POEM

Release 1.0

Congjian Wang

May 08, 2024

CONTENTS:

CHAPTER

ONE

INTRODUCTION

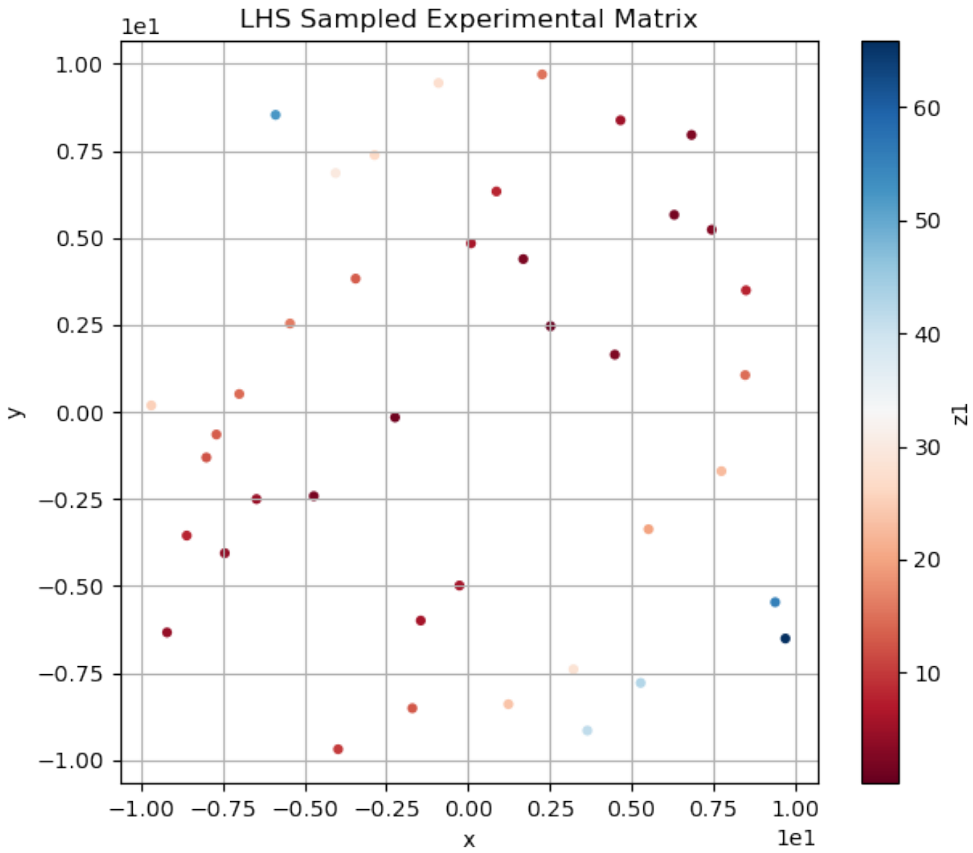
1.1 What is POEM

POEM is a platform for optimal experiment management, powered with automated machine learning to accelerate the discovery of optimal solutions, and automatically guide the design of experiments to be evaluated. POEM currently supports 1) random model explorations for experiment design, 2) sparse grid model explorations with Gaussian Polynomial Chaos surrogate model to accelerate experiment design ,3) time-dependent model sensitivity and uncertainty analysis to identify the importance features for experiment design, 4) model calibrations via Bayesian inference to integrate experiments to improve model performance, and 5) Bayesian optimization for optimal experimental design. In addition, POEM aims to simplify the process of experimental design for users, enabling them to analyze the data with minimal human intervention, and improving the technological output from research activities.

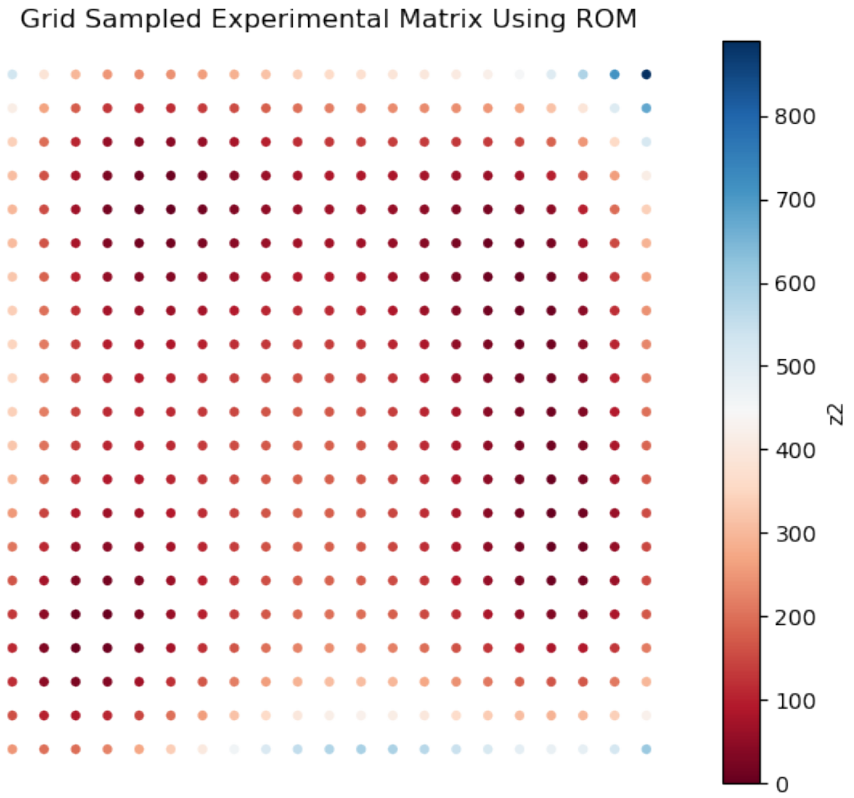
POEM leverages RAVEN (a robust platform to support model explorations and decision making) to allow for large scalability and reduction of the computational costs and provides access to complex physical models while performing optimal experimental design.

1.2 Capabilities

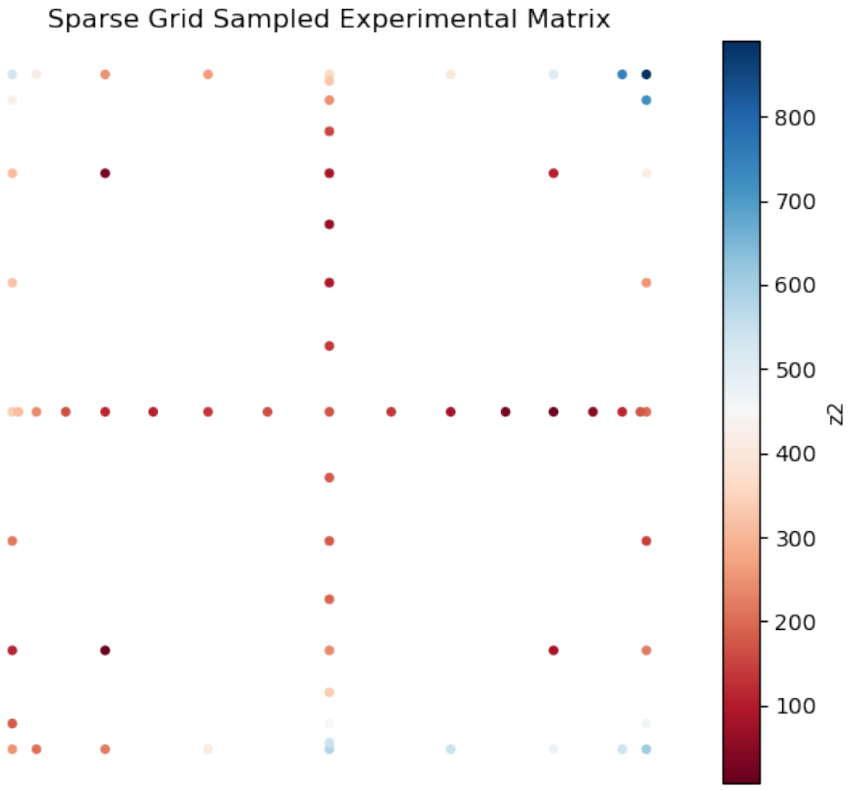
- Random model explorations for experiment design
  - See *Monte Carlo Sampling*
  - See *Latin Hypercube Sampling*

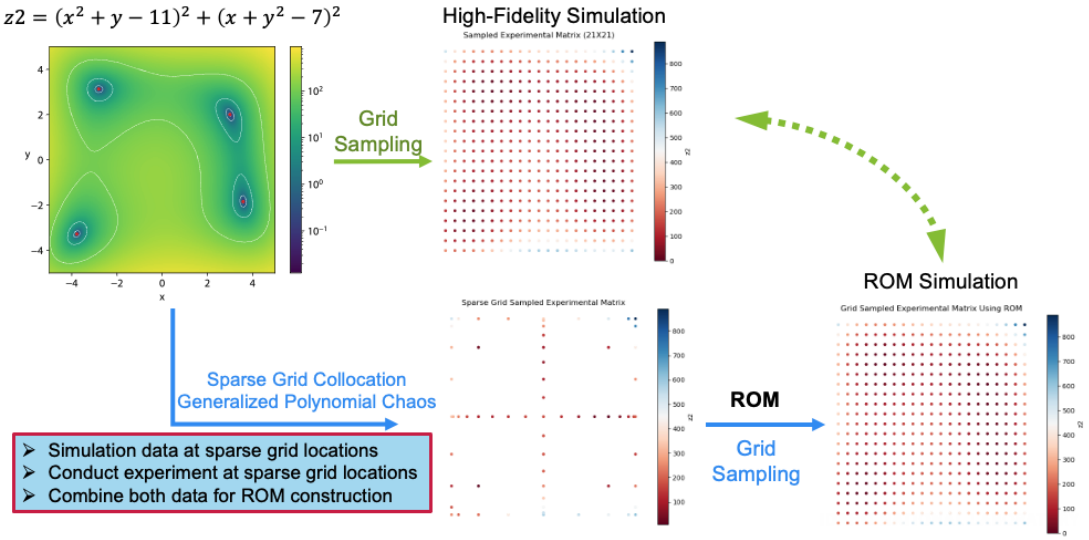


- Machine learning aided parameter space exploration
  - See *Train ROM*

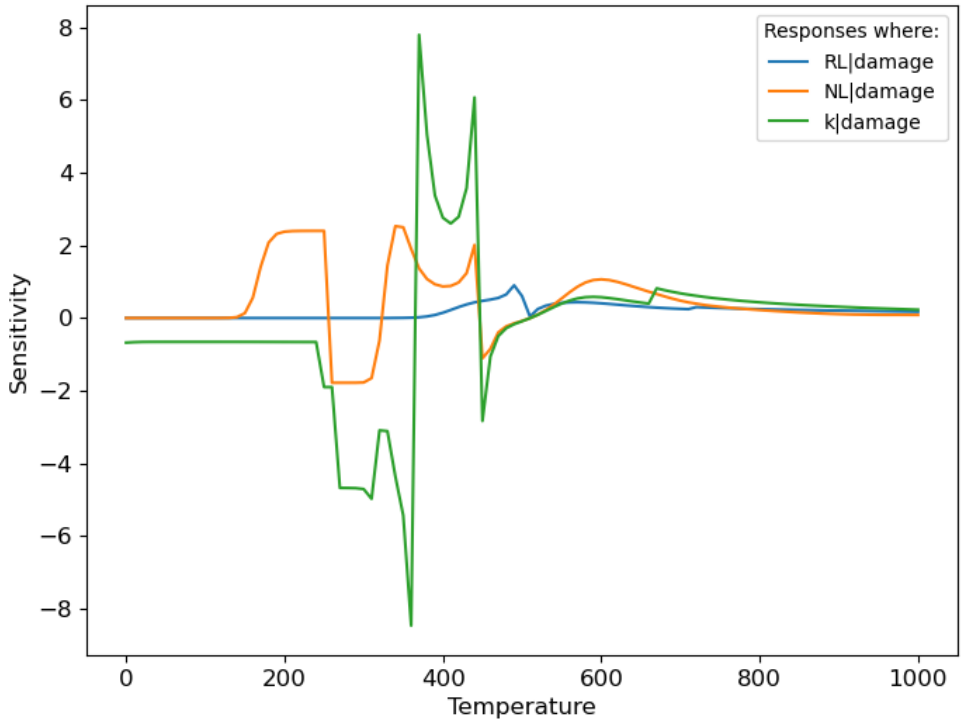


- Sparse grid stochastic collocation with Gaussian Polynomial Chaos expansions to accelerate experimental design
  - See *Generate Sparse Grid Locations*

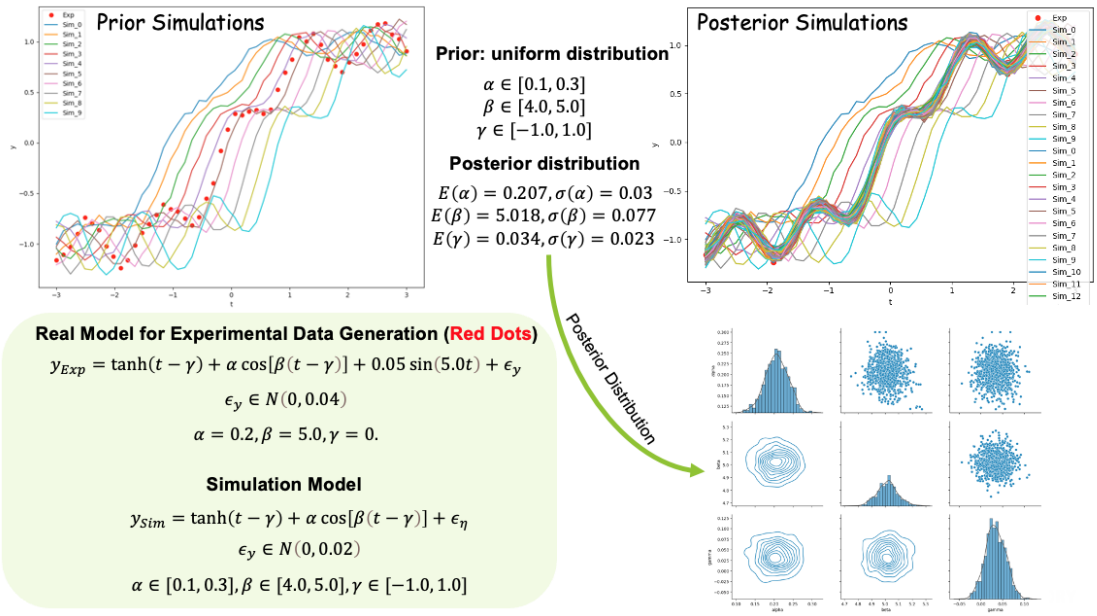




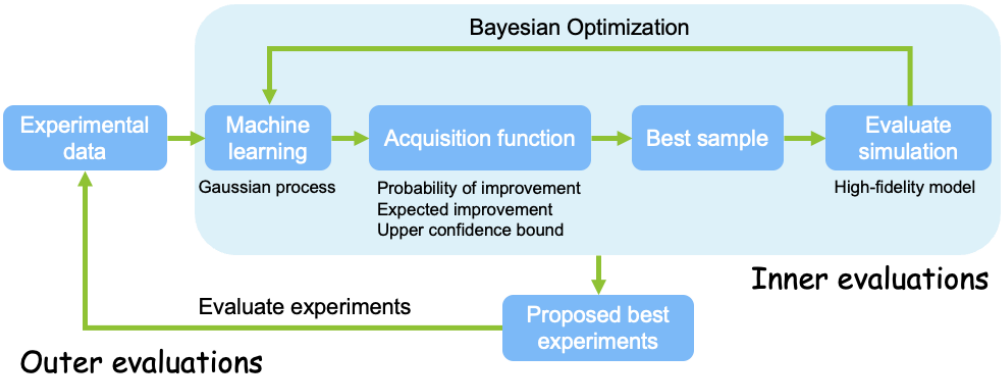
- Dynamic sensitivity and uncertainty analysis
  - See *Sensitivity Analysis*

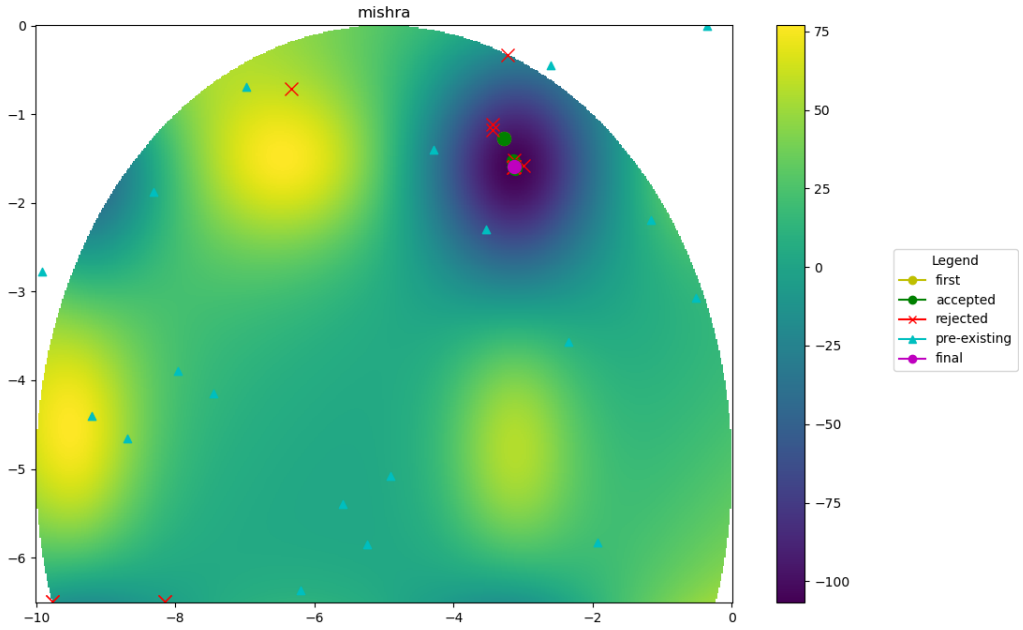


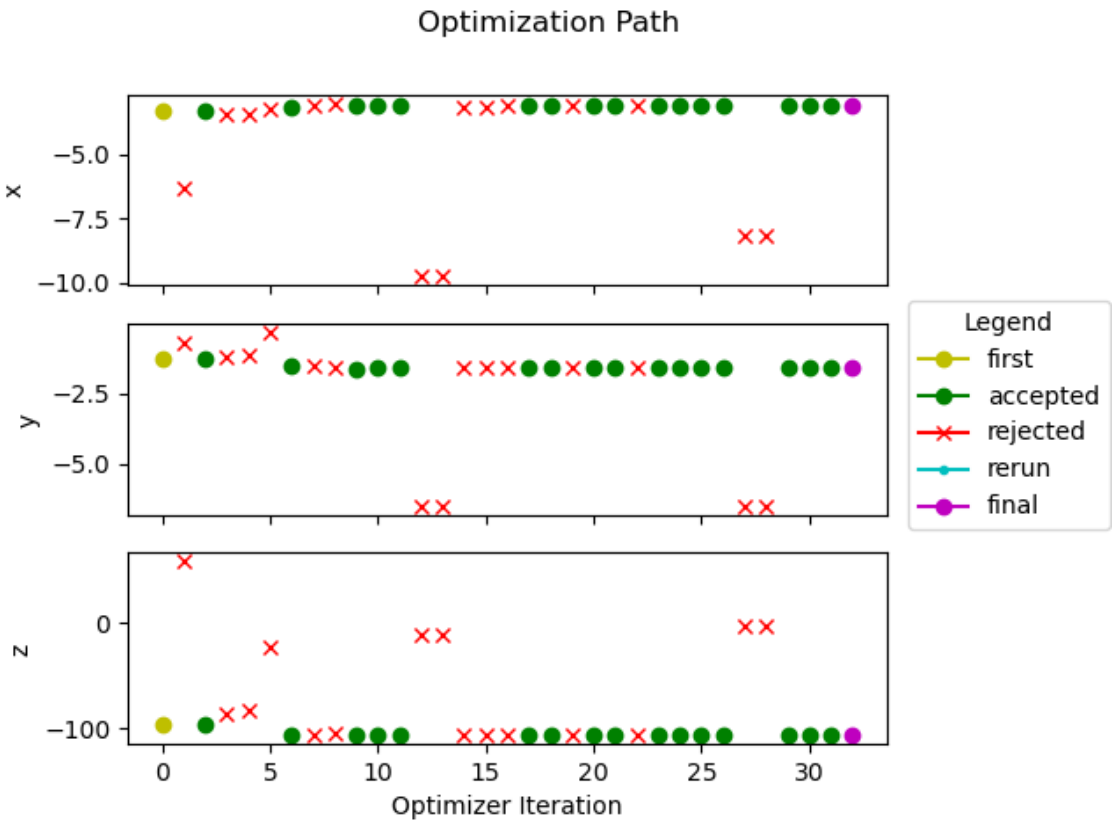
- Model calibration through Bayesian inference
  - See *Model Calibration*



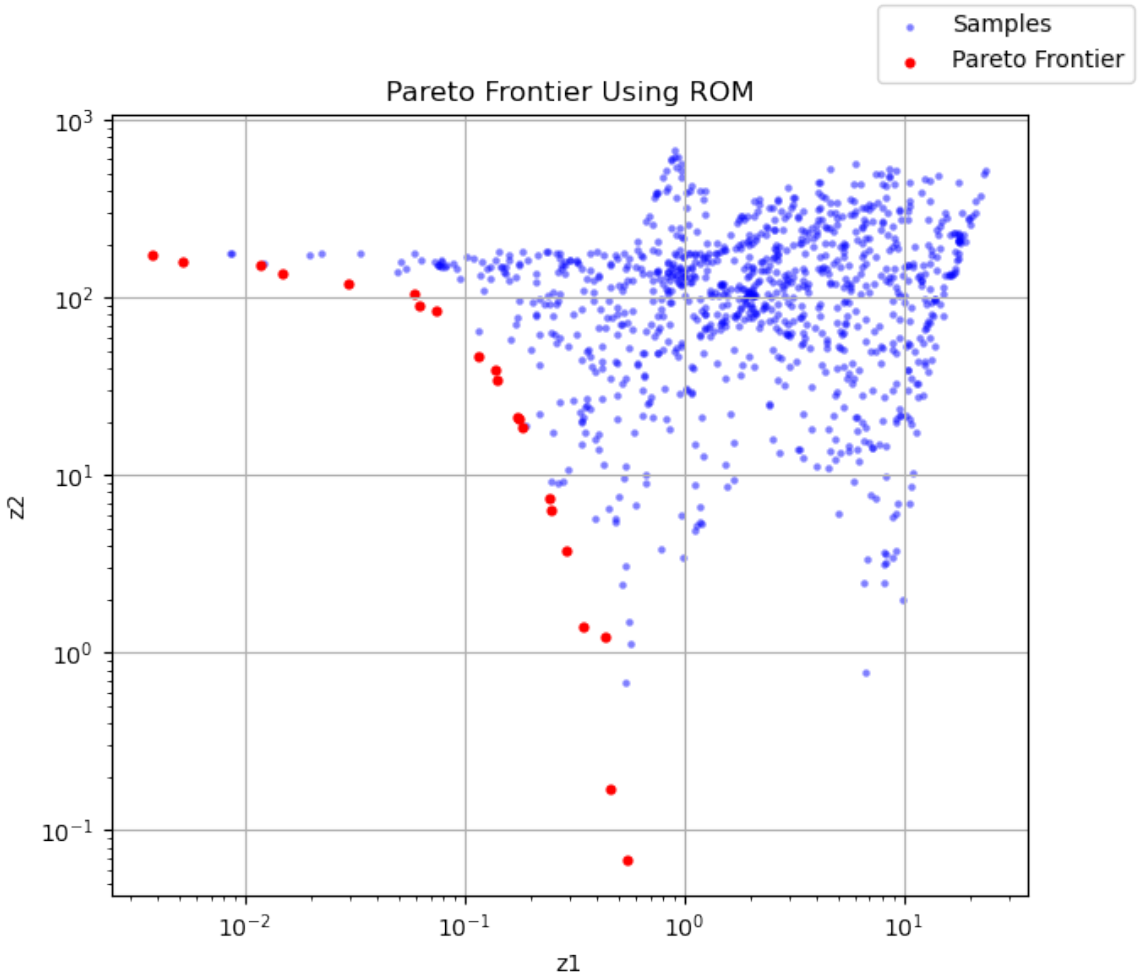
- Bayesian optimization for optimal experimental design
  - See *Bayesian Optimization*


Bayesian Optimization
Experimental data
Machine learning
Acquisition function
Best sample
Evaluate simulation
Gaussian process
Probability of improvement
Expected improvement
Upper confidence bound
High-fidelity model
Inner evaluations
Evaluate experiments
Proposed best experiments
Outer evaluations


mishra
0
−1
−2
−3
−4
−5
−6
−10
−8
−6
−4
−2
0
75
50
25
0
−25
−50
−75
−100
Legend
first
accepted
rejected
pre-existing
final



- Pareto Frontier to guide the design of experiment to be evaluated


Samples
Pareto Frontier
Pareto Frontier Using ROM
10^3
10^2
10^1
10^0
10^{-1}
z2
10^{-2}
10^{-1}
10^0
10^1
z1

CHAPTER

TWO

INSTALLATION

2.1 How to install?

2.1.1 Installation

```
conda create -n poem_libs python=3.10
conda activate poem_libs
pip install raven-framework baycal-ravenframework
```

2.1.2 Test

```
cd POEM/tests
python ../poem.py -i lhs_sampling.xml
raven_framework raven_lhs_sampling.xml
```

CHAPTER

THREE

QUICK START

3.1 Input Structure

POEM utilizes XML to define its input structure. The main input blocks are as follows:

3.1.1 Simulation block

The root node containing the entire input, all of the following blocks fit inside the `Simulaiton` block

```
<Simulation>

  <RunInfo>
    ...
  </RunInfo>

  <GlobalSettings>
    ...
  </GlobalSettings>

  <Distributions>
    ...
  </Distributions>

  <Models>
    ...
  </Models>

  <Files>
    ...
  </Files>

  <Functions>
    ...
  </Functions>

  <LikelihoodModel>
    ...
  </LikelihoodModel>

</Simulation>
```

3.1.2 GlobalSettings block

Specifies the global settings for the calculations. For example:

```
<GlobalSettings>
  <!-- Required Nodes -->
  <AnalysisType>LHS</AnalysisType>
  <limit>10</limit>
  <Inputs>x, y</Inputs>
  <Outputs>OutputPlaceHolder</Outputs>

  <!-- Optional Nodes -->
  <pivot>time</pivot>
  <dynamic>True</dynamic>

  <!-- Optional Nodes, uses for certain analysis -->
  <SparseGridData>path/to/data.csv</SparseGridData>
  <data>path/to/data.csv</data>
  <InitialInputs>0.1, 4.0, -1.0</InitialInputs>
  <PolynomialOrder>3</PolynomialOrder>
</GlobalSettings>
```

In general, this block accepts the following subnodes:

- Required Nodes
  - `AnalysisType`: The type of analysis, it accepts the following keywords:
    - `mc`: Simple Monte Carlo analysis for given model. See *Monte Carlo Sampling*.
    - `lhs`: Sample given model using Latin Hyper-cube Sampling (LHS) strategy. See *Latin Hypercube Sampling*.
    - `sensitivity`: Perform sensitivity analysis for given model. The `mean, variance, 95/95 percentile, correlation, spearman correlation, sensitivity coefficients, etc.` will be computed. See *Sensitivity Analysis*.
    - `sparse_grid_construction`: Generate sparse grid locations to guide experiments. These locations can be used to efficiently construct high-order Gaussian Polynomial Chaos surrogate model. See *Generate Sparse Grid Locations*.
    - `sparse_grid_rom`: Train a multi-variate high-order Gaussian Polynomial Chaos ROM/surrogate based on user provided experimental data. See *Train ROM*.
    - `train_rom`: Train a Gaussian Process ROM based on user provided data. See *Train ROM*.
    - `bayesian_optimization`: Perform Bayesian optimization based on user provided data and simulation model. See *Bayesian Optimization*.
    - `model_calibration`: Perform model calibration utilizing Bayesian inference based on user provided data and simulation model. See *Model Calibration*.
  - `limit`: The total number of model executions or the number of samples to generate.
  - `Inputs`: The list of input variables
  - `Outputs`: The list of output variables. If no output variables, `OutputPlaceHolder` can be used.
- Optional Nodes
  - `dynamic`: True if the user wants to perform time-dependent analysis, such as time-dependent ROM construction, sensitivity analysis, model calibration etc.

  - `pivot`: Required if `dynamic` is True. The pivot variable for dynamic analysis. Default is `time`.

- Optional Nodes for Certain Analysis
  - `SparseGridData`: The experimental data that can be used to train Gaussian Polynomial Chaos ROM. Only used by `sparse_grid_construction` and `sparse_grid_rom`.
  - `PolynomialOrder`: The highest order for the Gaussian Polynomial Chaos ROM. Only used by `sparse_grid_construction` and `sparse_grid_rom`
  - `data`: The experimental data that can be used to train Gaussian Process ROM. Only used by `train_rom` and `bayesian_optimization`.
  - `InitialInputs`: The initial values for the input variables listed by <Inputs> in the <GlobalSettings>

3.1.3 RunInfo block

Specifies the calculation settings (woring directory, number of parallel simulations, etc.)

```
<RunInfo>
  <WorkingDir>LHS</WorkingDir>
  <batchSize>1</batchSize>
</RunInfo>
```

In general, this block accepts the following subnodes:

- `WorkingDir`: specifies the absolute or relative path to a directory that will store all the results of the calculations.
- `batchSize`: specifies the number of parallel executed simultaneously.
- `JobName`: specifies the name to use for the job when submitting to a pbs queue.

RunInfo for Cluster Usage

```
<RunInfo>
  <WorkingDir>FirstMF</WorkingDir>
  <batchSize>3</batchSize>
  <clusterParameters>-W block=true</clusterParameters>
  <NumThreads>4</NumThreads>
  <mode>
    mpi
    <runQSUB/>
  </mode>
  <NodeParameter> </NodeParameter>
  <NumMPI>2</NumMPI>
  <expectedTime>0:10:00</expectedTime>
  <JobName>test_qsub</JobName>
</RunInfo>
```

3.1.4 Files block

Specifies the files to be used for the <Models> block as input. Users can specify as many input files as they need, and utilize <Input> node to specify the `name`, and the `path/to/file`.

```
<Files>
  <Input name="sauq" type="">../../models/sauq.m</Input>
  <Input name="rt" type="">../../models/RateTheory.m</Input>
  <Input name="kc" type="">../../models/KlemensCallawayModel.m</Input>
</Files>
```

3.1.5 Distributions block

POEM leverages RAVEN (https://github.com/idaholab/raven) input structure to build customized workflows for model explorations and optimal experiment design. In this case, POEM provides support for all the probability distributions available in RAVEN. The following are the example for the *Distributions* block.

```
<Distributions>
  <Uniform name='x'>
    <lowerBound>-10</lowerBound>
    <upperBound>0</upperBound>
  </Uniform>
  <Uniform name='y'>
    <lowerBound>-6.5</lowerBound>
    <upperBound>0</upperBound>
  </Uniform>
</Distributions>
```

In this block, the users need to define `distribution` for each variables listed in `GlobalSettings` `Inputs` node, and `name` for the distribution should match the variable name listed under `<GlobalSettings><Inputs>VariableList</Inputs></GlobalSettings>`.

3.1.6 Models block

Similar to `<Distributions>` block, POEM leverages RAVEN (https://github.com/idaholab/raven) `<Models>` input structure. In this case, POEM provides support for all the models available in RAVEN. The following are the example for the *Models* block.

```
<Models>
  <ExternalModel ModuleToLoad="../../models/mishraBirdConstrained.py" name="mishra"␣
↪subType="">
    <inputs>x, y</inputs>
    <outputs>z</outputs>
  </ExternalModel>
</Models>
```

As the name suggests, an external model is an entity that is embedded at run time. This object allows the user to create a python module that is going to be treated as a predefined internal model object.

The specifications of an External Model must be defined within the XML block `<ExternalModel>`. This blocks accepts the following subnodes:

- `inputs`: Each variable name needs to match a variable used/defined in the external python model.

- `outputs`: Each variable name needs to match a variable used/defined in the external python model.

Each variable defined in the <ExternalModel> <inputs> and <outputs> block is available in the module (each method implemented) as a python `self.` member.

3.1.7 Functions block

POEM leverages RAVEN (https://github.com/idaholab/raven) <Functions> input structure. In this case, POEM provides support for the usage of user-defined external functions. These functions are python modules, with a format is automatically interpretable by RAVEN software.

The following are the example for the *Functions* block.

```
<Functions>
  <External file="../../models/mishraBirdConstrained.py" name="constraint1">
    <variables>x,y</variables>
  </External>
</Functions>
```

In this section, the XML input syntax and the format of the accepted functions are fully specified. The specifications of an external function must be defined within the XML <External> block. This XML node requires the following attributes:

- `name`: user-defined name of this function.
- `file`: absolute or relative path specifying the code associated to this function.

In order to make the code aware of the variables the user is going to manipulate/use in her/his own python function, the variables need to be specified in the <variables> subnode input block. The user needs to input, within this block, only the variables directly used by the external function.

When the external function variables are defined, at runtime, the code initializes them and keeps track of their values during the simulation. Each variable defined in the <variables> block is available in the function as a python **self.** member. In the following, an example of a user-defined external function is reported. The method `evaluate` needs to be defined in the function file.

```
def evaluate(self):
  return self.a * self.c
```

3.1.8 LikelihoodModel block for Model Calibration

This node is only used by model calibration analysis. An example is presented:

```
<LikelihoodModel>
  <simTargets>eta</simTargets>
  <expTargets shape="1,50" computeCov='False' correlation='False'>
    -1.16074224 -1.10303445 -1.02830511 -0.89782965 -0.73765453 -0.7989537
     -0.86163706 -1.02209944 -1.12444044 -1.23657398 -1.16081758 -1.01219869
     -0.890747   -0.80444122 -0.70893668 -0.61012531 -0.65670863 -0.6768583
     -0.74732441 -0.81448647 -0.73232671 -0.54989334 -0.39796749 -0.07894291
      0.13067378  0.28999998  0.27418965  0.313329    0.32306704  0.2885684
      0.32736775  0.52458854  0.69446572  0.82419521  1.04393683  1.00435818
      1.0810376   0.97245373  0.82406522  0.76067559  0.70145544  0.79479965
      0.88035895  0.97750307  1.11524353  1.17159017  1.18299222  1.07255006
```

(continues on next page)

(continued from previous page)

```
      1.02835909  0.90784132
  </expTargets>
  <expCov diag="True">
       0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02,
       0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02,
       0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02,
       0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02,
       0.02, 0.02, 0.02, 0.02, 0.02, 0.02
  </expCov>
  <!-- <biasTargets></biasTargets>
  <biasCov diag="False"></biasCov> -->
  <!-- <romCov diag="True"></romCov> -->
</LikelihoodModel>
```

The <LikelihoodModel> node accepts the following subnodes:

- simTargets: Targets of simulations that are used in the calibration.
- expTargets: Targets of experiments that are used in the calibration. Either variables or list of values. This node accepts the following attributes:
  - shape: determine the number of targets and the number of experimental observations for each targets. For example, shape="3,2" will indicate 2 targets and 3 observations for each targets. While shape="10" will indicate one target with 10 observations. Omitting this optional attribute will result a single target with multiple observations instead.
  - computeCov: Indicate whether the experiment covariance matrix is provided or computed based on given experiment observations. If True, we will compute the covariance based on given observations, else, the user need to provide the covariance matrix.
  - correlation: Indicate whether the targets are correlated or not. If True, and compute is True, we will compute the covariance matrix, elif False and compute is True, we will only compute the variance of each target.
- expCov: Experiment covariance, i.e. measurement noise. This node accepts the following attribute:
  - diag: If True, only variance for each target is required to provide, else, the user need to provide the full covariance matrix.
- biasTargets: Model uncertainty/discrepancy/bias/error in Targets that are used in calibration
- biasCov: Model covariance, model bias/discrepancy or model inadequacy caused by missing physics or numerical approximation. This node accepts the following attribute:
  - diag: If True, only variance for each target is required to provide, else, the user need to provide the full covariance matrix.
- romCov: Model uncertainty caused by surrogate model, such as interpolation. This node accepts the following attribute:
  - diag: If True, only variance for each target is required to provide, else, the user need to provide the full covariance matrix.
- reduction: Allows reduction on likelihood model construction. This node accepts the following attributes:
  - type: The method used for reduction, default is **PCA**
  - basis: user provided basis vector for reduction
  - shape: determine the basis vectors for reduction. For example, shape="10,2" will indicate 2 basis vectors with dimension 10

CHAPTER

FOUR

MONTE CARLO SAMPLING

Utilize Monte Carlo Sampling to perform random model explorations for experiment design. In this analysis, set <AnalysisType>MC</AnalysisType>. For example, The following input will utilize Monte Carlo Sampling to generate 10 random samples of input variables x, y with associated Uniform distributions, respectively.

```
<?xml version="1.0" ?>
<Simulation>
  <RunInfo>
    <WorkingDir>MC</WorkingDir>
    <batchSize>1</batchSize>
  </RunInfo>

  <GlobalSettings>
    <AnalysisType>MC</AnalysisType>
    <limit>10</limit>
    <Inputs>x, y</Inputs>
    <Outputs>OutputPlaceHolder</Outputs>
  </GlobalSettings>

  <Distributions>
    <Uniform name='x'>
      <lowerBound>-10</lowerBound>
      <upperBound>0</upperBound>
    </Uniform>
    <Uniform name='y'>
      <lowerBound>-6.5</lowerBound>
      <upperBound>0</upperBound>
    </Uniform>
  </Distributions>

</Simulation>
```

When a dynamic model is provided, the users need to set <pivot> and <dynamic> node in the <GlobalSettings>. As illustrated in the following example.

```
<?xml version="1.0" ?>
<Simulation>
  <RunInfo>
    <WorkingDir>MC</WorkingDir>
    <batchSize>1</batchSize>
  </RunInfo>
```

(continues on next page)

(continued from previous page)

```
  <GlobalSettings>
    <AnalysisType>MC</AnalysisType>
    <limit>10</limit>
    <Inputs>x0, y0, z0</Inputs>
    <pivot>time</pivot>
    <dynamic>True</dynamic>
    <Outputs>x,y,z</Outputs>
  </GlobalSettings>

  <Distributions>
    <Normal name="x0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
    <Normal name="y0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
    <Normal name="z0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
  </Distributions>

  <Models>
    <ExternalModel ModuleToLoad="../models/lorentzAttractor.py" name="lorentzAttractor"␣
↪subType="">
      <inputs>inputGroup</inputs>
      <outputs>outputGroup</outputs>
    </ExternalModel>
  </Models>

</Simulation>
```

In addition, the users can use `inputGroup` and `outputGroup` to represent input and output variable list. As illustrated in above example <`ExternalModel`> node.

CHAPTER

FIVE

LATIN HYPERCUBE SAMPLING

Utilize Latin Hypercube Sampling (LHS) to perform random model explorations for experiment design. In this analysis, set <AnalysisType>LHS</AnalysisType>. For example, The following input will utilize LHS to generate 10 random LHS samples of input variables x, y with associated Uniform distributions, respectively.

```
<?xml version="1.0" ?>
<Simulation>
  <RunInfo>
    <WorkingDir>LHS_mishra</WorkingDir>
    <batchSize>1</batchSize>
  </RunInfo>

  <GlobalSettings>
    <AnalysisType>LHS</AnalysisType>
    <limit>10</limit>
    <Inputs>x, y</Inputs>
    <Outputs>z</Outputs>
  </GlobalSettings>

  <Distributions>
    <Uniform name='x'>
      <lowerBound>-10</lowerBound>
      <upperBound>0</upperBound>
    </Uniform>
    <Uniform name='y'>
      <lowerBound>-6.5</lowerBound>
      <upperBound>0</upperBound>
    </Uniform>
  </Distributions>

  <Models>
    <ExternalModel ModuleToLoad="../../models/mishraBirdConstrained" name="mishra"␣
↪subType="">
      <inputs>x, y</inputs>
      <outputs>z</outputs>
    </ExternalModel>
  </Models>

</Simulation>
```

When a dynamic model is provided, the users need to set <pivot> and <dynamic> node in the <GlobalSettings>. As illustrated in the following example.

```
<?xml version="1.0" ?>
<Simulation>
  <RunInfo>
    <WorkingDir>LHS</WorkingDir>
    <batchSize>1</batchSize>
  </RunInfo>

  <GlobalSettings>
    <AnalysisType>LHS</AnalysisType>
    <limit>10</limit>
    <Inputs>x0, y0, z0</Inputs>
    <pivot>time</pivot>
    <dynamic>True</dynamic>
    <Outputs>x,y,z</Outputs>
  </GlobalSettings>

  <Distributions>
    <Normal name="x0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
    <Normal name="y0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
    <Normal name="z0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
  </Distributions>

  <Models>
    <ExternalModel ModuleToLoad="../models/lorentzAttractor.py" name="lorentzAttractor"␣
↪subType="">
      <inputs>inputGroup</inputs>
      <outputs>outputGroup</outputs>
    </ExternalModel>
  </Models>

</Simulation>
```

In addition, the users can use `inputGroup` and `outputGroup` to represent input and output variable list. As illustrated in above example <`ExternalModel`> node.

CHAPTER

SIX

TRAIN ROM

Train reduced order models (ROM) or machine learning models based on experiment data, or mixed experiment data and simulation data.

6.1 Train Gaussian Process Model

In this analysis, set <AnalysisType>train_rom</AnalysisType>. For example, The following input will utilize data provided by <data> node to train Gaussian Process Model. After training, the inputs of the Gaussian Process Model will be sampled via Latin Hypercube Sampling algorithm with their associated distributions, and the number of samples is equal to <limit> value.

```
<?xml version="1.0" ?>
<Simulation>
  <RunInfo>
    <WorkingDir>GP</WorkingDir>
    <batchSize>1</batchSize>
  </RunInfo>

  <GlobalSettings>
    <AnalysisType>train_rom</AnalysisType>
    <data>../LHS/sampling_dump.csv</data>
    <limit>10</limit>
    <Inputs>x, y</Inputs>
    <Outputs>z1</Outputs>
  </GlobalSettings>

  <Distributions>
    <Uniform name='x'>
      <lowerBound>-10</lowerBound>
      <upperBound>0</upperBound>
    </Uniform>
    <Uniform name='y'>
      <lowerBound>-6.5</lowerBound>
      <upperBound>0</upperBound>
    </Uniform>
  </Distributions>

</Simulation>
```

6.2 Train Dynamic Gaussian Process Model

When a dynamic model is provided, the users need to set <pivot> and <dynamic> node in the <GlobalSettings>. As illustrated in the following example. In addition, set <AnalysisType>train_rom</AnalysisType>. For example, The following input will utilize data provided by <data> node to train Gaussian Process Model. After training, the inputs of the Gaussian Process Model will be sampled via Latin Hypercube Sampling algorithm with their associated distributions, and the number of samples is equal to <limit> value.

```
<?xml version="1.0" ?>
<Simulation>
  <RunInfo>
    <WorkingDir>GP_dynamic</WorkingDir>
    <batchSize>1</batchSize>
  </RunInfo>

  <GlobalSettings>
    <AnalysisType>train_rom</AnalysisType>
    <data>../LHS/sampling_dynamic_dump.csv</data>
    <limit>10</limit>
    <Inputs>x0, y0, z0</Inputs>
    <pivot>time</pivot>
    <dynamic>True</dynamic>
    <Outputs>x,y,z</Outputs>
  </GlobalSettings>

  <Distributions>
    <Normal name="x0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
    <Normal name="y0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
    <Normal name="z0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
  </Distributions>

  <Models>
    <ExternalModel ModuleToLoad="../models/lorentzAttractor.py" name="lorentzAttractor"␣
↪subType="">
      <inputs>inputGroup</inputs>
      <outputs>outputGroup</outputs>
    </ExternalModel>
  </Models>

</Simulation>
```

6.3 Train Gaussian Polynomial Chaos Model with Sparse Grid

In this analysis, set <AnalysisType>sparse_grid_rom</AnalysisType>. For example, The following input will utilize data provided by <SparseGridData> node to train Gaussian Polynomial Chaos Model. After training, the inputs of the Gaussian Process Model will be sampled via Monte Carlo Sampling algorithm with their associated distributions, and the number of samples is equal to <limit> value.

```
<?xml version="1.0" ?>
<Simulation>

  <GlobalSettings>
    <AnalysisType>sparse_grid_rom</AnalysisType>
    <Inputs>x, y</Inputs>
    <Outputs>z1</Outputs>
    <PolynomialOrder>2</PolynomialOrder>
    <limit>10</limit>
    <SparseGridData>dump_SparseGrid.csv</SparseGridData>
  </GlobalSettings>

  <Distributions>
    <Uniform name="x">
      <lowerBound>-10</lowerBound>
      <upperBound>10</upperBound>
    </Uniform>
    <Uniform name="y">
      <lowerBound>-10</lowerBound>
      <upperBound>10</upperBound>
    </Uniform>
  </Distributions>

  <Models>
    <ExternalModel ModuleToLoad="../models/matyas" name="externalModel" subType="">
      <inputs>x, y</inputs>
      <outputs>z1</outputs>
    </ExternalModel>
  </Models>

</Simulation>
```

For this ROM, the users can also set the highest order of the Gaussian Polynomial Chaos expansions. Just set <PolynomialOrder> in <GlobalSettings>.

6.4 Train Dynamic Gaussian Polynomial Chaos Model with Sparse Grid

When a dynamic model is provided, the users need to set <pivot> and <dynamic> node in the <GlobalSettings>. As illustrated in the following example. In this analysis, set <AnalysisType>sparse_grid_rom</AnalysisType>. For example, The following input will utilize data provided by <SparseGridData> node to train Gaussian Polynomial Chaos Model. After training, the inputs of the Gaussian Process Model will be sampled via Monte Carlo Sampling algorithm with their associated distributions, and the number of samples is equal to <limit> value.

```
<?xml version="1.0" ?>
<Simulation>
  <RunInfo>
    <WorkingDir>SparseGridDynamic</WorkingDir>
    <batchSize>1</batchSize>
  </RunInfo>

  <GlobalSettings>
    <AnalysisType>sparse_grid_rom</AnalysisType>
    <Inputs>x0, y0, z0</Inputs>
    <pivot>time</pivot>
    <dynamic>True</dynamic>
    <Outputs>x,y,z</Outputs>
    <PolynomialOrder>2</PolynomialOrder>
    <limit>10</limit>
    <SparseGridData>../SparseGrid/SparseGrid_dynamic_dump.csv</SparseGridData>
  </GlobalSettings>

  <Distributions>
    <Normal name="x0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
    <Normal name="y0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
    <Normal name="z0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
  </Distributions>

  <Models>
    <ExternalModel ModuleToLoad="../models/lorentzAttractor.py" name="lorentzAttractor"␣
↪subType="">
      <inputs>inputGroup</inputs>
      <outputs>outputGroup</outputs>
    </ExternalModel>
  </Models>

</Simulation>
```

For this ROM, the users can also set the highest order of the Gaussian Polynomial Chaos expansions. Just set <PolynomialOrder> in <GlobalSettings>.

CHAPTER

SEVEN

GENERATE SPARSE GRID LOCATIONS

Sparse grid model explorations with Gaussian Polynomial Chaos surrogate model to accelerate experiment design:

When generating sparse grid locations, the users can also set the highest order of the Gaussian Polynomial Chaos expansions. The higher of the Polynomial order, the more sparse grid locations will be generated. Just set <PolynomialOrder> in <GlobalSettings>.

Once the experiment data are generated at given sparse grid locations, the Gaussian Polynomial Chaos surrogate can be automatically constructed. See the Train ROM section on how to train Gaussian Polynomial Chaos surrogate model.

7.1 Static Model

```
<Simulation>
  <RunInfo>
    <WorkingDir>SparseGrid</WorkingDir>
    <Sequence>SparseGridSampler, print</Sequence>
    <batchSize>1</batchSize>
  </RunInfo>

  <GlobalSettings>
    <AnalysisType>sparse_grid_construction</AnalysisType>
    <Inputs>x, y</Inputs>
    <Outputs>z1</Outputs>
    <PolynomialOrder>3</PolynomialOrder>
  </GlobalSettings>

  <Distributions>
    <Uniform name="x">
      <lowerBound>-10</lowerBound>
      <upperBound>10</upperBound>
    </Uniform>
    <Uniform name="y">
      <lowerBound>-10</lowerBound>
      <upperBound>10</upperBound>
    </Uniform>
  </Distributions>

  <Models>
    <ExternalModel ModuleToLoad="../models/matyas" name="externalModel" subType="">
      <inputs>x, y</inputs>
```

(continues on next page)

(continued from previous page)

```
      <outputs>z1</outputs>
    </ExternalModel>
  </Models>

</Simulation>
```

7.2 Dynamic Model

When a dynamic model is provided, the users need to set <pivot> and <dynamic> node in the <GlobalSettings>. As illustrated in the following example.

```
<?xml version="1.0" ?>
<Simulation>
  <RunInfo>
    <WorkingDir>SparseGrid</WorkingDir>
    <Sequence>SparseGridSampler, print</Sequence>
    <batchSize>1</batchSize>
  </RunInfo>

  <GlobalSettings>
    <AnalysisType>sparse_grid_construction</AnalysisType>
    <Inputs>x0, y0, z0</Inputs>
    <pivot>time</pivot>
    <dynamic>True</dynamic>
    <Outputs>x,y,z</Outputs>
    <PolynomialOrder>3</PolynomialOrder>
  </GlobalSettings>

  <Distributions>
    <Normal name="x0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
    <Normal name="y0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
    <Normal name="z0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
  </Distributions>

  <Models>
    <ExternalModel ModuleToLoad="../models/lorentzAttractor.py" name="lorentzAttractor"␣
↪subType="">
      <inputs>inputGroup</inputs>
      <outputs>outputGroup</outputs>
    </ExternalModel>
  </Models>
```

(continues on next page)

(continued from previous page)

```
</Simulation>
```

CHAPTER

EIGHT

SENSITIVITY ANALYSIS

8.1 Static Sensitivity Analysis

In this analysis, set `<AnalysisType>sensitivity</AnalysisType>`. A Monte Carlo method will be utilized to sample the given model with `limit` number.

```
<?xml version="1.0" ?>
<Simulation>
  <RunInfo>
    <JobName>sauq</JobName>
    <WorkingDir>sauq_runs</WorkingDir>
    <batchSize>1</batchSize>
  </RunInfo>

  <GlobalSettings>
    <AnalysisType>sensitivity</AnalysisType>
    <limit>10</limit>
    <Inputs>x, y</Inputs>
    <Outputs>z1</Outputs>
  </GlobalSettings>

  <Distributions>
    <Uniform name='x'>
      <lowerBound>-10</lowerBound>
      <upperBound>0</upperBound>
    </Uniform>
    <Uniform name='y'>
      <lowerBound>-6.5</lowerBound>
      <upperBound>0</upperBound>
    </Uniform>
  </Distributions>

  <Models>
    <ExternalModel ModuleToLoad="../models/matyas" name="externalModel" subType="">
      <inputs>x, y</inputs>
      <outputs>z1</outputs>
    </ExternalModel>
  </Models>

</Simulation>
```

8.2 Dynamic Sensitivity Analysis

Time-dependent model sensitivity and uncertainty analysis to identify the importance features for experiment design: In this analysis, set <AnalysisType>sensitivity</AnalysisType>. A Monte Carlo method will be utilized to sample the given model with `limit` number. When a dynamic model is provided, the users need to set <pivot> and <dynamic> node in the <GlobalSettings>. As illustrated in the following example.

```
<?xml version="1.0" ?>
<Simulation>
  <RunInfo>
    <JobName>sauq</JobName>
    <WorkingDir>sauq_dynamic_external</WorkingDir>
    <batchSize>1</batchSize>
  </RunInfo>

  <GlobalSettings>
    <AnalysisType>sensitivity</AnalysisType>
    <limit>10</limit>
    <Inputs>x0, y0, z0</Inputs>
    <pivot>time</pivot>
    <dynamic>True</dynamic>
    <Outputs>x,y,z</Outputs>
  </GlobalSettings>

  <Distributions>
    <Normal name="x0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
    <Normal name="y0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
    <Normal name="z0">
      <mean>4</mean>
      <sigma>1</sigma>
    </Normal>
  </Distributions>

  <Models>
    <ExternalModel ModuleToLoad="../models/lorentzAttractor.py" name="lorentzAttractor"␣
↪subType="">
      <inputs>inputGroup</inputs>
      <outputs>outputGroup</outputs>
    </ExternalModel>
  </Models>

</Simulation>
```

CHAPTER

NINE

BAYESIAN OPTIMIZATION

Bayesian optimization for optimal experimental design.

9.1 Example

In this analysis, set <AnalysisType>bayesian_optimization</AnalysisType>. The existing experiment data can be provided through <data> in <GlobalSettings>. For example:

```
<?xml version="1.0" ?>
<Simulation verbosity="debug">
  <RunInfo>
    <WorkingDir>Optimization</WorkingDir>
    <batchSize>1</batchSize>
  </RunInfo>

  <GlobalSettings>
    <AnalysisType>bayesian_optimization</AnalysisType>
    <data>../LHS_mishra/sampling_dump.csv</data>
    <limit>10</limit>
    <Inputs>x, y</Inputs>
    <Outputs>z</Outputs>
  </GlobalSettings>

  <Distributions>
    <Uniform name='x'>
      <lowerBound>-10</lowerBound>
      <upperBound>0</upperBound>
    </Uniform>
    <Uniform name='y'>
      <lowerBound>-6.5</lowerBound>
      <upperBound>0</upperBound>
    </Uniform>
  </Distributions>

  <Models>
    <ExternalModel ModuleToLoad="../../models/mishraBirdConstrained.py" name="mishra"␣
↪subType="">
      <inputs>x, y</inputs>
      <outputs>z</outputs>
    </ExternalModel>
```

(continues on next page)

(continued from previous page)

```
  </Models>

  <Functions>
    <External file="../../models/mishraBirdConstrained.py" name="constraint1">
      <variables>x,y</variables>
    </External>
  </Functions>

</Simulation>
```

9.2 Python External Model and Constrain

```
import numpy as np

def evaluate(x,y):
  """
    Evaluates Mishra bird function.
    @ In, x, float, value
    @ In, y, float, value
    @ Out, evaluate, value at x, y
  """
  evaluate = np.sin(y)*np.exp(1.-np.cos(x))**2 + np.cos(x)*np.exp(1.-np.sin(y))**2 + (x-
↪y)**2
  return evaluate

def constraint(x,y):
  """
    Evaluates the constraint function @ a given point (x,y)
    @ In, x, float, value of the design variable x
    @ In, y, float, value of the design variable y
    @ Out, g(x,y), float, $g(x, y) = 25 - ((x+5.)**2 + (y+5.)**2)$
            the way the constraint is designed is that
            the constraint function has to be >= 0,
            so if:
            1) f(x,y) >= 0 then g = f
            2) f(x,y) >= a then g = f - a
            3) f(x,y) <= b then g = b - f
            4) f(x,y)  = c then g = 0.001 - (f(x,y) - c)
  """
  condition = 25.
  g = condition - ((x+5.)**2 + (y+5.)**2)
  return g

###
# RAVEN hooks
###

def run(self,Inputs):
  """
```

(continues on next page)

(continued from previous page)

```
    RAVEN API
    @ In, self, object, RAVEN container
    @ In, Inputs, dict, additional inputs
    @ Out, None
  """
  self.z = evaluate(self.x,self.y)

def constrain(self):
  """
    Constrain calls the constraint function.
    @ In, self, object, RAVEN container
    @ Out, explicitConstrain, float, positive if the constraint is satisfied
          and negative if violated.
  """
  explicitConstrain = constraint(self.x,self.y)
  return explicitConstrain
```

CHAPTER

TEN

MODEL CALIBRATION

Model calibrations via Bayesian inference to integrate experiments to improve model performance: In this analysis, set <AnalysisType>model_calibration</AnalysisType>. When a dynamic model is provided, the users need to set <pivot> and <dynamic> node in the <GlobalSettings>. As illustrated in the following example. In addition, the initial values for input variables can be provided via <InitialInputs>.

```
<?xml version="1.0" ?>
<Simulation verbosity="debug">

  <RunInfo>
    <WorkingDir>calibration_dynamic</WorkingDir>
    <batchSize>1</batchSize>
  </RunInfo>

  <GlobalSettings>
    <AnalysisType>model_calibration</AnalysisType>
    <pivot>time</pivot>
    <dynamic>True</dynamic>
    <limit>100</limit>
    <Inputs>alpha, beta, gamma</Inputs>
    <InitialInputs>0.1, 4.0, -1.0</InitialInputs>
    <Outputs>eta</Outputs>
  </GlobalSettings>

  <LikelihoodModel>
    <simTargets>eta</simTargets>
    <expTargets shape="1,50" computeCov='False' correlation='False'>
      -1.16074224 -1.10303445 -1.02830511 -0.89782965 -0.73765453 -0.7989537
      -0.86163706 -1.02209944 -1.12444044 -1.23657398 -1.16081758 -1.01219869
      -0.890747   -0.80444122 -0.70893668 -0.61012531 -0.65670863 -0.6768583
      -0.74732441 -0.81448647 -0.73232671 -0.54989334 -0.39796749 -0.07894291
        0.13067378  0.28999998  0.27418965  0.313329    0.32306704  0.2885684
        0.32736775  0.52458854  0.69446572  0.82419521  1.04393683  1.00435818
        1.0810376   0.97245373  0.82406522  0.76067559  0.70145544  0.79479965
        0.88035895  0.97750307  1.11524353  1.17159017  1.18299222  1.07255006
        1.02835909  0.90784132
    </expTargets>
    <expCov diag="True">
        0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02,
        0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02,
        0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02,
```

(continues on next page)

(continued from previous page)

```
        0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02,
        0.02, 0.02, 0.02, 0.02, 0.02, 0.02
    </expCov>
    <!-- <biasTargets></biasTargets>
    <biasCov diag="False"></biasCov> -->
    <!-- <romCov diag="True"></romCov> -->
  </LikelihoodModel>

  <Distributions>
    <Uniform name='alpha'>
      <lowerBound>0.1</lowerBound>
      <upperBound>0.3</upperBound>
    </Uniform>
    <Uniform name='beta'>
      <lowerBound>4</lowerBound>
      <upperBound>6</upperBound>
    </Uniform>
    <Uniform name='gamma'>
      <lowerBound>-1</lowerBound>
      <upperBound>1</upperBound>
    </Uniform>
  </Distributions>

  <Models>
    <ExternalModel ModuleToLoad="../models/model_cal" name="model" subType="">
      <inputs>inputGroup</inputs>
      <outputs>outputGroup</outputs>
    </ExternalModel>
  </Models>

</Simulation>
```

CHAPTER

ELEVEN

SUPPORT

The easiest way to get help with the project is to open an issue on Github.

The mailing list at … is also available for support.

11.1 Developer:

- Congjian, Wang: congjian.wang@inl.gov

CHAPTER

TWELVE

CONTRIBUTORS

CHAPTER

THIRTEEN

API REFERENCE

This page contains auto-generated API reference documentation[1].

13.1 src

13.1.1 Subpackages

13.1.1.1 src.poem

Subpackages

src.poem.templates

Package Contents

src.poem.templates.**configFileName = 'config.toml'**

src.poem.templates.**path**

src.poem.templates.**templateConfig**

Submodules

src.poem.PoemTemplate

Created on April 1, 2024 @author: wangc

This module inherits from base Template class for Input Templates, which use an established input template as an accelerated way to write new RAVEN workflows.

[1] Created with sphinx-autoapi

Module Contents

Classes

PoemTemplate	POEM: Template Class

Attributes

logger

src.poem.PoemTemplate.**logger**

class src.poem.PoemTemplate.**PoemTemplate**

Bases: ravenframework.InputTemplates.TemplateBaseClass.Template

POEM: Template Class

loadTemplate(*filename*)

Loads template file statefully. @ In, filename, str, name of file to load (xml) @ Out, None

createWorkflow(*inputs*, *miscDict*)

creates a new RAVEN workflow based on the information in dicitonary "inputs". @ In, inputs, dict, dictionary that contains xml node info that need to append, i.e. {RavenXMLNodeTag: ListOfNodes} @ In, miscDict, dict, dictionary that contains xml node text info that need to update, i.e. {RavenXMLNodeTag: value} @ Out, xml.etree.ElementTree.Element, modified copy of template ready to run

writeWorkflow(*template*, *destination*, *run=False*)

Writes a template to file. @ In, template, xml.etree.ElementTree.Element, file to write @ In, destination, str, path and filename to write to @ In, run, bool, optional, if True then run the workflow after writing? good idea? @ Out, errors, int, 0 if successfully wrote [and run] and nonzero if there was a problem

runWorkflow(*destination*)

Runs the workflow at the destination. @ In, destination, str, path and filename of RAVEN input file @ Out, res, int, system results of running the code

src.poem.PoemTemplateInterface

Created on April 1, 2024 @author: wangc

This module is the POEM Template interface, which use the user provided input XML file to construct the corresponding RAVEN workflows i.e. RAVEN input XML file.

Module Contents

Classes

PoemTemplateInterface	POEM Template Interface

Attributes

logger
fh
formatter
ravenFrameworkPath

src.poem.PoemTemplateInterface.**logger**

src.poem.PoemTemplateInterface.**fh**

src.poem.PoemTemplateInterface.**formatter**

src.poem.PoemTemplateInterface.**ravenFrameworkPath**

class src.poem.PoemTemplateInterface.**PoemTemplateInterface**(*filename*)

Bases: object

POEM Template Interface

uniformDistNode

samplerVarNode

externalModelNode

lhExternalModelNode

lhModelNode

reductionNode

basisNode

validAnalysis = ['sensitivity', 'sparse_grid_construction', 'sparse_grid_rom', 'lhs', 'mc', 'train_rom',...

analysisRequired

analysisOptions

getTemplateFile()

getOutput()

get the processed outputs from this class: PoemTemplateInterface @ In, None @ Out, (outputDict, miscDict), tuple, first dictionary contains the whole element that need to be appended in

the templated input, while the second dictionary contains only the values that need to be replaced.

readInput()

Read the POEM input files, and construct corresponding ET elements @ In, None @ Out, None

readGlobalSettings(*xmlNode*)

Read the RunInfo XML node from the input root @ In, xmlNode, xml.etree.ElementTree.Element, the input root xml @ Out, None

checkInput()

Check the consistency of user provided inputs @ In, None @ Out, None

static buildPointSet(*name*, *inputs*, *outputs*)

Build single PointSet XML node @ In, name, str, the name for the PointSet @ In, inputs, str, string that contains the list of input variables @ In, outputs, str, string that contains the list of output variables @ out, pointSet, xml.etree.ElementTree.Element, the constructed PointSet XML node

static buildHistorySet(*name*, *inputs*, *outputs*, *pivot='time'*)

Build single HistorySet XML node @ In, name, str, the name for the PointSet @ In, inputs, str, string that contains the list of input variables @ In, outputs, str, string that contains the list of output variables @ out, historySet, xml.etree.ElementTree.Element, the constructed HistorySet XML node

static buildPrint(*name*, *source*)

Build single OutStream Print XML node @ In, name, str, the name for the PointSet @ In, source, str, string that contains name of Data Object @ out, printObj, xml.etree.ElementTree.Element, the constructed print XML node

buildSamplerVariable(*inputs*, *distNode*, *limit=20*, *grid=False*, *init=None*)

build the sampler variable block

static buildSparseGridSampler(*name='SparseGrid'*)

static buildMonteCarloSampler(*name*, *limit*)

static buildVariableGroup(*name*, *varList*)

Build variable group node @ In, name, str, the name for the variable group @ In, varList, list, list of variables @ out, group, xml.etree.ElementTree.Element, the constructed variable group XML node

buildStatsGroup(*name*, *inputs*, *outputs*)

Build basic statistic variable group node @ In, name, str, the name for the variable group @ In, inputs, list, list of input variables @ In, outputs, list, list of output variables @ out, group, xml.etree.ElementTree.Element, the constructed variable group XML node

static findRequiredNode(*xmlNode*, *nodeTag*)

Find the required xml node @ In, xmlNode, xml.etree.ElementTree.Element, xml element node @ In, nodeTag, str, node tag that is used to find the node @ Out, subnode, xml.etree.ElementTree.Element, xml element node

src.poem._utils

utilities for use within POEM

Module Contents

Functions

get_raven_loc()	Return RAVEN location: read from POEM/.ravenconfig.xml
get_plugin_loc(plugin[, raven_path])	Get plugin location in installed RAVEN

Attributes

argvs

src.poem._utils.**get_raven_loc**()

Return RAVEN location: read from POEM/.ravenconfig.xml @ In, None @ Out, loc, string, absolute location of RAVEN

src.poem._utils.**get_plugin_loc**(*plugin*, *raven_path=None*)

Get plugin location in installed RAVEN @ In, plugin, string, the plugin name @ In, raven_path, string, optional, if given then start with this path @ Out, plugin_loc, string, location of plugin

src.poem._utils.**argvs**

src.poem.plotUtils

Module Contents

Functions

addPoint(ax, x, y, accepted)	
plotFunction(fig, title, method, constraint, xscale, ...)	Plots a 2D function as a colormap. Returns parameters suitable to plotting in a pcolormesh call.
animate(n, ax, x, y, a)	
animatePlot(x, y, a, fig, title, method, constraint, ...)	
optPath(x, y, a, fig, title, method, constraint, ...)	

Attributes

markerMap
markers

src.poem.plotUtils.**markerMap**

src.poem.plotUtils.**markers**

src.poem.plotUtils.**addPoint**(*ax*, *x*, *y*, *accepted*)

src.poem.plotUtils.**plotFunction**(*fig*, *title*, *method*, *constraint*, *xscale*, *yscale*, *cscale=None*, *log=False*, *samps=500*, *xp=None*, *yp=None*)

Plots a 2D function as a colormap. Returns parameters suitable to plotting in a pcolormesh call. @ In, title, string, title name for figure @ In, method, function, method to call with x,y to get z result @ In, constraint, function, boolean method that determines acceptability @ In, xscale, tuple(float), low/hi value for x @ In, yscale, tuple(float), low/hi value for y @ In, cscale, tuple(float), optional, low and high values for the color map @ In, log, bool, optional, if False will not lognormalize the color map @ In, samps, int @ In, extData, pandas.Dataframe @ Out, X, np.array(np.array(float)), mesh grid of X values @ Out, Y, np.array(np.array(float)), mesh grid of Y values @ Out, Z, np.array(np.array(float)), mesh grid of Z (response) values

src.poem.plotUtils.**animate**(*n*, *ax*, *x*, *y*, *a*)

src.poem.plotUtils.**animatePlot**(*x*, *y*, *a*, *fig*, *title*, *method*, *constraint*, *xscale*, *yscale*, *cscale=None*, *log=False*, *samps=500*, *xp=None*, *yp=None*)

src.poem.plotUtils.**optPath**(*x*, *y*, *a*, *fig*, *title*, *method*, *constraint*, *xscale*, *yscale*, *cscale=None*, *log=False*, *samps=500*, *xp=None*, *yp=None*)

src.poem.poemUtils

Created on April 1, 2024 @author: wangc

Module Contents

Functions

convertNodeTextToList(nodeText[, sep])	Convert space or comma separated string to list of string
convertNodeTextToFloatList(nodeText[, sep])	Convert space or comma separated string to list of float
convertStringToFloat(xmlNode)	Convert xml node text to float
convertStringToInt(xmlNode)	Convert xml node text to integer.
toString(s)	Method aimed to convert a string in type str
convertStringToBool(nodeText)	Convert string to bool

Attributes

logger

src.poem.poemUtils.**logger**

src.poem.poemUtils.**convertNodeTextToList**(*nodeText*, *sep=None*)

Convert space or comma separated string to list of string @ In, nodeText, str, string from xml node text @ Out, listData, list, list of strings

src.poem.poemUtils.**convertNodeTextToFloatList**(*nodeText*, *sep=None*)

Convert space or comma separated string to list of float @ In, nodeText, str, string from xml node text @ Out, listData, list, list of floats

src.poem.poemUtils.**convertStringToFloat**(*xmlNode*)

Convert xml node text to float @ In, xmlNode, xml.etree.ElementTree.Element, xml element @ Out, val, float, value of xml element text

src.poem.poemUtils.**convertStringToInt**(*xmlNode*)

Convert xml node text to integer. @ In, xmlNode, xml.etree.ElementTree.Element, xml element @ Out, val, integer, value of xml element text

src.poem.poemUtils.**toString**(*s*)

Method aimed to convert a string in type str @ In, s, string, string to be converted @ Out, s, string, the casted value

src.poem.poemUtils.**convertStringToBool**(*nodeText*)

Convert string to bool @ In, nodeText, str, string from xml node text @ Out, val, bool, True or False

CHAPTER

FOURTEEN

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

FIFTEEN

CONTRIBUTIONS

All contributions are welcome. You can help this project grow in multiple ways, from creating an issue, reporting an improvement or a bug, and creating a pull request to the development branch. The people involved at some point in the development of the package can be found in the *contributors page*.

PYTHON MODULE INDEX

S

INDEX

O

P

R

S

T

U

V

W