**User Guide of**

**Optimization Problem Solving   
with External Model in Raven Environment**

**Junyung Kim**

**Daniel Garrett**

**Paul W. Talbot**

Digital Reactor Technology & Development

Idaho National Laboratory

Table of Contents

[1 Optimization in PyNumero Framework 3](#_Toc111634786)

[2 ROM Construction in RAVEN 5](#_Toc111634787)

[3 Python Script File for Optimization in PyNumero 8](#_Toc111634788)

[Appendix I 13](#_Toc111634789)

[Appendix II 15](#_Toc111634790)

[1 Bouncing Ball 15](#_Toc111634791)

[1.1 Modelica Input file 15](#_Toc111634792)

[1.2 RAVEN Input file 16](#_Toc111634793)

[1.2.1 RunInfo 16](#_Toc111634794)

[1.2.2 Files 17](#_Toc111634795)

[1.2.3 Models 17](#_Toc111634796)

[1.2.4 Distributions 18](#_Toc111634797)

[1.2.5 Samplers 18](#_Toc111634798)

[1.2.6 DataObjects 19](#_Toc111634799)

[1.2.7 Steps 20](#_Toc111634800)

[1.3 Output 21](#_Toc111634801)

[2 Rankine Cycle Balance of Plant Model 24](#_Toc111634802)

[2.1 Modelica Input File 25](#_Toc111634803)

[2.2 RAVEN Input File 26](#_Toc111634804)

[2.2.1 RunInfo 26](#_Toc111634805)

[2.2.2 Files 26](#_Toc111634806)

[2.2.3 Models 27](#_Toc111634807)

[2.2.4 Distributions 28](#_Toc111634808)

[2.2.5 Samplers 28](#_Toc111634809)

[2.2.6 DataObjects 29](#_Toc111634810)

[2.2.7 Steps 30](#_Toc111634811)

[2.2.8 OutStreams 31](#_Toc111634812)

[2.3 Output 32](#_Toc111634813)

[References 34](#_Toc111634814)

# Optimization in PyNumero Framework

An integrated energy system (IES) is a comprehensive system integrating several energy sources (e.g., electricity and heat) and products (e.g., hydrogen). System integration, analysis, exploration, and optimization should be completed in the digital space. A Digital Twin (DT) approach is especially suitable for IES which requires integration of multiple subsystems and high reliability. One application of the DT concept is to conduct real-time optimization of the control and operations of a cyber-physical system (CPS) with data-driven simulations.

To develop the DT of a physical system, system-level simulation modelling in Modelica language and RAVEN (Risk Analysis Virtual Environment) (Rabiti, Alfonsi and Cogliati) have been mobilized to create a reduced order model (ROM) of the physical system. ROM works as DT of the physical system, and an optimization framework can receive the ROM and provide optimal inputs maximizing/minimizing objective functions.

The objective of this document is to provide users guidance showing how to create a DT in RAVEN and solve an optimization problem using external models in a Python-based optimization framework. For illustrating optimization workflows, system-level simulation models in Modelica and RAVEN have been connected to create a ROM of the physical system which is fed forward to PyNumero (Jose S. Rodriguez, Rodriguez and Parker), a module within Pyomo (Python Optimization Modeling Objects) (Hart, Watson and Woodruff, Pyomo: modeling and solving mathematical programs in Python). PyNumero is a package for developing parallel algorithms for nonlinear programming, and it provides users an interface to an external model (i.e., ROM) as a block in a Pyomo model. Pyomo is a Python-based, open-source optimization modeling language with a diverse set of optimization capabilities.Users who have an interest in Pyomo and PyNumero can refer to the websites below:

*Introduction of Pyomo*:

[https://Pyomo.readthedocs.io/en/stable/index.html](https://pyomo.readthedocs.io/en/stable/index.html).

*PyNumero Examples*:

[https://github.com/Pyomo/Pyomo/tree/main/Pyomo/contrib/pynumero/examples](https://github.com/Pyomo/pyomo/tree/main/pyomo/contrib/pynumero/examples)

In general, optimization packages in Python (e.g., Pyomo, PuLP (Mitchell, O'Sullivan and Dunning), cvxpy (Diamond and Boyd), and ortools) require the user to define inputs/outputs, an objective function, and constraints as algebraic expressions. The uniqueness of PyNumero is that it allows users to obtain these expressions from the external model directly. In addition, PyNumero can be used alongside Pyomo to provide a unified Python platform for both modeling and solving optimization problems (Jose S. Rodriguez, Rodriguez and Parker).

Once RAVEN creates a ROM, PyNumero can use the ROM and find optimal inputs maximizing or minimizing objective functions. The ExternalGreyBox model in PyNumero enhances the interface between the external model and the optimization framework: one does not need to provide inputs, constraints, and outputs explicitly as algebraic expressions to the optimization framework, but simply plug in the ROM to the ExternalGreyBox block in the framework. In this document, we will focus on explaining the workflow: how one can interface a system-level simulation model with RAVEN and PyNumero sequentially. Figure 1 shows a conceptual schematic of the optimization workflow.

|  |
| --- |
|  |
| **Figure 1. Optimization workflow.** |

# ROM Construction in RAVEN

In this section, as an example, the external Python model attenuate.py is trained and its ROM is saved to disk as a pickled file. RAVEN plays a key role transforming a physics-based system level model to a ROM. Some key code blocks of a RAVEN input file are presented to explain the general workflow of ROM construction in RAVEN: run simulations, train ROMs, serialize ROM. One can refer to attenuate.py in

…/raven/tests/framework/Models/External/AllMethods

and introductory model documentation can be found in

…/raven/docs/pdf/analytic\_tests.pdf

The full RAVEN input file used in this section can be found in Appendix I.

|  |
| --- |
| <Simulation>  ...  <Steps>  <MultiRun name="sample">  <Input class="DataObjects" type="PointSet">dummyIN</Input>  <Model class="Models" type="ExternalModel">attenuate</Model>  <Sampler class="Samplers" type="Grid">grid</Sampler>  <Output class="DataObjects" type="PointSet">samples</Output>  </MultiRun>  <RomTrainer name="train">  <Input class="DataObjects" type="PointSet">samples</Input>  <Output class="Models" type="ROM">out\_rom</Output>  </RomTrainer>  <IOStep name="serialize">  <Input class="Models" type="ROM">out\_rom</Input>  <Output class="Files" type="">rom\_out.py</Output>  </IOStep>  <IOStep name="pickle">  <Input class="Models" type="ROM">out\_rom</Input>  <Output class="Files" type="">rom\_pickle.pk</Output>  </IOStep>  </Steps>  ...  </Simulation> |

The <Steps> node combines other subnodes under <Steps> to detail a step in the RAVEN workflow:

1. <MultiRun> named “sample”, is used to run the multiple instances of the driven code and collect the output in the DataObjects. <Sampler> is inputted to communicate to the Step that the driven code needs to be perturbed through the Grid sampling strategy.
2. <RomTrainer> named train, is used to construct the linear regression ROM using DataObjects called samples. Notice that samples is the output from <MultiRun> and it becomes the input of <RomTrainer>.
3. One needs two inputs to solve optimization problem using PyNumero: 1) a Python script file for optimization and 2) a pickled ROM file. Two <IOStep> blocks work separately to create those inputs. Notice that both <IOStep name="serialize"> and <IOStep name="pickle"> receive a ROM model called out\_rom as an input, which is the output from <Models>.

The Python script file (i.e., rom\_out.py) contains the infrastructure inheriting the ExternalGreyBoxModel of PyNumero and other code blocks to print a file with Pyomo syntax. More details of the PyNumero codes will be discussed in Section 3. The ‘.pk’ output file is the pickled version of the trained ROM, and it will become an input of the Python script file. The subtype of the ROM model can be of any kind and is the user’s decision to choose the training method. It is worth noting that users should carefully select ROM type because it affects the optimization time in PyNumero framework.

|  |
| --- |
| <Simulation>  ...  <Models>  <ExternalModel ModuleToLoad="../tests/framework/Models/External/attenuate.py" name="attenuate" subType="">  <inputs>y1,y2</inputs>  <outputs>ans, fromInit, fromReadMoreXML, fromCNISelf, fromCNIDict</outputs>  <moreXMLInfo>  <valueForXML>3.14159</valueForXML>  </moreXMLInfo>  </ExternalModel>  <ROM name="out\_rom" subType="LinearRegression">  <Features>y1, y2</Features>  <Target>ans</Target>  </ROM>  </Models>  ...  </Simulation> |

The <Models> block represents the projection from the input to the output space. The <ExternalModel> represents a physical or mathematical model that is directly implemented by the user in a Python module. Notice that subnode called <ROM> is placed inside of <Models> and one specifies a ROM type as a subtype argument (LinearRegressionis specified here). RAVEN supports a variety of ROM types for users. Please refer to **Section 15.3 on ROMs** in RAVEN input manual (Revision 7).

# Python Script File for Optimization in PyNumero

As described in Section 2., two files are required to solve the optimization problem in PyNumero framework: a Python script and a pickled ROM file. The Python script is for interfacing RAVEN ROM and Pynumero Grey Box Model: it passes the pickled ROM file into the External Grey Box Block in PyNumero. In this section, we will explain how the Python script file generated by <IOStep name="serialize"> in the RAVEN input file (i.e., rom\_out.py) interfaces a ROM file and the External Grey Box Model in PyNumero. However, it is worth noting that the Python script file here simply works as a template for users who are not familiar with solving optimization problems in the PyNumero framework. Users could change the objective function, constraints, or other options.

|  |
| --- |
|  |

First, to run the Python script, one needs to import necessary packages. Notice that Pyomo objects exist within the Pyomo.environ. In addition, it is required to import ExternalGreyBoxModel and ExternalGreyBoxBlock from the interface of Pynumero package. Notice that ExternalGreyBoxBlock automatically creates input and output variables corresponding to ExternalGreyBoxModel. It allows the user to pass these in and, if provided, use the provided variable data objects instead of creating them. The pickle package is imported to load the pickled ROM file.

There are three major code blocks in rom\_out.py: RAVEN-ROM-Pyomo class derived from the External Grey Box Model in PyNumero, Pyomo model, and Pyomo Grey Box for nonlinear programming.

|  |
| --- |
|  |

Class ravenROM is what allows an external model to be solved in PyNumero. It is for loading the pickled ROM file and collecting information from the external model loaded including a list of names for inputs and outputs. This class contains a method computing the derivatives and the Jacobian of the output with respect to the inputs. These derivatives and Jacobian calculated will be used later for an optimization solver.

|  |
| --- |
|  |

A function PyomoModel sets up an External Grey Box Block for producing a Pyomo modeling component. It creates Pyomo variables to represent the inputs and the outputs from the external model. For setting up the objective function, one can use sense option as an argument of pyo.Objective to specify either minimization or maximization. Still, users have flexibility to define the function based on the problem at hand since an optimization problem could be set up many different ways.

|  |
| --- |
|  |

At last, the code above creates a Pyomo grey box nonlinear programming problem and provides Pyomo the model that contains the external grey box block(s). This code block is how Pyomo actually solves an example optimization problem. Notice that, in this code block, the external model (i.e., ext\_model) is loaded into pyomoModel and users can freely add other inputs, constraints and an objective in concreteModel. Details of adding inputs and constraints in a concrete model of Pyomo can be found in (Hart, Laird and Watson). Currently, cyipopt, Python wrapper for the IPOPT (Biegler and Zavala) optimization package, written in Cython, is currently the only solver supported.

|  |
| --- |
| **Text  Description automatically generated** |

Once the code runs successfully, the user can see GreyModelOutput.txt file in the same folder of rom\_out.py file. It contains summary of the optimization problem including which solver has been used, whether the problem terminated successfully, and what the optimized input/output values are. In the Optimization Result in GreyModelOutput.txt, one can find that output value, ans, has its minimized value (0.358796455415) when both y1 and y2 have 0.938770350605.

# Appendix I

**RAVEN input file for optimization problem solving of attenuate.py**

|  |
| --- |
| <?xml version="1.0" ?>  <Simulation>    <TestInfo>      <name>framework.Models.External.serialize\_pyomo</name>      <author>cogljj</author>      <created>2021-08-18</created>      <classesTested>Models.ROM</classesTested>      <description>        This test serializes a ROM to use with PYOMO.        It can be used with:        cd tests/framework/Models/External        python3 SerializePyomo/rom\_out.py -r SerializePyomo/rom\_pickle.pk -f ../../../..      </description>    </TestInfo>    <RunInfo>      <WorkingDir>SerializePyomo</WorkingDir>      <Sequence>sample, train, serialize, pickle</Sequence>    </RunInfo>    <Steps>      <MultiRun name="sample">        <Input class="DataObjects" type="PointSet">dummyIN</Input>        <Model class="Models" type="ExternalModel">attenuate</Model>        <Sampler class="Samplers" type="Grid">grid</Sampler>        <Output class="DataObjects" type="PointSet">samples</Output>      </MultiRun>      <RomTrainer name="train">        <Input class="DataObjects" type="PointSet">samples</Input>        <Output class="Models" type="ROM">out\_rom</Output>      </RomTrainer>      <IOStep name="serialize">        <Input class="Models" type="ROM">out\_rom</Input>        <Output class="Files" type="">rom\_out.py</Output>      </IOStep>      <IOStep name="pickle">        <Input class="Models" type="ROM">out\_rom</Input>        <Output class="Files" type="">rom\_pickle.pk</Output>      </IOStep>    </Steps>    <Files>      <Input name="rom\_out.py" type="Pyomo">rom\_out.py</Input>      <Input name="rom\_pickle.pk" type="">rom\_pickle.pk</Input>    </Files>    <Models>      <ExternalModel ModuleToLoad="../AllMethods/attenuate" name="attenuate" subType="">        <inputs>y1,y2</inputs>        <outputs>ans,fromInit,fromReadMoreXML,fromCNISelf,fromCNIDict</outputs>        <moreXMLInfo>          <valueForXML>3.14159</valueForXML>        </moreXMLInfo>      </ExternalModel>      <ROM name="out\_rom" subType="LinearRegression">        <Features>y1,y2</Features>        <Target>ans</Target>      </ROM>    </Models>    <Distributions>      <Uniform name="dist">        <lowerBound>0</lowerBound>        <upperBound>1</upperBound>      </Uniform>    </Distributions>    <Samplers>      <Grid name="grid">        <variable name="y1">          <distribution>dist</distribution>          <grid type='CDF' construction='equal' steps='1'>0 1</grid>        </variable>        <variable name="y2">          <distribution>dist</distribution>          <grid type='CDF' construction='equal' steps='1'>0 1</grid>        </variable>      </Grid>    </Samplers>    <DataObjects>      <PointSet name="dummyIN">        <Input>y1,y2</Input>      </PointSet>      <PointSet name="samples">        <Input>y1, y2</Input>        <Output>ans</Output>      </PointSet>    </DataObjects>  </Simulation> |

# Appendix II

**Use Cases with DYMOLA simulation models**

# Bouncing Ball

In this example, a standard Modelica example called BouncingBall, which simulates the trajectory of an object falling in one dimension from a height with given velocity is shown as an example. The objective is set to maximize the height of the ball with starting height and velocity during the trajectory.

## Modelica Input file

One needs to install the NHES package in DYMOLA (Dassault Systèmes AB). The NHES package can be downloaded from the HYBRID repository (Frick, Alfonsi and Rabiti) in GitHub (i.e., <https://github.com/idaholab/HYBRID>). After successfully installing the package, one can find the BouncingBall model in NHES\RAVEN\_Interface\Examples\Bouncing\_Ball. Before running the model, it is required to change Modelica codes so that the maximum value of height during the trajectory is calculated.

|  |
| --- |
| model Bouncing\_Ball    "Bouncing Ball Example used for RAVEN DYMOLA Code Interface"    parameter Real e=0.7 "coefficient of restitution";    parameter Real g=9.81 "gravity acceleration";    parameter Real hstart = 5 "height of ball at time zero";    parameter Real vstart = 0 "velocity of ball at time zero";    Real h\_max(start = hstart, fixed = true);    Real h(start=hstart,fixed=true) "height of ball";    Real v(start=vstart,fixed=true) "velocity of ball";    Boolean flying(start=true) "true, if ball is flying";    Boolean impact;    Real v\_new;    Integer foo;    parameter Real T = 1e-4;  equation    impact = h <= 0.0;    foo = if impact then 1 else 2;    der(v) = if flying then -g else 0;    der(h) = v;    h\_max = max(h, h\_max);    when {h <= 0.0 and v <= 0.0,impact} then      v\_new = if edge(impact) then -e\*pre(v) else 0;      flying = v\_new > 0;      reinit(v, v\_new);    end when;  //   annotation (uses(Modelica(version="3.2.1")),  //     experiment(StopTime=10, Interval=0.1),  //     \_\_DYMOLA\_experimentSetupOutput);    annotation (Icon(coordinateSystem(preserveAspectRatio=false), graphics={          Ellipse(lineColor = {75,138,73},                  fillColor={255,255,255},                  fillPattern = FillPattern.Solid,                  extent={{-100,-100},{100,100}}),          Polygon(lineColor = {0,0,255},                  fillColor = {75,138,73},                  pattern = LinePattern.None,                  fillPattern = FillPattern.Solid,                  points={{-36,56},{64,-4},{-36,-64},{-36,56}})}), Diagram(          coordinateSystem(preserveAspectRatio=false)),      experiment(        StopTime=10,        \_\_DYMOLA\_NumberOfIntervals=10000,        \_\_DYMOLA\_Algorithm="Dassl"),      \_\_DYMOLA\_experimentSetupOutput);  end Bouncing\_Ball; |

## RAVEN Input file

### RunInfo

|  |
| --- |
| <Simulation>  ...    <RunInfo>      <JobName>BouncingBall\_CreatePickle\_</JobName>      <Sequence>        sample,trainROM,sampleROM,dumpROM      </Sequence>      <WorkingDir>BouncingBall\_output</WorkingDir>      <batchSize>8</batchSize>    </RunInfo>  ...  </Simulation> |

### Files

An executable (dymosim.exe) and a simulation initialization file (dsin.txt) can be generated after simulating the Modelica model (BouncingBall.mo) using the DYMOLA Graphical User Interface (GUI) or DYMOLA Application Programming Interface (API)-routines. In <Files> block, the location of the initialization file and the name of pickled file which will be serialized are specified to RAVEN.

|  |
| --- |
| <Simulation>  ...    <Files>      <Input name="BouncingBallInput" type="DYMOLAInitialisation"> \..\..\dsin.txt</Input>      <Input name="rom\_pickle" type="">BouncingBall.pk</Input>    </Files>  ...  </Simulation> |

### Models

The model executable is specified to RAVEN in <Models> block, and the N-dimensional inverse distance weighting algorithm <NDinvDistWeight> (D'amario) used for constructing a ROM.

|  |
| --- |
| <Simulation>  ...    <Models>      <Code name="BouncingBall" subType = "DYMOLA">        <executable>\..\..\dymosim.exe</executable>      </Code>      <ROM name="ROM" subType="NDinvDistWeight">        <pivotParameter>Time</pivotParameter>        <Features>hstart, vstart</Features>        <Target>h\_max</Target>        <p>3</p>      </ROM>    </Models>  ...  </Simulation> |

### Distributions

The <Distributions> block defines probability distribution functions for inputs. In this example, **Uniform(20,40)** and **Uniform(0,30)** have been placed for starting height and velocity respectively.

|  |
| --- |
| <Simulation>  ...    <Distributions>      <Uniform name="hstart\_dist">        <lowerBound>20</lowerBound>        <upperBound>40</upperBound>      </Uniform>      <Uniform name="vstart\_dist">        <lowerBound>0</lowerBound>        <upperBound>30</upperBound>      </Uniform>    </Distributions>  ...  </Simulation> |

### Samplers

In this example, 500 of hstart and vstart are sampled following hstart\_dist and vstart\_dist respectively.

|  |
| --- |
| <Simulation>  ...    <Samplers>      <MonteCarlo name="my\_mc">        <samplerInit>          <limit>500</limit>          <initialSeed>42</initialSeed>        </samplerInit>        <variable name="hstart">          <distribution>hstart\_dist</distribution>        </variable>        <variable name="vstart">          <distribution>vstart\_dist</distribution>        </variable>      </MonteCarlo>    </Samplers>  ...  </Simulation> |

### DataObjects

|  |
| --- |
| <Simulation>  ...    <DataObjects>      <PointSet name="inputPlaceHolder">          <Input>hstart, vstart</Input>          <Output>OutputPlaceHolder</Output>      </PointSet>      <PointSet name="samples">        <Input>hstart, vstart</Input>        <Output>h\_max, Time</Output>      </PointSet>      <PointSet name="samplesROM">        <Input>hstart, vstart</Input>        <Output>h\_max</Output>      </PointSet>    </DataObjects>  ...  </Simulation> |

### Steps

|  |
| --- |
| <Simulation>  ...    <Steps>      <MultiRun name="sample">        <Input class="Files" type="DYMOLAInitialisation">BouncingBallInput</Input>        <Model class="Models" type="Code">BouncingBall</Model>        <Sampler class="Samplers" type="MonteCarlo">my\_mc</Sampler>        <Output class="DataObjects" type="PointSet">samples</Output>      </MultiRun>      <MultiRun name="sampleROM">        <Input class="DataObjects" type="PointSet">inputPlaceHolder</Input>        <Model class="Models" type="ROM">ROM</Model>        <Sampler class="Samplers" type="MonteCarlo">my\_mc</Sampler>        <Output class="DataObjects" type="PointSet">samplesROM</Output>      </MultiRun>      <RomTrainer name="trainROM">        <Input class="DataObjects" type="PointSet">samples</Input>        <Output class="Models" type="ROM">ROM</Output>      </RomTrainer>      <IOStep name="dumpROM">        <Input class="Models" type="ROM">ROM</Input>        <Output class="Files" type="">rom\_pickle</Output>      </IOStep>    </Steps>  ...  </Simulation> |

## Output

Optimization using a RAVEN-generated ROM can be performed in other environments like Python scripts or Jupyter notebooks easily. The Python code below has been shown for users who use Jupyter notebook. Notice that the code is almost identical to rom\_out.py except a little change.

|  |
| --- |
| Text  Description automatically generated |

|  |
| --- |
|  |

The code cell above is to load a pickled ROM file and set the path of the shell script file, raven\_framework.

|  |
| --- |
| Text  Description automatically generated |

Once, again, the function, pyomoModel, can be set up by users to perform whatever optimization they want. Notice that, in this case, sense argument in pyo.Objective is set to maximize the objective function.

|  |
| --- |
| A picture containing chart  Description automatically generated |

This code cell is for setting the optimization solver and executing PyNumero to solve the optimization problem we assigned in the function, pyomoModel.

|  |
| --- |
| Text  Description automatically generated |

After running the solver and calling concreteModel, one can find that the concreteModel has been updated with the optimal value. Notice that the optimal input values generated from the ROM is close to analytical optimal values. One can find the reason why the optimal values found (i.e., 39.859201 for hstart and 27.573399 for vstart) were not same as analytically solved optimal values (i.e., 39.859295 for hstart and 29.932214 for vstart) from the ROM type used. N-dimensional inverse distance weighting algorithm is a type of deterministic method for multivariate interpolation with a known scattered set of points. The assigned values to unknown points are calculated via a weighted average of the values available at the known points. Since analytical optimal values are in fact placed at the edge of inputs boundary, ROM could not afford to predict precise values.

# Rankine Cycle Balance of Plant Model

In this example, a Rankine cycle balance of plant (BOP) model in the NHES package[[1]](#footnote-1) has been used for the demonstrative purpose showing the optimization problem solving. The Optimization problem is defined as below:

1. Scenario:
   * Time domain: [0,130] sec.
   * External electricity load demand changes over time starting at 30 sec[[2]](#footnote-2). Until 130 sec., the load demand signal is replicated in sine function.
   * Operators maneuver Turbine bypass valve (TBV) so that the amount of steam goes to the turbine can be controlled.
2. Goal:
   * Match electrical power generated as close as possible to the load demanded.
3. Feature Variables:
   * Feature variables: TBV opening area at different time steps (30, 40, …, 120 sec)
4. Objective Function:

where

= load demanded at time t (unit: W)

= steam BV opening area at time t;

= electricity generated given steam TBV opening area at time t (unit: W)

One may need to notice that 1e8 has been used as a scaling factor in the objective function so that the magnitude of the objective function value does not increase rapidly. Scaling is of importance for the numerical solution especially if the optimization method one may use is not scaling invariant.

## Modelica Input File

Modelica.Blocks.Sources.Sine and Modelica.Blocks.Continuous.Integrator components are used along with a newly defined math block calculating the objective function. One can find in Figure 1-a that the math block receives two inputs from the sine and powerSensor components and send an output to the Integrator component.

For complying with the variable naming convention in the RAVEN framework (no commas, no space in a variable name), the use of a combiTable component in DYMOLA is limited. Rather than using any types of the table component in DYMOLA, vectorization approach for time-dependent RAVEN input has been introduced. Figure 1-b shows the Modelica code snippet developed for vectorization of time and input variables.

|  |
| --- |
|  |
| **a. Rankine cycle model in the Graphs mode in DYMOLA.** |
| **Text  Description automatically generated** |
| **b. Modelica Code snippet of the modified Rankine cycle balance of plant DYMOLA model.** |
| **Figure 1. Modified Rankine cycle balance of plant DYMOLA model.** |
|  |

## RAVEN Input File

### RunInfo

|  |
| --- |
| <Simulation>  ...    <RunInfo>      <JobName>BOP\_CreatePickle\_</JobName>      <Sequence>       sample, trainROM, sampleROM, dumpROM, writeHistories      </Sequence>      <WorkingDir> BOP\_NDinvDistWeight</WorkingDir>      <batchSize>8</batchSize>    </RunInfo>  ...  </Simulation> |

### Files

|  |
| --- |
| <Simulation>  ...    <Files>      <Input name="BOPInput" type="DYMOLAInitialisation"> /../../dsin.txt</Input>      <Input name="rom\_pickle" type="">BOP.pk</Input>    </Files>  ...  </Simulation> |

### Models

|  |
| --- |
| <Simulation>  ...    <Models>      <Code name="BOP" subType = "DYMOLA">        <executable>/../../dymosim.exe</executable>  <alias type="input" variable="valve\_area\_t30\_40">BOP.valve\_area\_vec[2]</alias>  <alias type="input" variable="valve\_area\_t40\_50">BOP.valve\_area\_vec[3]</alias>  <alias type="input" variable="valve\_area\_t50\_60">BOP.valve\_area\_vec[4]</alias>  <alias type="input" variable="valve\_area\_t60\_70">BOP.valve\_area\_vec[5]</alias>  <alias type="input" variable="valve\_area\_t70\_80">BOP.valve\_area\_vec[6]</alias>  <alias type="input" variable="valve\_area\_t80\_90">BOP.valve\_area\_vec[7]</alias>  <alias type="input" variable="valve\_area\_t90\_100">BOP.valve\_area\_vec[8]</alias>  <alias type="input" variable="valve\_area\_t100\_110">BOP.valve\_area\_vec[9]</alias>  <alias type="input" variable="valve\_area\_t110\_120">BOP.valve\_area\_vec[10]</alias>  <alias type="input" variable="valve\_area\_t120\_130">BOP.valve\_area\_vec[11]</alias>      </Code>      <ROM name="ROM" subType="NDinvDistWeight">        <pivotParameter>Time</pivotParameter>  <Features> valve\_area\_t30\_40, valve\_area\_t40\_50, valve\_area\_t50\_60, valve\_area\_t60\_70, valve\_area\_t70\_80,valve\_area\_t80\_90,valve\_area\_t90\_100,valve\_area\_t100\_110,valve\_area\_t110\_120, valve\_area\_t120\_130 </Features>        <Target>BOP.obj.y</Target>        <p>1</p>      </ROM>    </Models>  ...  </Simulation> |

### Distributions

|  |
| --- |
| <Simulation>  ...    <Distributions>      <Uniform name=" va\_t30\_40\_dist">        <lowerBound>0.1</lowerBound>        <upperBound>0.2</upperBound>      </Uniform>      <Uniform name=" va\_t40\_50\_dist">        <lowerBound>0.15</lowerBound>        <upperBound>0.25</upperBound>      </Uniform>  ...    </Distributions>  ...  </Simulation> |

### Samplers

|  |
| --- |
| <Simulation>  ...    <Samplers>      <MonteCarlo name="my\_mc">        <samplerInit>          <limit>2000</limit>          <initialSeed>42</initialSeed>        </samplerInit>        <variable name="valve\_area\_t30\_40">          <distribution>va\_t30\_40\_dist</distribution>        </variable>        <variable name="valve\_area\_t40\_50">          <distribution>va\_t40\_50\_dist</distribution>        </variable>  ...      </MonteCarlo>    </Samplers>  ...  </Simulation> |

### DataObjects

|  |
| --- |
| <Simulation>  ...    <DataObjects>      <PointSet name="inputPlaceHolder">  <Input>valve\_area\_t30\_40, valve\_area\_t40\_50, valve\_area\_t50\_60, valve\_area\_t60\_70, valve\_area\_t70\_80, valve\_area\_t80\_90, valve\_area\_t90\_100, valve\_area\_t100\_110, valve\_area\_t110\_120, valve\_area\_t120\_130</Input>  <Output>OutputPlaceHolder</Output>      </PointSet>      <PointSet name="samples">  <Input>valve\_area\_t30\_40, valve\_area\_t40\_50, valve\_area\_t50\_60, valve\_area\_t60\_70, valve\_area\_t70\_80, valve\_area\_t80\_90, valve\_area\_t90\_100, valve\_area\_t100\_110, valve\_area\_t110\_120, valve\_area\_t120\_130</Input>  <Output>BOP.obj.y</Output>      </PointSet>      <HistorySet name="histories">  <Input>valve\_area\_t30\_40, valve\_area\_t40\_50, valve\_area\_t50\_60, valve\_area\_t60\_70, valve\_area\_t70\_80, valve\_area\_t80\_90, valve\_area\_t90\_100, valve\_area\_t100\_110, valve\_area\_t110\_120, valve\_area\_t120\_130</Input>  <Output>BOP.obj.y, Time</Output>  <options>  <pivotParameter>Time</pivotParameter>  </options>      </HistorySet>    </DataObjects>  ...  </Simulation> |

### Steps

|  |
| --- |
| <Simulation>  ...    <Steps>      <MultiRun name="sample">        <Input class="Files" type="DymolaInitialisation">BOPInput</Input>        <Model class="Models" type="Code">BOP</Model>        <Sampler class="Samplers" type="MonteCarlo">my\_mc</Sampler>        <Output class="DataObjects" type="PointSet">samples</Output>        <Output class="DataObjects" type="HistorySet">histories</Output>      </MultiRun>      <MultiRun name="sampleROM">        <Input class="DataObjects" type="PointSet">inputPlaceHolder</Input>        <Model class="Models" type="ROM">ROM</Model>        <Sampler class="Samplers" type="MonteCarlo">my\_mc</Sampler>        <Output class="DataObjects" type="PointSet">samplesROM</Output>      </MultiRun>      <RomTrainer name="trainROM">        <Input class="DataObjects" type="PointSet">samples</Input>        <Output class="Models" type="ROM">ROM</Output>      </RomTrainer>      <IOStep name="dumpROM">        <Input class="Models" type="ROM">ROM</Input>        <Output class="Files" type="">rom\_pickle</Output>      </IOStep>      <IOStep name="writeHistories" pauseAtEnd="True">        <Input class="DataObjects" type="HistorySet">histories</Input>        <Input class="DataObjects" type="PointSet">samples</Input>        <Output class="OutStreams" type="Print">histories</Output>        <Output class="OutStreams" type="Print">samples</Output>      </IOStep>    </Steps>  ...  </Simulation> |

### OutStreams

|  |
| --- |
| <Simulation>  ...  <OutStreams>  <Print name="samples">  <type>csv</type>  <source>samples</source>  </Print>  <Print name="histories">  <type>csv</type>  <source>histories</source>  </Print>  </OutStreams>  ...  </Simulation> |

## Output

After running the solver, one can find that the optimized input values and the corresponding output value are saved in concreteModel. Once plugging in those input values back to the DYMOLA simulation model, one can visually confirm that the objective function is minimized with given optimal inputs.

|  |
| --- |
|  |
| **Figure 2. Optimization result in the PyNumero Framework.** |

|  |
| --- |
|  |
| **a. Turbine Bypass Valve (TBV) opening area.** |
|  |
| **b. Power generation vs. load demanded.** |
|  |
| **c. Objective function value.** |
| **Figure 3. Visualization of the optimized result.** |

# References

Biegler, Lorenz T and Victor M Zavala. "Large-scale nonlinear programming using IPOPT: An integrating framework for enterprise-wide dynamic optimization." *Computers & Chemical Engineering* (2009): 575-582. 33.3.

D'amario, Alessandro. *A reduced-order Inverse Distance Weighting technique for the efficient mesh-motion of deformable interfaces and moving shapes in computational problems*. Dissertation. Milano: Politecnico di Milano, 2016.

Dassault Systèmes AB. *Dymola Dynamic Modeling Laboratory Full User Manual*. User Manual. Lund, Sweden: Dassault Systèmes AB, 2021.

Diamond, Steven and Stephen Boyd. "CVXPY: A Python-Embedded Modeling Language for Convex Optimization." *Journal of Machine Learning Research* (2016): 1-5. 17.

Frick, Konor L, et al. *Hybrid User Manual (INL/MIS-20-60624-Rev001)*. Program Document. Idaho Falls, ID, USA: Idaho National Laboratory, 2022.

Hart, William E, et al. *Pyomo - Optimization Modeling in Python*. Springer, 2021. Vol. 67.

Hart, William E, Jean-Paul Watson and L David Woodruff. "Pyomo: modeling and solving mathematical programs in Python." *Mathematical Programming Computation* 3.3 (2011): 219-260.

Jose S. Rodriguez, Robert Parker, Carl D. Laird, Bethany Nicholson, John D. Siirola, Michael Bynum, et al. "Scalable Parallel Nonlinear Optimization with PyNumero and Parapint." *Optimization Online* (2021).

Mitchell, Stuart, Michael O'Sullivan and Iain Dunning. *PuLP: A Linear Programming Toolkit for Python*. Auckland, New Zealand: The University of Auckland, 2011.

Rabiti, Cristian, et al. *RAVEN User Manual (INL/EXT-15-34123 Rev. 7)*. Manual. Idaho Falls, ID, USA: Idaho National Laboratory, 2022.

1. One can find a model in the following location: NHES.Systems.BalanceOfPlant.Turbine.Example.SteamTurbine\_L1\_boundaries\_Test\_b [↑](#footnote-ref-1)
2. 30 seconds was postulated to make simulation achieve a steady state. [↑](#footnote-ref-2)