to save edits. FOQUS and Aspen need to be closed when you save the file so Turbine can update its config for new instances. Lines 93-108 set the “User Settings”, which include the timeout for the total run, the timeout for the run setup, and the timeout for the post initialization run. These values are all in minutes.

16.4 Contact

and

Support

There are multiple ways to contact the development team, get support, file a bug, make a feature request and even contribute code changes to FOQUS:

- Send a private email to ccsi-support@acceleratecarboncapture.org for contacting an internal set of developers.
- Subscribe to and send an email to our ccsi-users@acceleratecarboncapture.org public discussion forum to ask a question of the existing user base.
- Use any of the public GitHub features:
 - Read or start a new [Discussion](#)
 - Open a new [Issue](#) if you believe you’ve found a bug (please include detailed steps on how to reproduce the error, including if possible, screenshots and log files.) This is also where you can make feature requests.
 - Contribute changes to the FOQUS project by opening a [Pull Request](#)

General information about the Carbon Capture Simulation for Industry Impact (CCSI²) project, of which FOQUS is a part, can be found on the <https://www.acceleratecarboncapture.org/> web site.

DEVELOPER DOCUMENTATION

Since the [source code for all of FOQUS is publicly available](#), the more adventurous user may wish to look at the inner-workings of FOQUS to get a better understand how it works, contribute a fix to a bug, or add new features to the source tree. Other members of our CCSI partnership (national laboratories, industry and academic institutions) may be more actively involved in the development of FOQUS.

This chapter describes at a high level how any such person can set themselves up for getting, building, running, testing, documenting and contributing to FOQUS development.

17.1 Development Tools, Technology and Process

FOQUS is primarily written in Python. We use the following software development tools, technologies and processes:

- GitHub is where the [FOQUS source code](#) resides.
- We make extensive use of GitHub's [Issue Tracker](#) , [Pull Requests](#) and [Project Boards](#) for managing the development tasks using a modified Kanban development process.
- [ReadTheDocs](#) is used to generate and host our on-line documentation.
- For Continuous Integration (CI) we use [GitHub Actions](#).
- [Anaconda](#) for isolating Python runtime and development environment.

17.2 Developer Setup

Working as a developer is similar to how a user would work with FOQUS with the exception that they will need a copy of the source to work with. Here is rough set of steps to get setup:

- Download and install [Anaconda](#).
- In a terminal create a conda env in which to work:

```
conda create --name ccsi-foqus -c conda-forge python=3.10 pywin32=306
conda activate ccsi-foqus
```

- In a terminal, get the FOQUS source:

```
conda activate ccsi-foqus
cd CCSI-Toolset # Or a dir of your choice
git clone git@github.com:CCSI-Toolset/FOQUS.git # Note: clone the FOQUS repo if
↳ you expect to contribute
cd FOQUS
```

- Build and Install FOQUS as a developer:

```
pip install -r requirements-dev.txt # This will pick up both user and developer
→required packages.
foqus # Start the app
```

17.3 Pre-commit hooks (optional, but recommended)

Pre-commit hooks are scripts that are automatically run by Git “client-side” (i.e. on a developer’s local machine) whenever `git commit` is run. If the pre-commit scripts terminates with an error, the commit will be interrupted, requiring the developer to address the failure before being able to complete the commit.

Note: This is different (and complementary to) “server-side” checks, i.e. scripts that check the code on the side of the Git remote *after* the code is committed and pushed, such as the Continuous Integration (CI) suite triggered whenever a commit is pushed to an open PR in the FOQUS GitHub repository.

Pre-commit checks are especially useful to ensure that the code is formatted correctly *before* it is pushed to the FOQUS GitHub repository, which otherwise typically would cause the developer to 1) be notified by the failing CI check that the code wasn’t formatted; 2) run the formatter manually; 3) create a new commit with the formatting changes; 4) push the formatted code again.

FOQUS uses the **pre-commit** framework to manage a few hooks that are useful for FOQUS developers.

The **pre-commit** command is already installed as part of FOQUS’s developer dependencies. However, the pre-commit *checks* (i.e. the actual scripts that Git will be running) must be installed (using `pre-commit install`) as a separate step whenever the FOQUS repository is cloned:

```
pre-commit install
```

For more information, refer to the [pre-commit “Quick Start” page](#).

17.4 Run Tests

From top level of foqus repo:

```
pytest
python foqus.py -s test/system_test/ui_test_01.py
```

17.5 Building the Docs locally

To build a local copy of the documentation:

```
cd FOQUS/docs
make clean
make html
```

Then open the file `FOQUS/docs/build/html/index.html` to view the results.

17.6 Contact and Support

There are multiple ways to contact the development team, get support, file a bug, make a feature request and even contribute code changes to FOQUS:

- Send a private email to ccsi-support@acceleratecarboncapture.org for contacting an internal set of developers.
- Subscribe to and send an email to our ccsi-users@acceleratecarboncapture.org public discussion forum to ask a question of the existing user base.
- Use any of the public GitHub features:
 - Read or start a new [Discussion](#)
 - Open a new [Issue](#) if you believe you've found a bug (please include detailed steps on how to reproduce the error, including if possible, screenshots and log files.) This is also where you can make feature requests.
 - Contribute changes to the FOQUS project by opening a [Pull Request](#)

General information about the Carbon Capture Simulation for Industry Impact (CCSI²) project, of which FOQUS is a part, can be found on the <https://www.acceleratecarboncapture.org/> web site.

VECTOR VARIABLES SUPPORT CAPABILITY

18.1 Contents

18.1.1 Vector Variable Support Capability - Introduction

Motivation:

Vector variables and parameters have often been a part of chemical process, property, and economic models developed by scientists and engineers. Some process variables of this kind, include component concentrations in a material stream, and temperature, pressure, component concentration profiles across separation columns used for absorption, regeneration, and distillation operations. Model parameters for some physical and thermodynamic properties like viscosity, surface tension, interfacial area, enthalpy, entropy, fugacity, etc., are also vectors. Economic models include capital and operating costs indexed over different components, like unit operations, raw materials, utilities, and time horizon.

In order to develop or leverage such models for implementing simulation, optimization, and quantitative analysis in general, it is important for the modeling/model integration platform to support vector variables. Hence, the vector variable support capability is introduced in FOQUS, to allow creating and interfacing with vector variables across different modeling platforms like Python, MATLAB, and Aspen Plus.

New Features in FOQUS for vector support:

In order to support vector variables, along with continued support for scalar variables, the following new features have been introduced in FOQUS:

1. Automated GUI enabled addition and deletion of input and output vector variable elements in the node panel.
2. Access vector variable elements in the node script through a specific index-based python syntax.
3. Automated GUI enabled modification of SimSinter files based on Aspen models, for including the required vector variable elements in it.
4. Access vector variable elements from Aspen models by uploading the modified SimSinter files to turbine, and loading the turbine simulation in a FOQUS node.
5. Run node simulations successfully with vector-based Python, Pyomo, Aspen, and MATLAB models.
6. Access vector variable elements from the node, in the optimization module, easily through an index-based python syntax.

Potential**Applications:**

The vector variable support capability in FOQUS can be used for various applications based on the models, after they have been successfully set up.

Some of the applications include, but are not limited to:

1. Parameter estimation for vector-based models.
2. Decision making for process design, using sensitivity study for process vector variables.
3. Access and analyze computational fluid dynamics model profiles.

18.1.2 Example 1 - Handling Vector Variables in Python Models

Problem Statement: Consider the multi-dimensional Rosenbrock function:

$$\sum_{i=0}^4 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2$$

Set up the Python based function in a FOQUS flowsheet node using an input vector variable. Further simulate the function in the FOQUS flowsheet.

Reference: <https://www.sfu.ca/~ssurjano/rosen.html>

Instructions

Step 1: Create a FOQUS node named 'rosenbrock' in the flowsheet section, as shown in Figure 1.

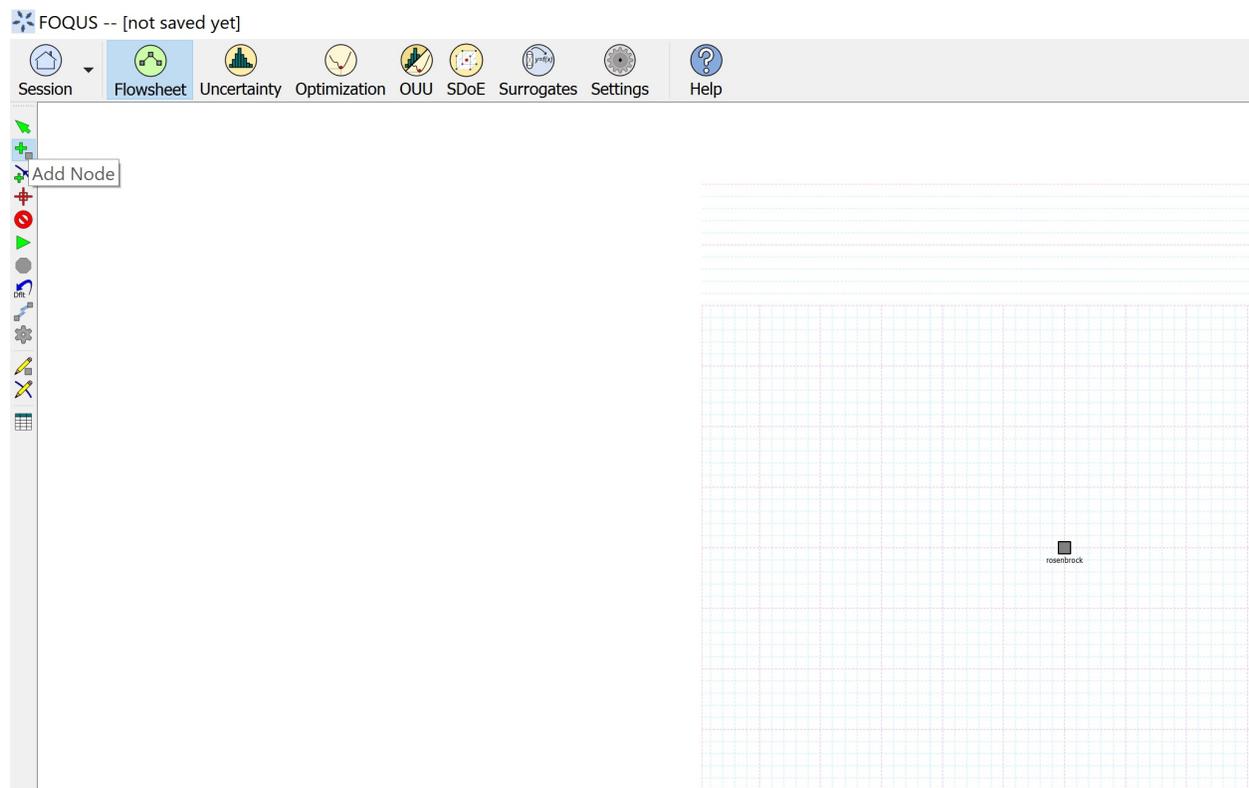


Fig. 1: Figure 1: FOQUS node for Rosenbrock function

Step 2: Open the node editor and under ‘Input Variables’ section, click on ‘+’ to add a new input variable. Further, in the user interface prompt, enter the variable name ‘x’ as shown in Figure 2, and click Ok.

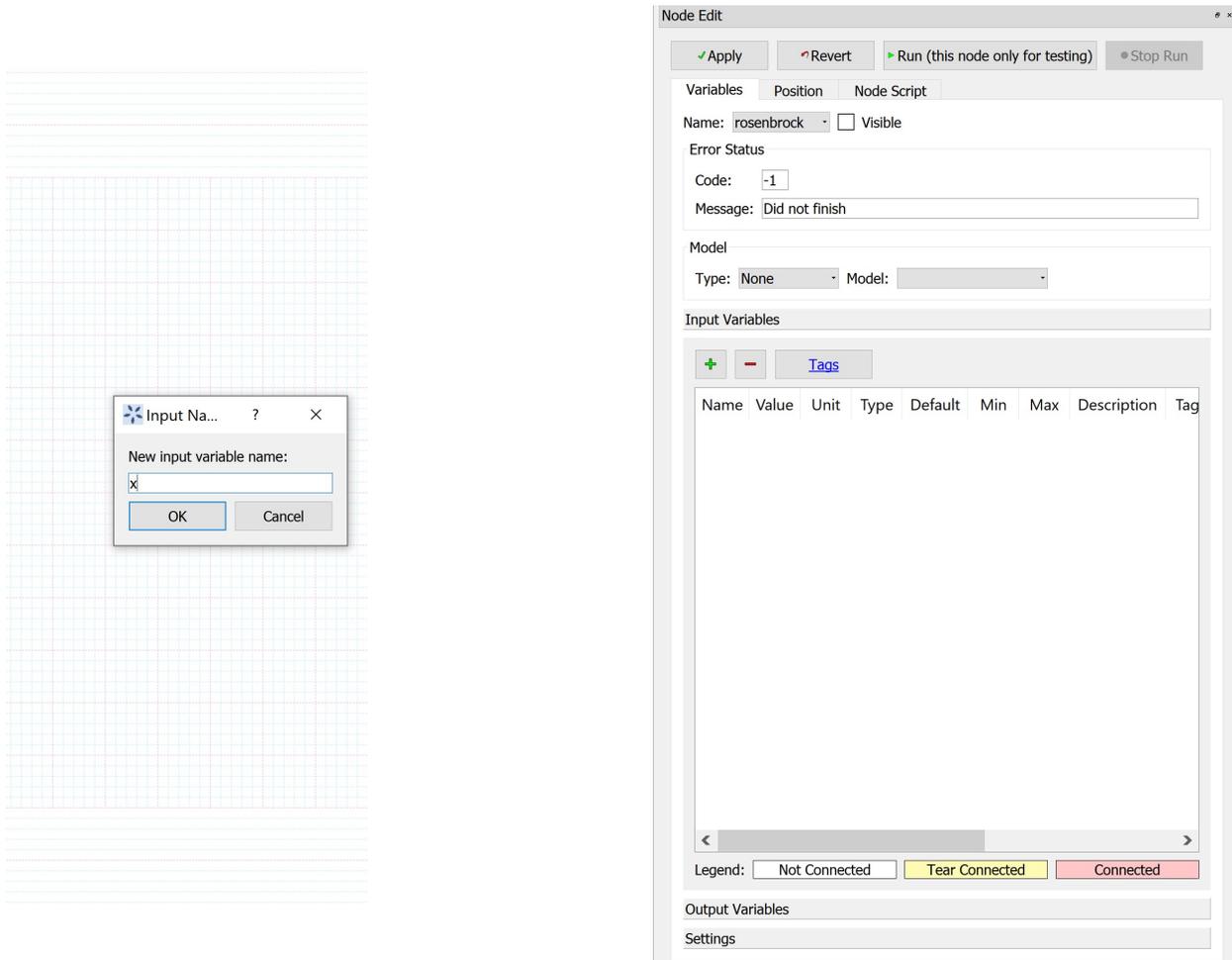


Fig. 2: Figure 2: Add input vector variable ‘x’

Step 3: In the next user interface prompt, enter 6 as the size of the variable as shown in Figure 3, and click Ok. This would correspond to the number of elements in the vector, if size is greater than 1.

Step 4: In the next user interface prompt, enter 0 as the minimum value for the variable elements, shown in Figure 4, and click Ok.

Step 5: In the next user interface prompt, enter 1 as the maximum value for the variable elements, shown in Figure 5, and click Ok.

Step 6: In the next user interface prompt, enter 0.5 as the value for the variable elements, shown in Figure 6, and click Ok. These are the values at which the simulation would run.

At this point in the implementation, the ‘Input Variables’ section of the node editor is populated with the vector variable elements x_0 through x_5 , (which correspond to vector variable ‘x’) as shown in Figure 7.

Step 7: Under ‘Output Variables’ section, click on ‘+’ to add a new output variable. Further, in the user interface prompt, enter the variable name ‘y’ as shown in Figure 8, and click Ok.

Step 8: In the next user interface prompt, enter 1 as the size of the variable (since it is a scalar), as shown in Figure 9, and click Ok.

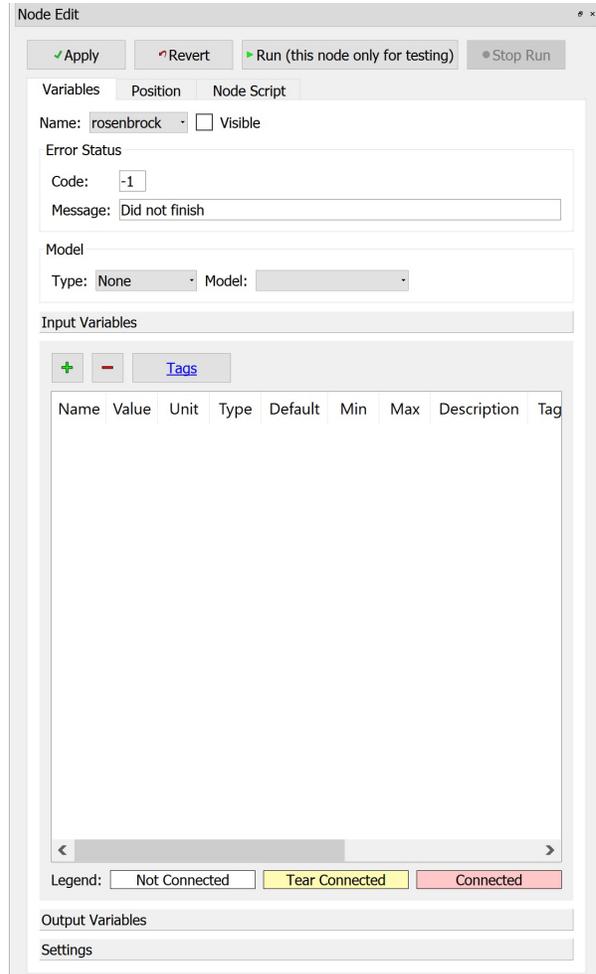
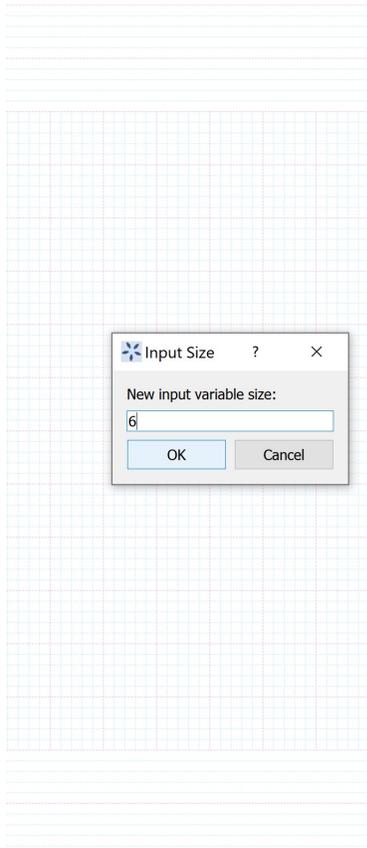


Fig. 3: Figure 3: Specify size of the input vector variable

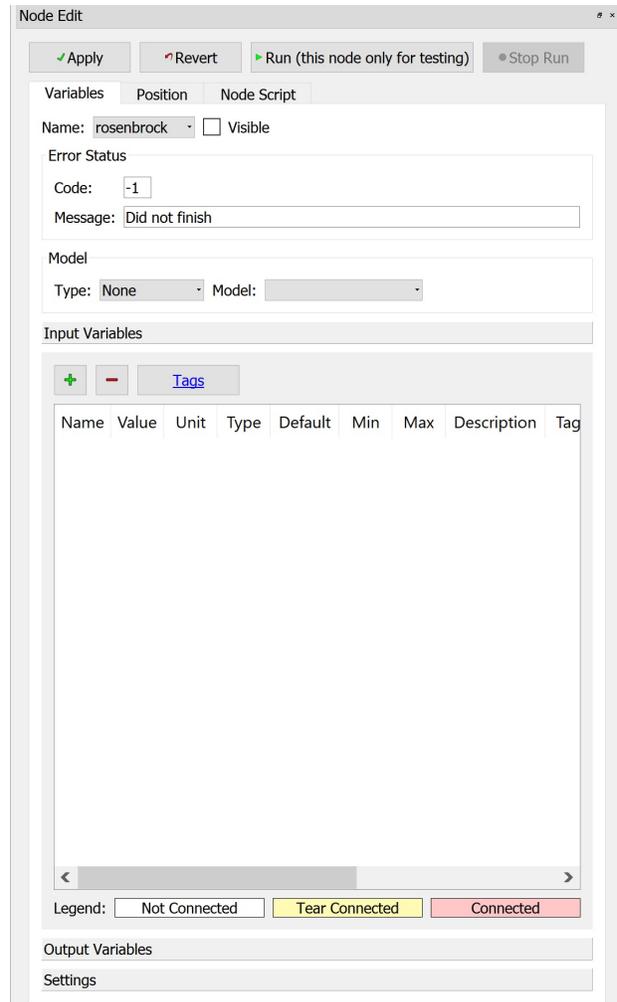
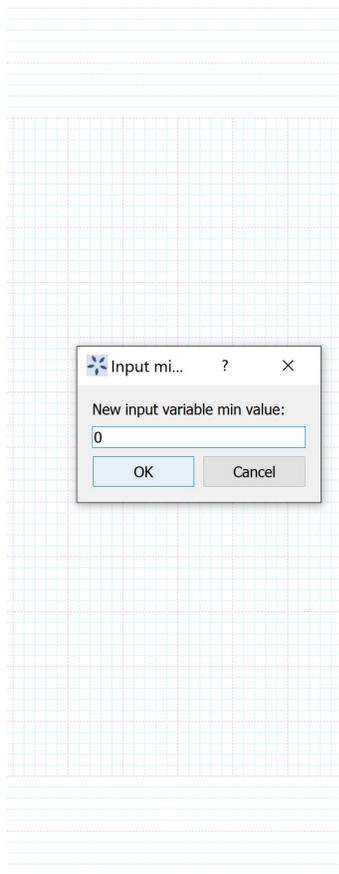


Fig. 4: Figure 4: Specify minimum value of the input vector variable elements

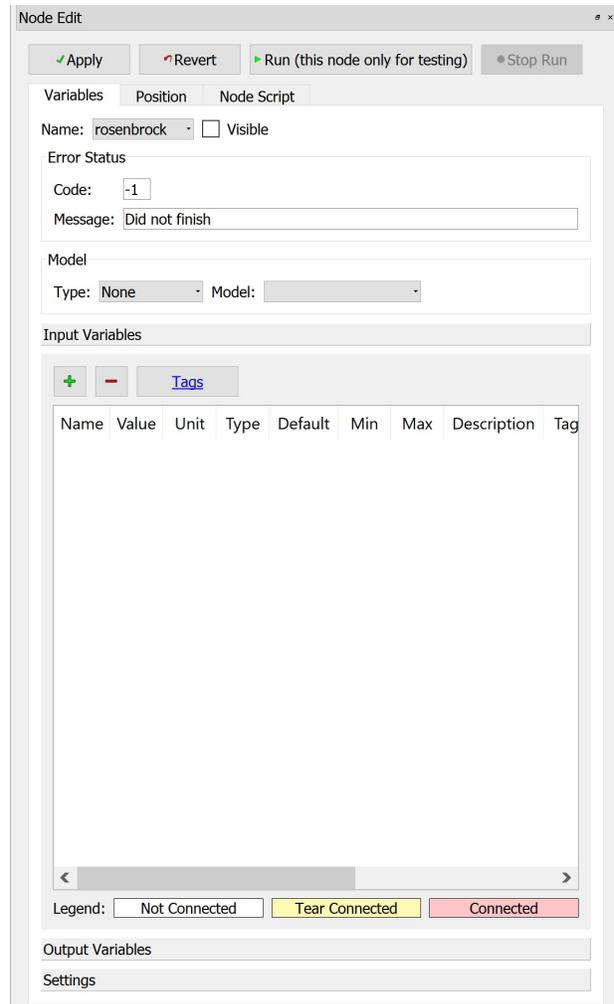
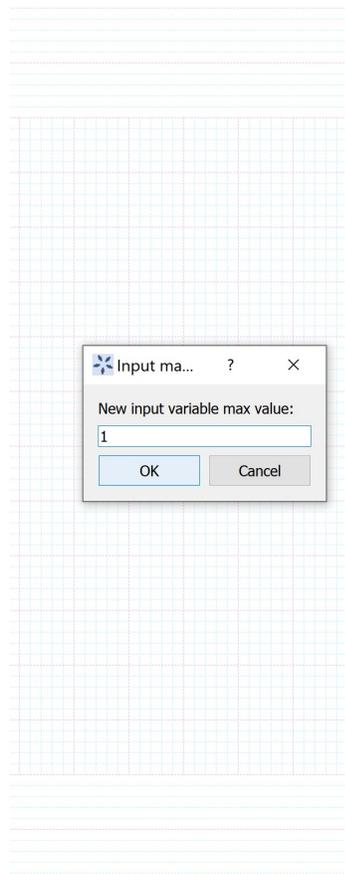


Fig. 5: Figure 5: Specify maximum value of the input vector variable elements

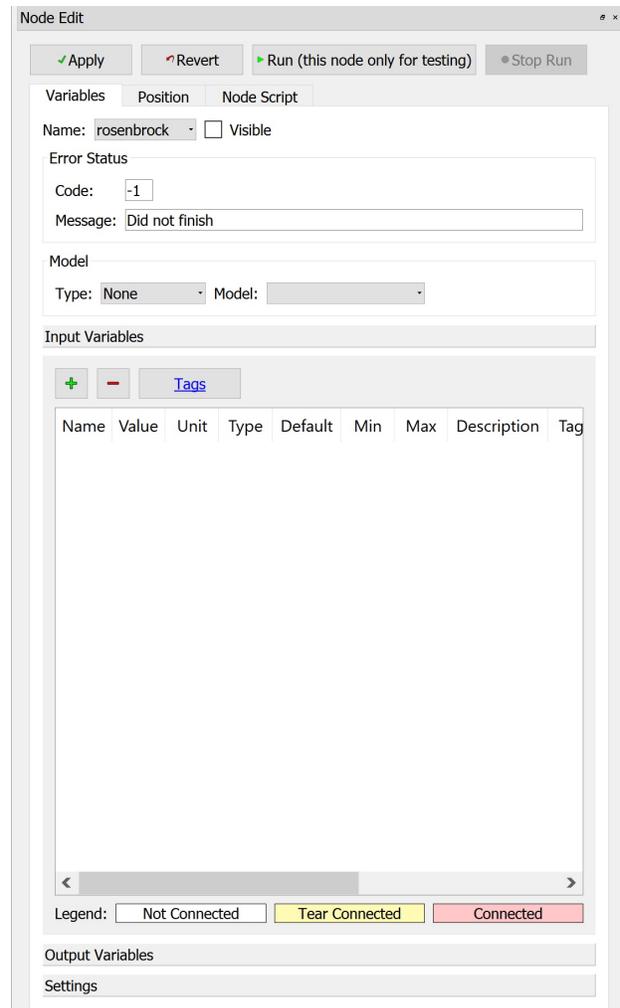
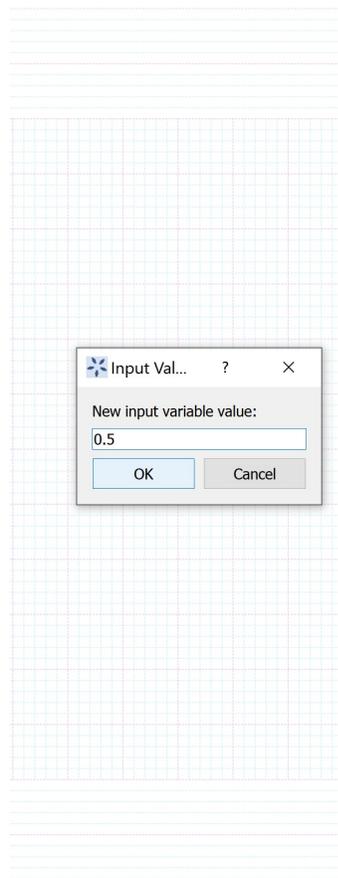


Fig. 6: Figure 6: Specify value of the input vector variable elements

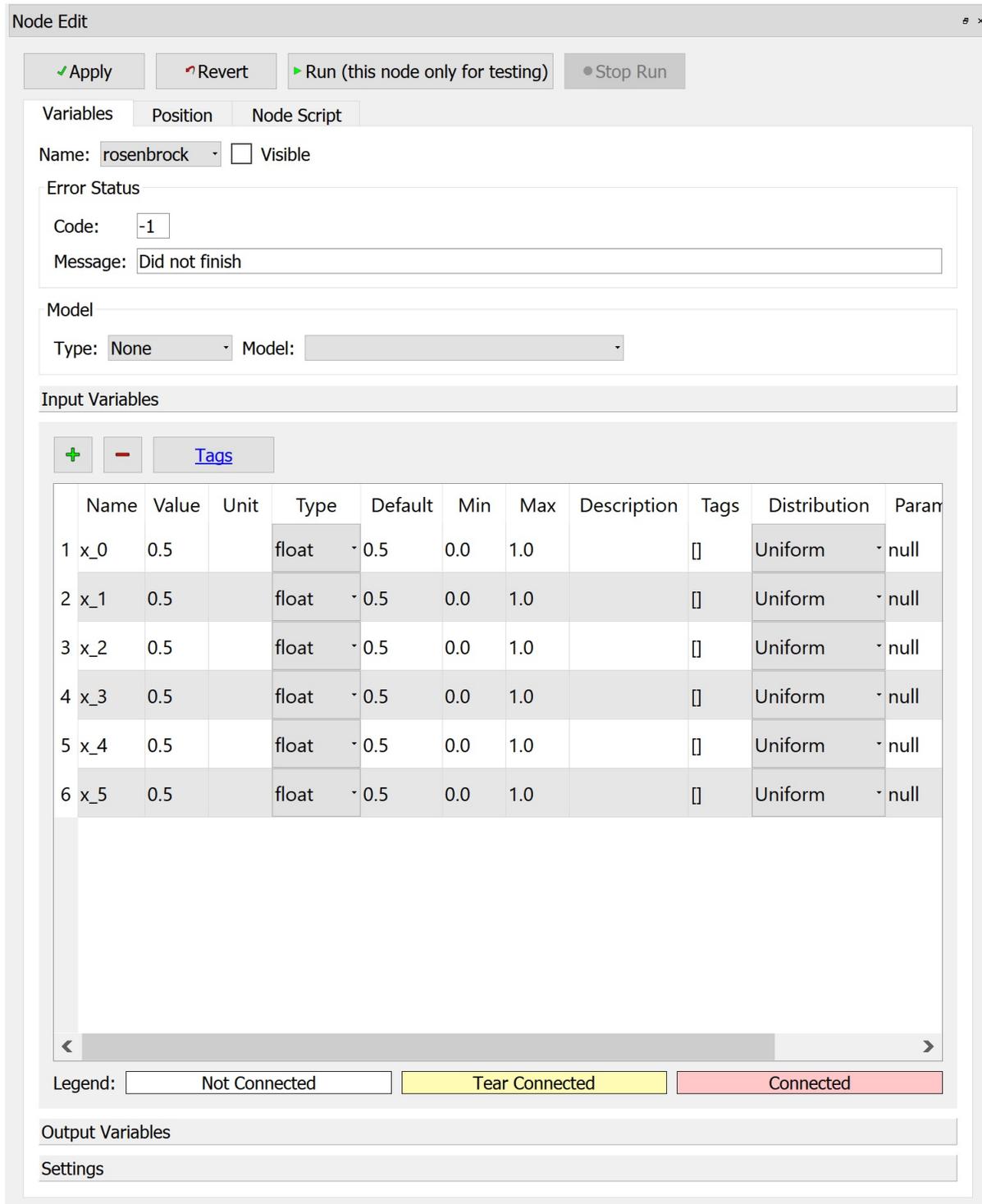


Fig. 7: Figure 7: Input vector variable elements in Node Panel

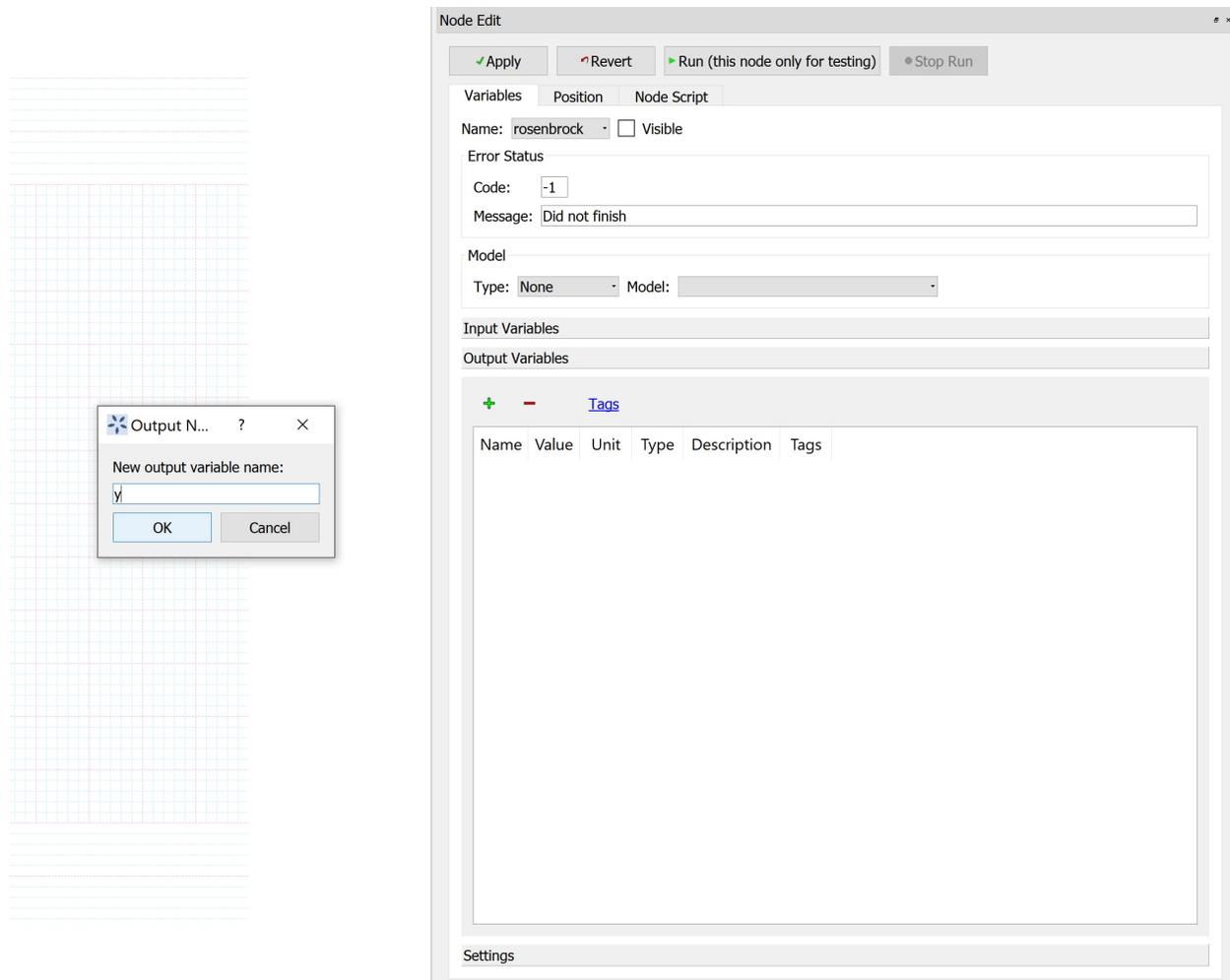


Fig. 8: Figure 8: Add output scalar variable 'y'

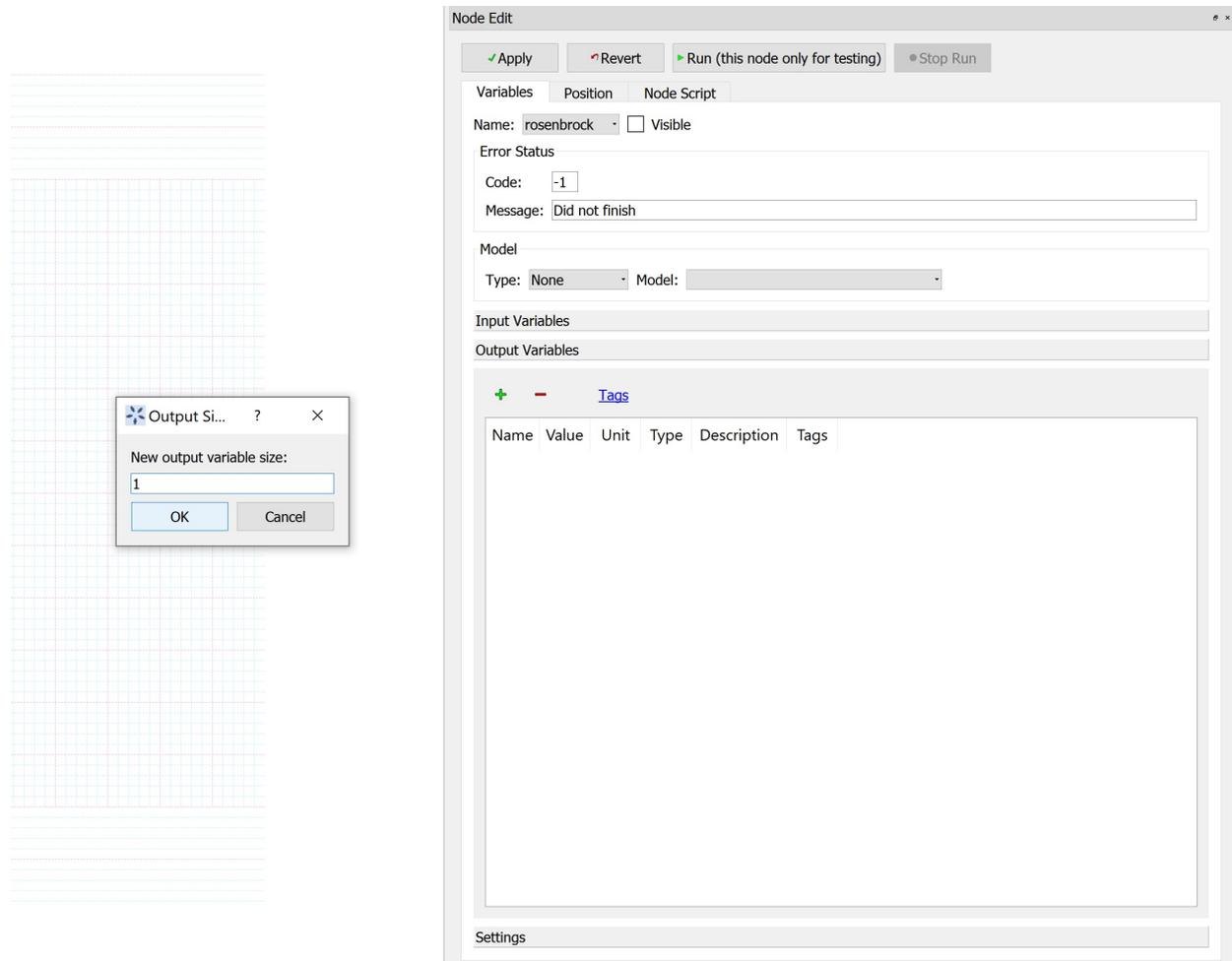


Fig. 9: Figure 9: Specify size of the output variable

At this point in the implementation, the ‘Output Variables’ section of the node editor is populated with the scalar variable y as shown in Figure 10.

Step 9: Enter the Rosenbrock function in the node script (set to ‘post’ mode) in python format, as shown in Figure 11.

In order to access the input vector variable element values in the node script, the syntax to be used is: `self.inVarsVector[“vector_name”].vector[index][‘value’]`, where the index ranges from 0 to (size-1) Eg. In this case, if the value of 1st element of “x” needs to be accessed, we would specify: `self.inVarsVector[“x”].vector[0][‘value’]`

Step 10: Run the flowsheet simulation, and the output variable value is displayed, as shown in Figure 12.

The simulation runs successfully, and the output variable value is obtained as per the Rosenbrock function set up in the node script, corresponding to the input vector element values.

It is to be noted that the functioning of other FOQUS modules like UQ, Optimization, and Surrogate Modeling is not affected by introducing vector variables. The individual scalar variables corresponding to the vectors get displayed in the user interface of these modules.

Important points to be noted

1. If the size specified for an input or output variable is greater than 1, the variable is constructed as a vector, and for size = 1, it is created as a scalar.
2. The vector elements are separate scalar node variables that get created, and grouped together in a node vector variable object ‘x’, in the background of FOQUS. Hence, ‘x’ can be considered as a vector variable that gets created in the background, to store the scalar variables corresponding to its elements, at specific index locations.
3. Each scalar variable corresponding to the vector takes the bounds and values from the user interface prompt. Note that in order to specify different bounds and values for each vector element, the user can provide a list with values specific to the elements. For this example, if the node needs to be simulated for different values of each vector element, when prompted for the values, the user can enter the list [val_0, val_1, . . . , val_5] in the interface, where val_0, val_1, . . . , val_5 are distinctly specified values for each element by the user.
4. The output vector variables are constructed in a similar manner, as the input vector.
5. In order to access the input or output vector variable element values in the node script, the syntax to be used is:
`self.inVarsVector[“vector_name”].vector[index][‘value’]` for input vector variable elements, and
`self.outVarsVector[“vector_name”].vector[index][‘value’]` for output vector variable elements where the index ranges from 0 to (size-1)
6. The syntax for accessing the scalar variables, created standalone, or associated with a vector, remains the same.

18.1.3 Example 2 - Handling Vector Variables in Aspen Plus Models

Problem Statement: Consider the MEA solvent based carbon capture system in the MEA_ssm product, as a part of the CCSI-Toolset on GitHub.

An application of vector variable support for Aspen Plus models in FOQUS is accessing the temperature, pressure, or composition profiles along a separation column, and observing the change in profile with variations in process parameters or configurations.

In this example, for the same simulation conditions specified in CCSI_MEAModel.bkp, we will access the absorber temperature profile for liquid and vapor phases as an output vector variable in FOQUS.

The absorber intercooler cooling water flowrates will be accessed as scalar input variables, in order to observe changes in the absorber temperature profile, with its variation, as a potential application for the user

Set up the Aspen Plus model, CCSI_MEAModel.bkp, in a FOQUS flowsheet node with the input and output variables as mentioned above. Further simulate the model in the FOQUS flowsheet for the original conditions provided in the Aspen file.

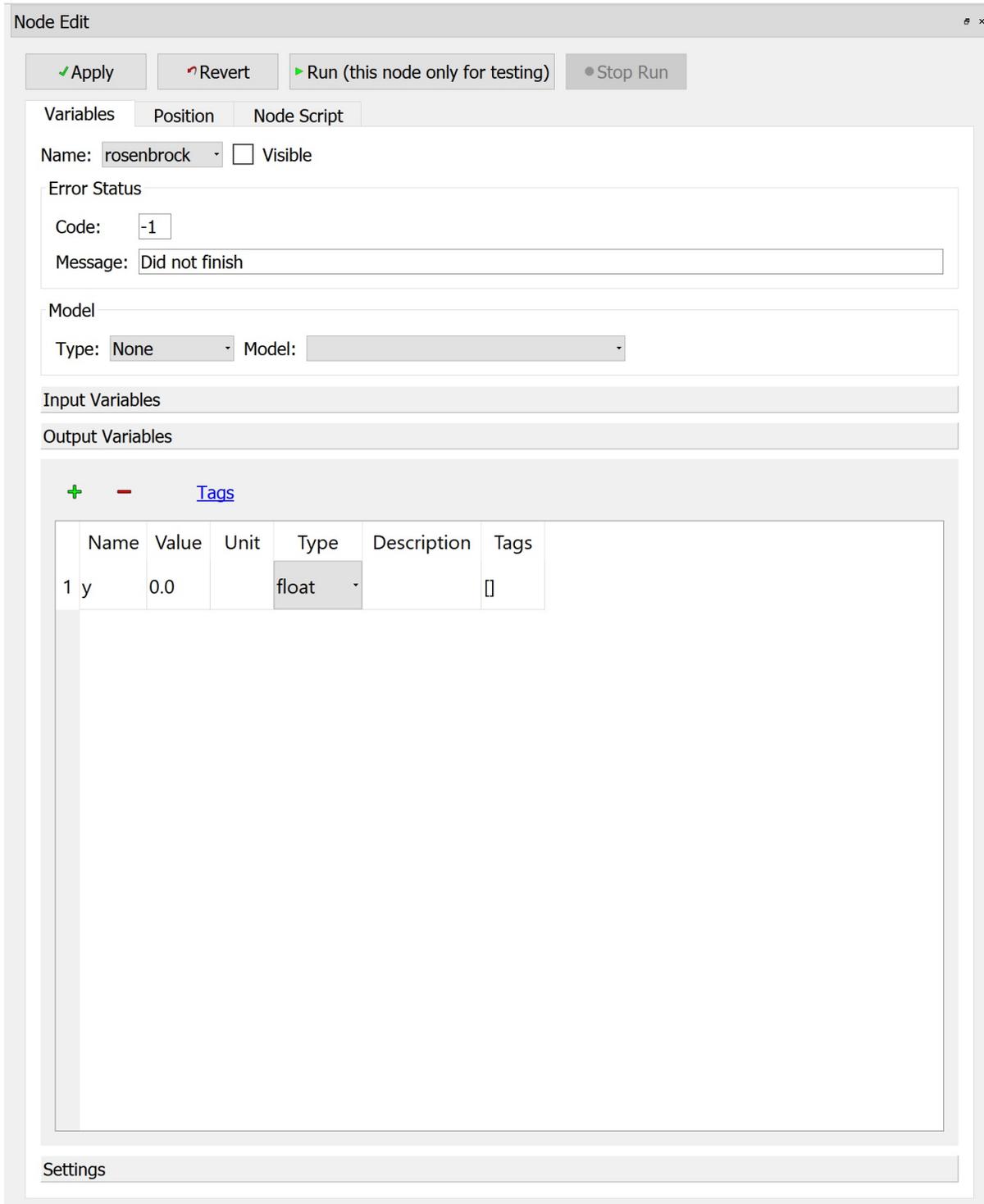
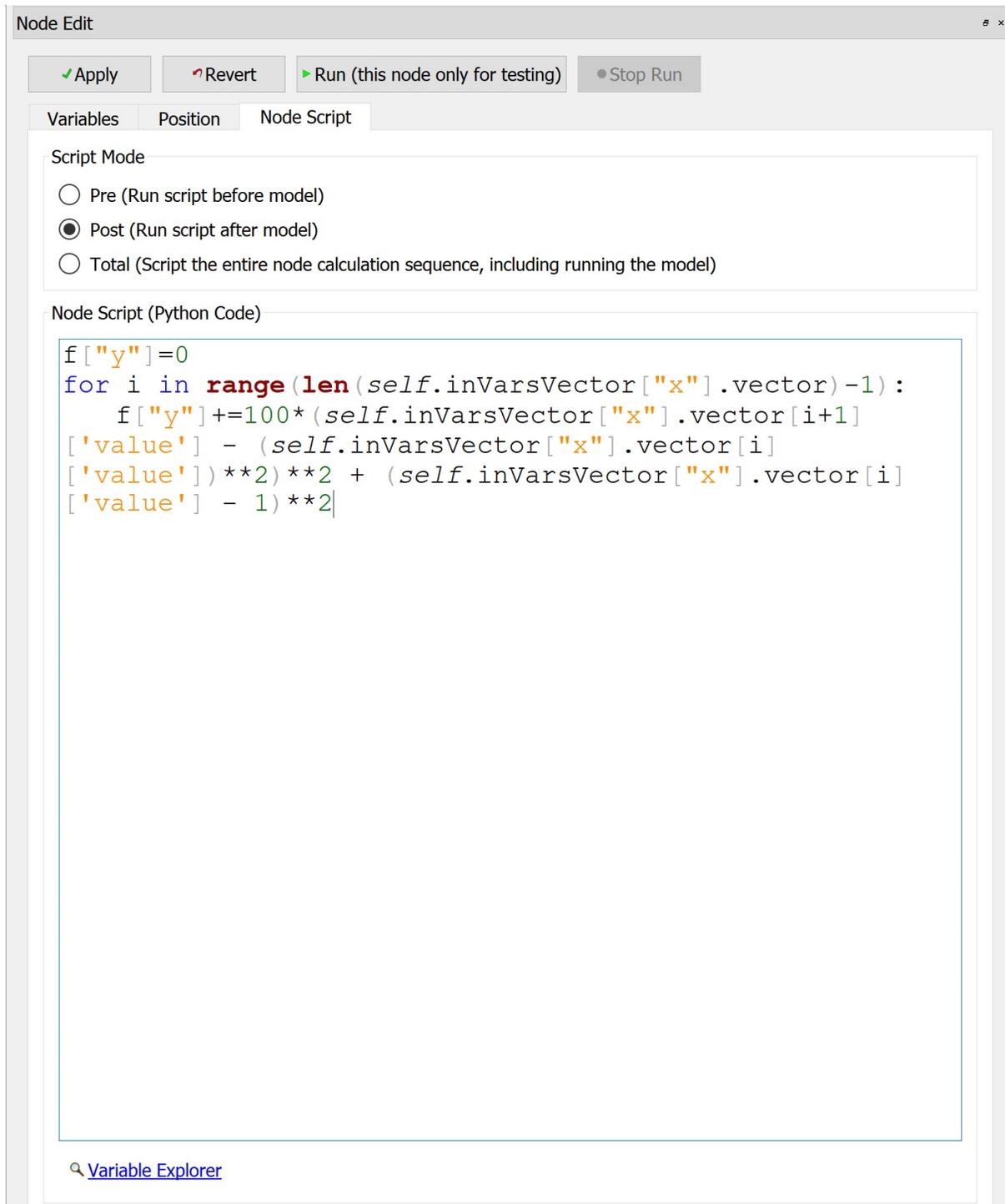


Fig. 10: Figure 10: Output variable in Node Panel



The screenshot shows a 'Node Edit' window with a toolbar containing 'Apply', 'Revert', 'Run (this node only for testing)', and 'Stop Run'. Below the toolbar are tabs for 'Variables', 'Position', and 'Node Script'. The 'Node Script' tab is active, showing a 'Script Mode' section with three radio buttons: 'Pre (Run script before model)', 'Post (Run script after model)' (which is selected), and 'Total (Script the entire node calculation sequence, including running the model)'. Below this is a 'Node Script (Python Code)' section containing the following code:

```
f["y"]=0
for i in range(len(self.inVarsVector["x"].vector)-1):
    f["y"]+=100*(self.inVarsVector["x"].vector[i+1]
['value'] - (self.inVarsVector["x"].vector[i]
['value'])**2)**2 + (self.inVarsVector["x"].vector[i]
['value'] - 1)**2
```

At the bottom left of the window, there is a link for 'Variable Explorer'.

Fig. 11: Figure 11: Rosenbrock function in Node Script

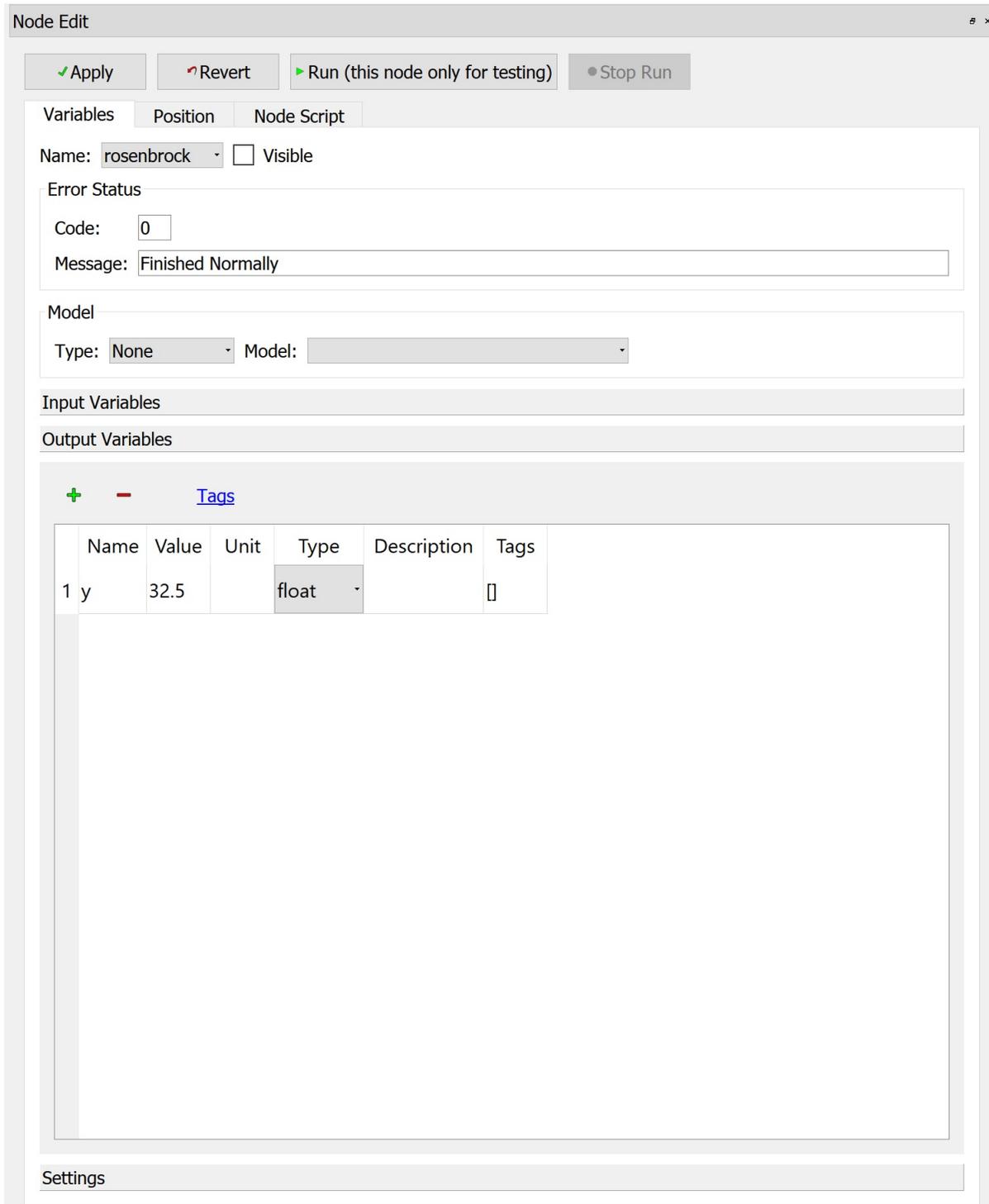


Fig. 12: Figure 12: Simulation Result

Instructions

Step 1: Create a SimSinter file named CCSI_MEAModel.json based on CCSI_MEAModel.bkp, ccsi10.dll, and ccsi.opt, using the SinterConfigGUI, as shown in Figure 1.

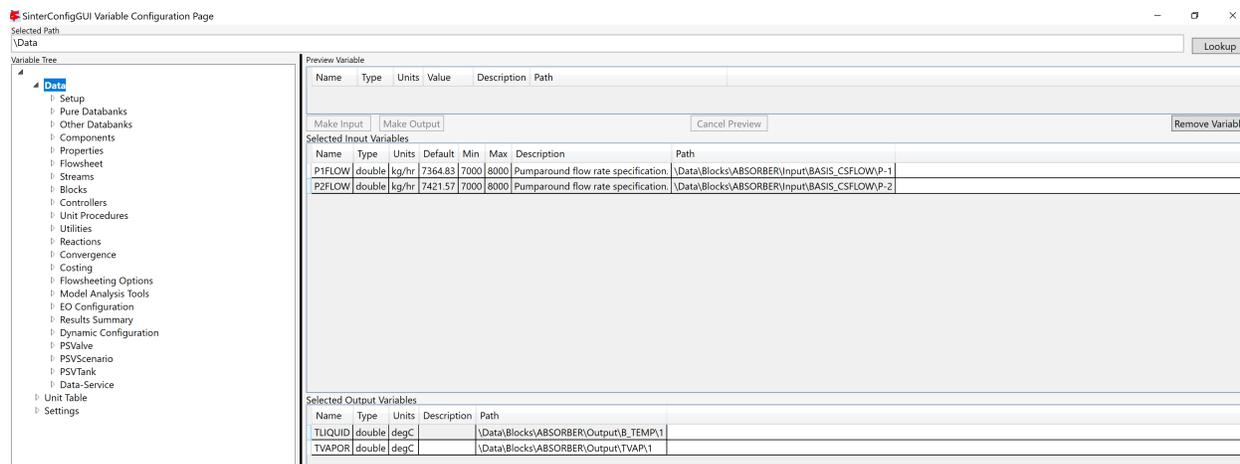


Fig. 13: Figure 1: Setting up the initial SimSinter file

In order to create the file, access the pumparound flowrates of the absorber as input variables, and temperatures of liquid, vapor phases at the first stage of absorber, following the ‘Path’ shown in the figure, for navigating through the variable tree in the GUI.

This would create an initial, non-vector SimSinter file, which will be used as a basis for vectorizing it in the next steps.

Step 2: Add the following files to the FOQUS Working Directory: CCSI_MEAModel.json, CCSI_MEAModel.bkp, ccsi10.dll, ccsi.opt.

Step 3: Open a FOQUS session, and give it an appropriate name. Further, click the Session tab dropdown, and select the “Vectorize SimSinter File” option, as shown in Figure 2.

A user interface dialog box with the heading “SimSinter Configuration File Vectorize” will be displayed as shown in Figure 3.

Step 4: In the user interface dialog shown in Step 3:

- (i) Enter the SimSinter file name, CCSI_MEAModel.json, in the text box under the heading.
- (ii) Enter None in the text box next to “Input Vector Details”, since we don’t have any input vectors of interest in our Aspen model.
- (iii) Next to “Output Vector Details”, enter a list of tuples, each one containing the vector variable name as string type, (according to the variable tree in SimSinter), and the number of elements required as a part of the vector.
In this case, “B_TEMP” is the variable name corresponding to the liquid phase temperature along the absorber column, and “TVAP” is the variable name corresponding to the vapor phase temperature along the absorber column, according to their path in SimSinter. These vector variables are indexed over the absorber stages, which is why the number of elements in each of them would be 90. Hence, enter the list [(“B_TEMP”,90),(“TVAP”,90)] in the text box.
- (iv) Enter CCSI_MEAModel_vectorize.json in the text box next to “Vectorize SimSinter File Name”.

After entering the user inputs for vectorizing the SimSinter file, the dialog box must resemble Figure 4:

Click “Ok” and wait for the dialog box to disappear.

Step 5: In the FOQUS working directory, check whether the file CCSI_MEAModel_vectorize.json has been created. If it hasn’t been created, repeat steps 3 and 4 with the correct user inputs. If it is created, open the json file directly to

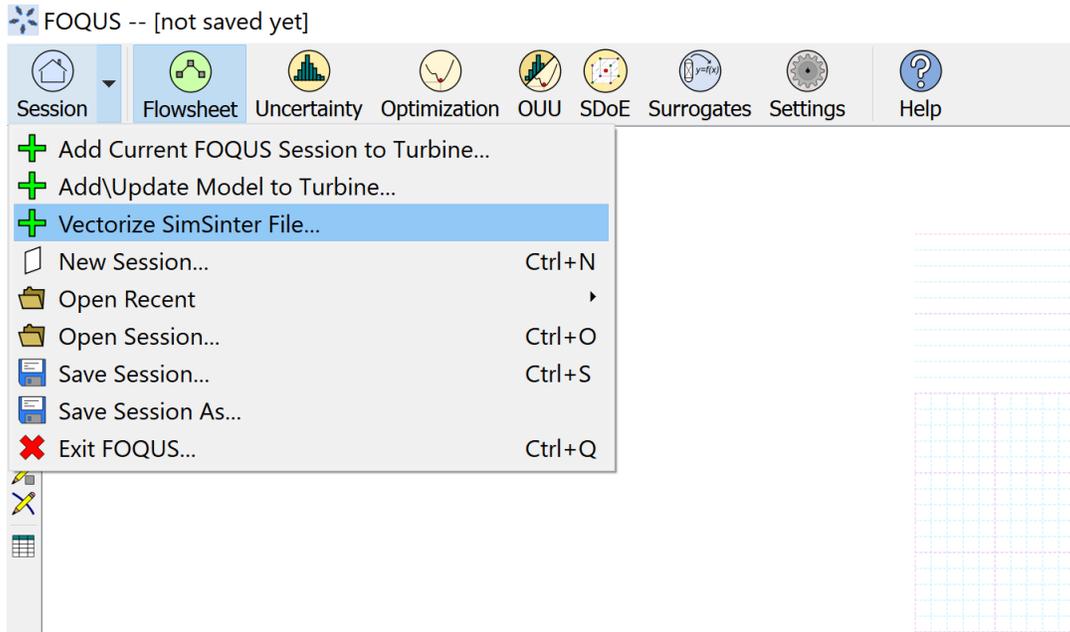


Fig. 14: Figure 2: Select the option to vectorize SimSinter file

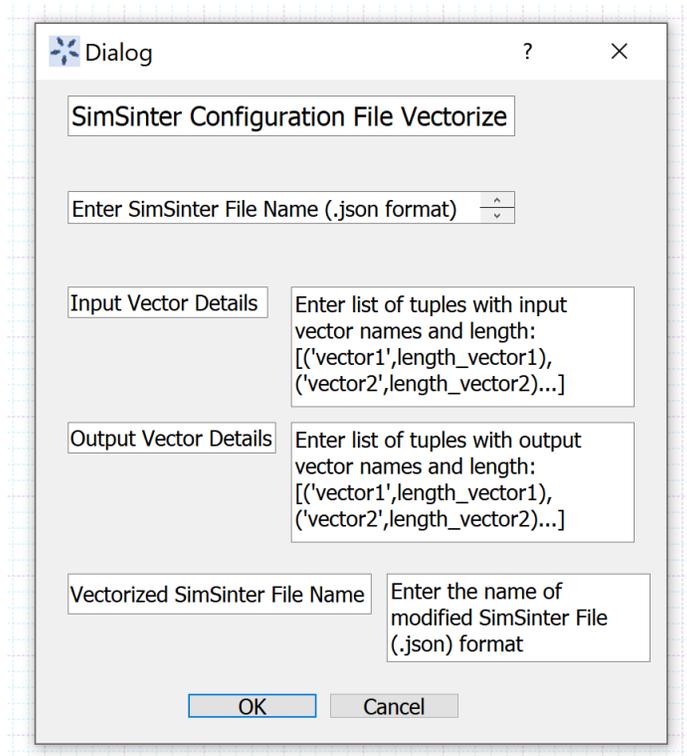


Fig. 15: Figure 3: User interface for vectorizing SimSinter file

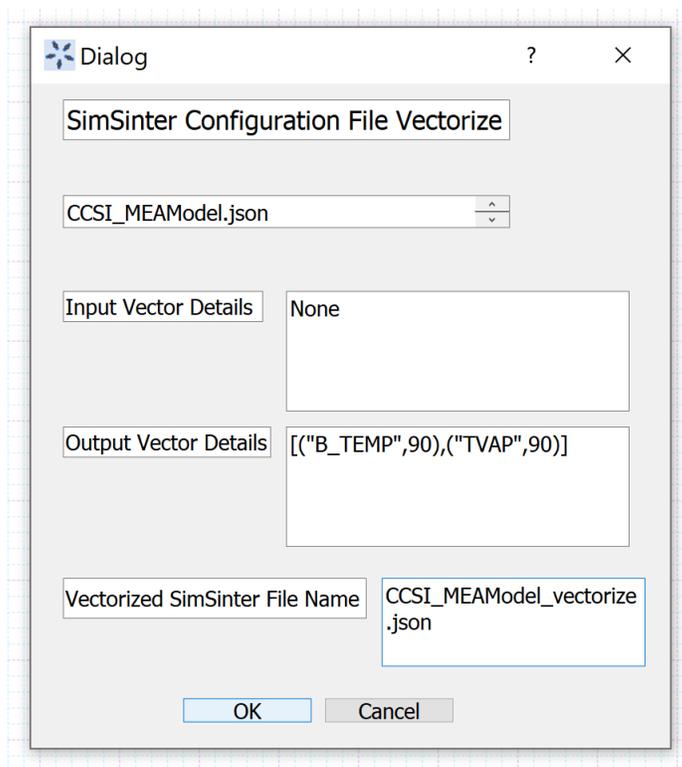


Fig. 16: Figure 4: User inputs entered for vectorizing SimSinter file

check whether it has been modified correctly.

In the json file, the dictionary corresponding to the “outputs” key must contain 180 scalar variables 90 each corresponding to the vector variables “B_TEMP” and “TVAP”. Each scalar output variable corresponding to the vector must contain a “vector” field which includes the vector variable it is associated with, and an “index” field, which defines its location in the vector. The indices range from 0 to (size-1), where size is the number of elements in the vector. Also, its path must correctly correspond to the one in the SimSinter variable tree.

Step 6: Under the “Section” dropdown in FOQUS, click on “AddUpdate Model to Turbine”, browse and upload CCSI_MEAModel_vectorize.json to Turbine, through the user interface that gets displayed.

Step 7: In the “Flowsheet” section, create a node named “MEA_Model”. Open the node editor, select Model Type “Turbine”, and Model “CSI_MEAModel_vectorize”.

Once the turbine model is loaded in the FOQUS node, the scalar input variables get displayed as shown in Figure 5. The scalar output variables associated with the vector variables “B_TEMP” and “TVAP” get displayed as shown in Figure 6.

Note that the output variable values will get loaded based on the status of the Aspen file that was used to build the original SimSinter file. The values displayed in the above figure already correspond to the Aspen file run and saved with results based on the original simulation inputs.

Step 8: Run the flowsheet simulation, to ensure that it has been set up correctly.

Once it runs successfully, the user can obtain the absorber temperature profiles for different values of cooling water flowrates, as per the requirement, by generating a simulation ensemble in the UQ module of FOQUS.

Important points to be noted

Node Edit

Variables Position Node Script

Name: MEA_Model Visible

Error Status

Code: -1

Message: Did not finish

Model

Type: Turbine Model: CCSI_MEAModel_vectorize

Input Variables

	Name	Value	Unit	Type	Default	Min	Max	Description	Tag
1	P1FLOW	7364.83	kg/hr	float	7364.83	7000.0	8000.0	Pumparound flow rate specification. []	
2	P2FLOW	7421.57	kg/hr	float	7421.57	7000.0	8000.0	Pumparound flow rate specification. []	

Legend: Not Connected Tear Connected Connected

Output Variables

Settings

Fig. 17: Figure 5: Turbine Model loaded in FOQUS node - Inputs

Node Edit

Apply Revert Run (this node only for testing) Stop Run

Variables Position Node Script

Name: MEA_Model Visible

Error Status

Code: 0

Message: Finished Normally

Model

Type: Turbine Model: CCSI_MEAModel_vectorize

Input Variables

Output Variables

+ - Tags

	Name	Value	Unit	Type	Description	Tags
1	B_TEMP_0	40.9899735	degC	float	None	[]
2	B_TEMP_1	40.9915624	degC	float	None	[]
3	B_TEMP_10	41.0117693	degC	float	None	[]
4	B_TEMP_11	41.0157402	degC	float	None	[]
5	B_TEMP_12	41.0205189	degC	float	None	[]
6	B_TEMP_13	41.0263793	degC	float	None	[]
7	B_TEMP_14	41.0336984	degC	float	None	[]
8	B_TEMP_15	41.0429994	degC	float	None	[]
9	B_TEMP_16	41.0550111	degC	float	None	[]
10	B_TEMP_17	41.0707517	degC	float	None	[]
11	B_TEMP_18	41.0916469	degC	float	None	[]
12	B_TEMP_19	41.119696	degC	float	None	[]

Settings

Fig. 18: Figure 6: Turbine Model loaded in FOQUS node - Outputs

1. In the “SimSinter Configuration File Vectorize” dialog box, the syntax for entering the input as well as output vector details is the same.
2. In order to access the input or output vector variable element values in the node script, the syntax to be used is:
self.inVarsVector[“vector_name”].vector[index][‘value’] for input vector variable elements, and
self.outVarsVector[“vector_name”].vector[index][‘value’] for output vector variable elements, where the index ranges from 0 to (size-1)
3. In order to access the input or output vector variable element values in the optimization module for specifying the objective function or constraint, the syntax to be used is:
x[“node_name”][“vector_name”][index] for input vector variables f[“node_name”][“vector_name”][index] for output vector variables
node_name is the name of the FOQUS node vector_name is the name of the vector from which the elements need to be accessed index corresponds to the element’s location in the vectors
The vector name and index can be found in the “vector” and “index” fields of the scalar variables associated with the vector, in the vectorized json file.
4. The syntax for accessing the scalar variables, created standalone, or associated with a vector, remains the same.

REFERENCES

- C. Tong, “PSUADE User’s Manual, Version 1.2.0,” Tech. Rep. LLNL-SM-407882, Lawrence Livermore National Laboratory, Livermore, CA 94551-0808, May 2011.
- A. Cozad, N. V. Sahinidis, and D. C. Miller, “Automatic Learning of Algebraic Models for Optimization,” *AICHE Journal*, vol. 60, pp. 2211–2227, 2014.
- C. B. Storlie, H. D. Bondell, B. J. Reich, and H. H. Zhang, “Surface estimation, variable selection, and the nonparametric oracle property,” *Statistica Sinica*, vol. 21, no. 2, pp. 679–705, 2011.
- C. B. Storlie, B. J. Reich, J. C. Helton, L. P. Swiler, and C. J. Sallaberry, “Analysis of computationally demanding models with continuous and categorical inputs,” *Reliability Engineering & System Safety*, vol. 113, pp. 30–41, 2013.
- B. J. Reich, C. B. Storlie, and H. D. Bondell, “Variable selection in bayesian smoothing spline anova models: Application to deterministic computer codes,” *Technometrics*, vol. 51, no. 2, pp. 110–120, 2009.
- J. H. Wegstein, “Accelerating Convergence of Iterative Processes,” *j-CACM*, vol. 1, no. 6, pp. 9–13, 1958.
- N. Hansen, *Towards a New Evolutionary Computation. Advances in Estimation of Distribution Algorithms, ch. The CMA Evolution Strategy: A Comparing Review*, pp. 75–102. Springer, 2006.
- S. G. Johnson, “The nlopt nonlinear-optimization package.” <http://ab-initio.mit.edu/nlopt>, May 2015.
- E. Jones, T. Oliphant, P. Peterson, et al., “Scipy: Open source scientific tools for python.” <http://www.scipy.org/>, May 2015.
- K. Bhat, B. Sherman, K. Ajayi, B. Ng, J. Eslick, J. Ou, and J. Kress, “Solventfit: A calibration tool for solvent-based CO2 capture models,” in 2015 CCSI Industry Advisory Board (IAB) Program Review Meeting, (Reston, VA), September 2015.
- K. Wu, F. Chollet, “Serialization and saving.” https://keras.io/guides/serialization_and_saving/, April 2020.
- M. Abadi, A. Agarwal, P. Barham, et al., “TensorFlow: An end-to-end open source machine learning platform.” <https://www.tensorflow.org/>, 2015.
- A. Meurer, C.P. Smith, M. Paprocki, et al., “SymPy 1.10.1 Documentation.” <https://docs.sympy.org/latest/index.html>, Jan 2017.
- S. Marcel, Y. Rodriguez, “Torchvision the machine-vision package of torch.” In Proceedings of the 18th ACM international conference on Multimedia (pp. 1485-1488), October 2010.
- L. Buitinck, G. Louppe, M. Blondel, et al., “API design for machine learning software: experiences from the scikit-learn project.” European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases, September 2013.
- M. A. Bouhlef, J. T. Hwang, N. Bartoli, et al., “A Python surrogate modeling framework with derivatives.” *Advances in Engineering Software*, Vol 135 (pp. 102662), September 2019.

CONTACT AND SUPPORT

There are multiple ways to contact the development team, get support, file a bug, make a feature request and even contribute code changes to FOQUS:

- Send a private email to ccsi-support@acceleratecarboncapture.org for contacting an internal set of developers.
- Subscribe to and send an email to our ccsi-users@acceleratecarboncapture.org public discussion forum to ask a question of the existing user base.
- Use any of the public GitHub features:
 - Read or start a new [Discussion](#)
 - Open a new [Issue](#) if you believe you've found a bug (please include detailed steps on how to reproduce the error, including if possible, screenshots and log files.) This is also where you can make feature requests.
 - Contribute changes to the FOQUS project by opening a [Pull Request](#)

General information about the Carbon Capture Simulation for Industry Impact (CCSI²) project, of which FOQUS is a part, can be found on the <https://www.acceleratecarboncapture.org/> web site.

COPYRIGHT AND LICENSE

Copyright (c) 2012 - 2024

21.1 Copyright

Notice

Foqus was produced under the DOE Carbon Capture Simulation Initiative (CCSI), and is copyright (c) 2012 - 2024 by the software owners: Oak Ridge Institute for Science and Education (ORISE), TRIAD National Security, LLC., Lawrence Livermore National Security, LLC., The Regents of the University of California, through Lawrence Berkeley National Laboratory, Battelle Memorial Institute, Pacific Northwest Division through Pacific Northwest National Laboratory, Carnegie Mellon University, West Virginia University, Boston University, the Trustees of Princeton University, The University of Texas at Austin, URS Energy & Construction, Inc., et al.. All rights reserved.

NOTICE. This Software was developed under funding from the U.S. Department of Energy and the U.S. Government consequently retains certain rights. As such, the U.S. Government has been granted for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable, worldwide license in the Software to reproduce, distribute copies to the public, prepare derivative works, and perform publicly and display publicly, and to permit other to do so.

21.2 License

Agreement

Foqus Copyright (c) 2012 - 2024, by the software owners: Oak Ridge Institute for Science and Education (ORISE), TRIAD National Security, LLC., Lawrence Livermore National Security, LLC., The Regents of the University of California, through Lawrence Berkeley National Laboratory, Battelle Memorial Institute, Pacific Northwest Division through Pacific Northwest National Laboratory, Carnegie Mellon University, West Virginia University, Boston University, the Trustees of Princeton University, The University of Texas at Austin, URS Energy & Construction, Inc., et al. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the Carbon Capture Simulation Initiative, U.S. Dept. of Energy, the National Energy Technology Laboratory, Oak Ridge Institute for Science and Education (ORISE), TRIAD National Security, LLC., Lawrence Livermore National Security, LLC., the University of California, Lawrence Berkeley National Laboratory, Battelle Memorial Institute, Pacific Northwest National Laboratory, Carnegie Mellon University, West Virginia University, Boston University, the Trustees of Princeton University, the University of Texas at Austin,

URS Energy & Construction, Inc., nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

You are under no obligation whatsoever to provide any bug fixes, patches, or upgrades to the features, functionality or performance of the source code (“Enhancements”) to anyone; however, if you choose to make your Enhancements available either publicly, or directly to Lawrence Berkeley National Laboratory, without imposing a separate written license agreement for such Enhancements, then you hereby grant the following license: a non-exclusive, royalty-free perpetual license to install, use, modify, prepare derivative works, incorporate into other computer software, distribute, and sublicense such enhancements or derivative works thereof, in binary and source code form.

22.1 Overview

The Framework for Optimization, Quantification of Uncertainty, and Surrogates (FOQUS) serves as the primary computational platform enabling advanced Process Systems Engineering (PSE) capabilities to be integrated with commercial process simulation software. It can be used to synthesize, design, and optimize a complete carbon capture system while considering uncertainty. FOQUS enables users to effectively screen potential capture concepts in the context of a complete industrial process so that trade-offs can be appropriately evaluated. The technical and economic performance characteristics of the capture process are highly dependent on employing an effective approach for process synthesis. Since large-scale carbon capture processes are outside of current experience, heuristic and evolutionary approaches are likely to be inadequate. Thus, a key aspect of FOQUS is that it bridges this gap by supporting a superstructure-based approach to determine the optimal process configuration and equipment interconnections.

22.2 Modules

1. SimSinter provides a wrapper to enable models created in process simulators to be linked into a FOQUS Flowsheet.
2. The FOQUS Flowsheet is used to link simulations together and connect model variables between simulations on the flowsheet. FOQUS enables linking models from different simulation packages.
3. Simulations are run through Turbine, which manages the multiple runs needed to build surrogate models, perform derivative-free optimization or conduct an Uncertainty Quantification (UQ) analysis. Turbine provides the capability for job queuing and enables these jobs to be run in parallel using cloud- or cluster-based computing platforms or a single workstation.
4. The Surrogates module can create algebraic surrogate models to support large-scale deterministic optimization, including superstructure optimization to determine process configurations. One of the available surrogate models is the Automated Learning of Algebraic Models for Optimization (ALAMO). ALAMO is an external product due to background Intellectual Property (IP) issues.
5. The Derivative-Free Optimization (DFO) module enables derivative-free (or simulation-based) optimization directly on the process models linked together on a FOQUS Flowsheet. It utilizes Excel to calculate complex objective functions, such as the cost of electricity.
6. The UQ module enables the effects of uncertainty to be propagated through the complete system model, sensitivity of the model to be assessed, and the most significant sources of uncertainty identified to enable prioritizing of experimental resources to obtain additional data.
7. The Optimization Under Uncertainty (OUU) module combines the capabilities of the DFO and the UQ modules to enable scenario-based optimization, such as optimization over a range of operating scenarios.

8. The Sequential Design of Experiments (SDOE) module currently provides a way to construct flexible space-filling designs based on a user-provided candidate set of input points. The method allows for new designs to be constructed as well as augmenting existing data to strategically select input combinations that minimizes the distance between points. Development of this module is continuing and will soon include other options for design construction.

22.3 Application Based Examples

FOQUS has been used to solve problems based on comprehensive analysis and optimization of carbon capture systems. Some relevant research work that includes FOQUS can be found in the following publications:

Chen, Y., Eslick, J.C., Grossmann, I.E., Miller, D.C., 2015. Simultaneous process optimization and heat integration based on rigorous process simulations. *Computers and Chemical Engineering* 81, 180–199.

Gao, Q., Miller, D.C., 2015. Optimization of amine-based solid sorbent chemistry for post-combustion carbon capture. Paper presented at: 2015 International Pittsburgh Coal Conference; 5–8 October 2015; Pittsburgh, PA, USA.

Ma, J., Mahapatra, P., Zitney, S.E., Biegler, L.T., Miller, D.C., 2016. D-RM Builder: A software tool for generating fast and accurate nonlinear dynamic reduced models from high-fidelity models. *Computers and Chemical Engineering* 94, 60–74.

Miller, D.C., Agarwal, D., Bhattacharyya, D., Boverhof, J., Chen, Y., Eslick, J., Leek, J., Ma, J., Mahapatra, P., Ng, B., Sahinidis, N.V., Tong, C., Zitney, S.E., 2017. Innovative computational tools and models for the design, optimization and control of carbon capture processes, in: Papadopoulos, A.I., Seferlis, P. (Eds.), *Process Systems and Materials for CO₂ Capture: Modelling, Design, Control and Integration*. John Wiley & Sons Ltd, Chichester, UK, pp. 311–342.

Soepyan, F.B., Anderson-Cook, C.M., Morgan, J.C., Tong, C.H., Bhattacharyya, D., Omell, B.P., Matuszewski, M.S., Bhat, K.S., Zamarripa, M.A., Eslick, J.C., Kress, J.D., Gattiker, J.R., Russell, C.S., Ng, B., Ou, J.C., Miller, D.C., 2018. Sequential Design of Experiments to Maximize Learning from Carbon Capture Pilot Plant Testing. In: Eden, M.R., Ierapetritou, M.G., Towler, G.P. (Editors), *13th International Symposium on Process Systems Engineering (PSE 2018)*. Elsevier, Amsterdam, pp. 283-288.

Additional research work can be found on <https://www.acceleratecarboncapture.org/publications>

22.4 Contact and Support

There are multiple ways to contact the development team, get support, file a bug, make a feature request and even contribute code changes to FOQUS:

- Send a private email to ccsi-support@acceleratecarboncapture.org for contacting an internal set of developers.
- Subscribe to and send an email to our ccsi-users@acceleratecarboncapture.org public discussion forum to ask a question of the existing user base.
- Use any of the public GitHub features:
 - Read or start a new [Discussion](#)
 - Open a new [Issue](#) if you believe you've found a bug (please include detailed steps on how to reproduce the error, including if possible, screenshots and log files.) This is also where you can make feature requests.
 - Contribute changes to the FOQUS project by opening a [Pull Request](#)

General information about the Carbon Capture Simulation for Industry Impact (CCSI²) project, of which FOQUS is a part, can be found on the <https://www.acceleratecarboncapture.org/> web site.