Delivery date:

31-08-2015

Authors:

Sigve Karolius

MoDeNa

Deliverable D5.7

Production Release - Parameter Estimation Framework

Principal investigators:

Sigve Karolius
Heinz A Preisig
Henrik Rusche
Dominik Christ

Collaborators:

Mandar Thombre

Project coordinator:

Heinz A Preisig

heinz.preisig@chemeng.ntnu.no

Abstract:

The MoDeNa parameter estimation framework is implemented for the purpose of ensuring that surrogate models, which is fitted to the input-output behaviour of the complex model, are automatically updated. The framework is executed on demand, i.e. the model requesting the surrogate is controlling the domain in which the surrogate is fitted and the strategy on how the domain is being extended and what error chriterion is used, working recursively accross scales.

Contents

1 Introduction

The parameter estimation part of the MoDeNa software framework is based on the methodology commonly referred to as: "Model Based Design of Experiment" (MBDoE), outlined in (Franceschini and Macchietto, 2008) and illustrated in Figure 1. It has been implemented to make it highly modular, and require minimal effort from the user. Moreover, it utilises a interface to the programming language R, which has a rich library of space-filling algorithms and regression techniques. In the need for a parameter

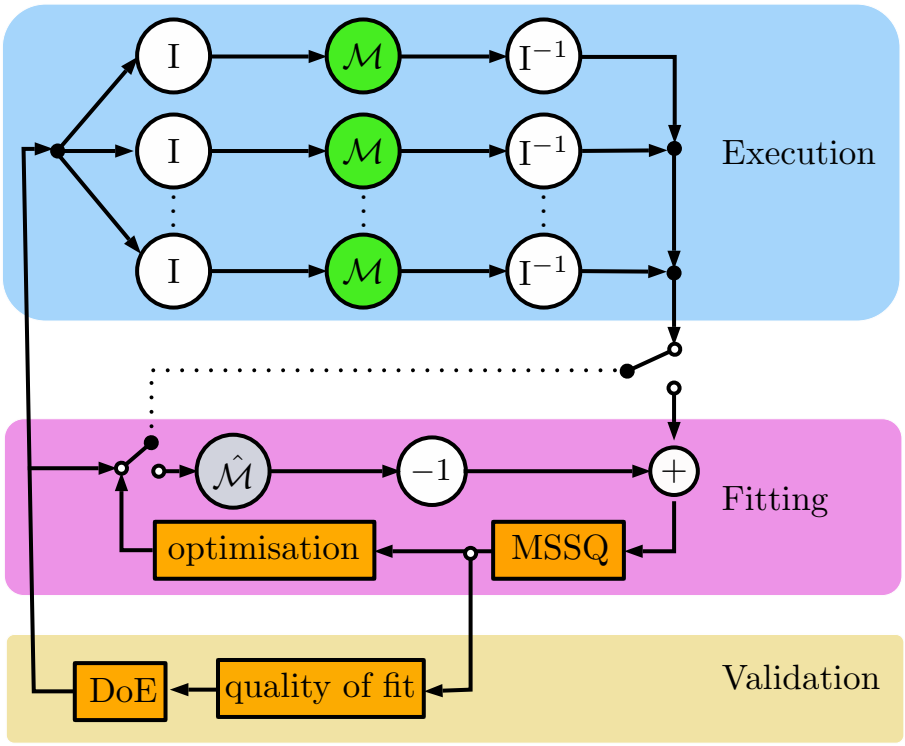

Figure 1: Figure illustrating the procedure of Model Based Design of Experiments for parameter fitting and validation of a surrogate model $\hat{\mathcal{M}}$ to the input-output behaviour of a complex model $\mathcal{M}$. The execution phase performs individual simulations of the exact model for a set of samples in order to provide input-output data for the fitting procedure. The parameters of the surrogate model are determined by regression and optimisation, e.g. using minimal sum of squares as the objective function. Finally the parameters are validated using some measure, e.g. maximum error, to quantify the gap between the surrogate and the exact model. In the event that the model is not deemed valid the framework tries to add additional samples to the range through a space filling design of experiments algorithm.

estimation framework is due to the use of surrogates in-place of the complex models at each scale. The surrogate model approximates the input-output behaviour of the complex model for the purpose of significantly reducing the computational cost of the complete simulation, which requires many evaluations of the models on every scale. This makes it possible to incorporate nano-scale models into macro-scale applications by interlinking them, e.g. all the way from molecular dynamics to computational fluid dynamics and structural mechanics.

In this aspect, the task of the parameter estimation part of the software framework is to ensure that each surrogate model can always be executed, i.e. that it is validated to possess predictive capabilities

within a margin of error. Moreover, the framework must handle different states of the surrogate model: initialisation, validity range, parameter fitting, parameter improvement, and validation.

2 Modelling Concept

The theoretical perspective of the parameter estimation framework was outlined in (Preisig et al., 2014), it mathematically describes the connection between models in a multi-scale application as follows:

$$\mathbf{y}^{\mathrm{i}} := \mathcal{M}\left(\left[\mathcal{M}_j^{\mathrm{i}-1}\right], \mathbf{u}^{\mathrm{i}}\right) \qquad ; \quad \mathbf{u}^{\mathrm{i}} \in \mathbf{U}_{max}^{\mathrm{i}}$$

The equation states that the output from a model, $\mathbf{y}$, is functionally dependent on models from the lower scales, as well as the input $\mathbf{u}^{\mathrm{i}}$. Moreover, the input is bound by the absolute domain for the complex model $\mathbf{U}_{max}^{\mathrm{i}}$ as illustrated in Figure 2. The mathematical description of a surrogate model is

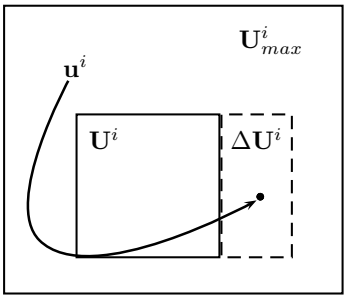

Figure 2: The design space is made up of three parts: the absolute domain $\mathbf{U}_{max}^{\mathrm{i}}$, i.e. the maximum bounds of the model; the local domain $\mathbf{U}^{\mathrm{i}}$, i.e. the domain where the surrogate has been validated; the extended domain $\Delta\mathbf{U}^{\mathrm{i}}$, a domain added to the design space.

analogous, apart for the parameters ($\boldsymbol{\theta}$):

$$\hat{\mathbf{y}}^{\mathrm{i}} = \hat{\mathcal{M}}\left(\left[\hat{\mathcal{M}}_j^{\mathrm{i}-1}\right], \mathbf{u}^{\mathrm{i}}; \boldsymbol{\theta}\right) \qquad ; \quad \mathbf{u}^{\mathrm{i}} \in \mathbf{U}^{\mathrm{i}} \subset \mathbf{U}_{max}^{\mathrm{i}} \;\; , \;\; \boldsymbol{\theta} \in \mathbb{R}$$

The connections between scales combined with the MBDoE algorithm leads to a computational workflow that is somewhat difficult to understand, but that Figure 4 attempts to illustrate. The figure shows a application which employs several surrogate models and evaluates them iteratively, checking after each execution that the MoDeNa framework is not reporting any exceptions. When a exception is detected the model-based design of experiments from Figure 1 is executed in order to re-fit the parameters. However, this implies executing exact simulations of the lower scale, e.g. the mesoscale in Figure 3, which in turns uses surrogate models from other scales. This means that the parameter estimation framework allows the dependencies to cascade all the way from the macroscopic scale to the electronic scale.

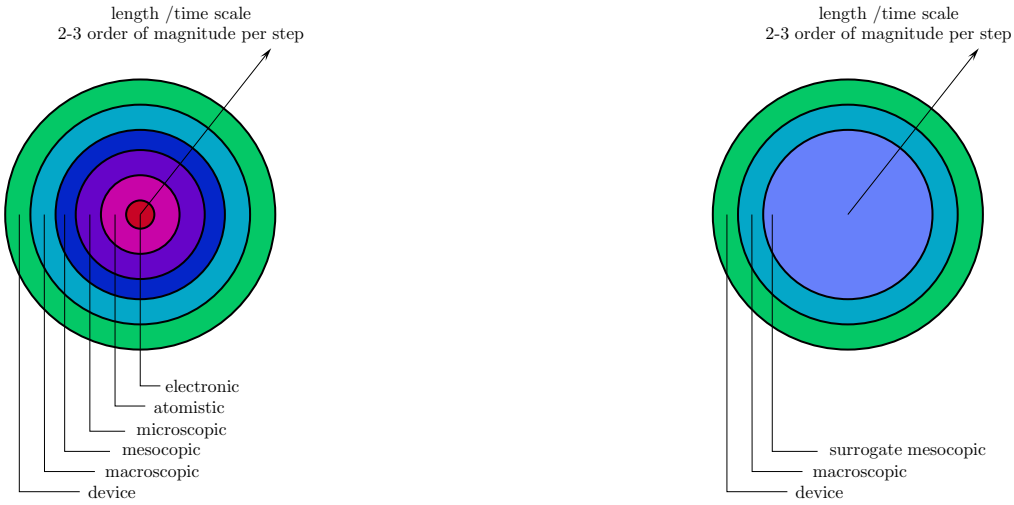

Figure 3: The left figure shows a multi-scale device where models from every scale is interlinked, whereas the right illustrates how the complex model of the mesoscopic scale is replaced by a surrogate.

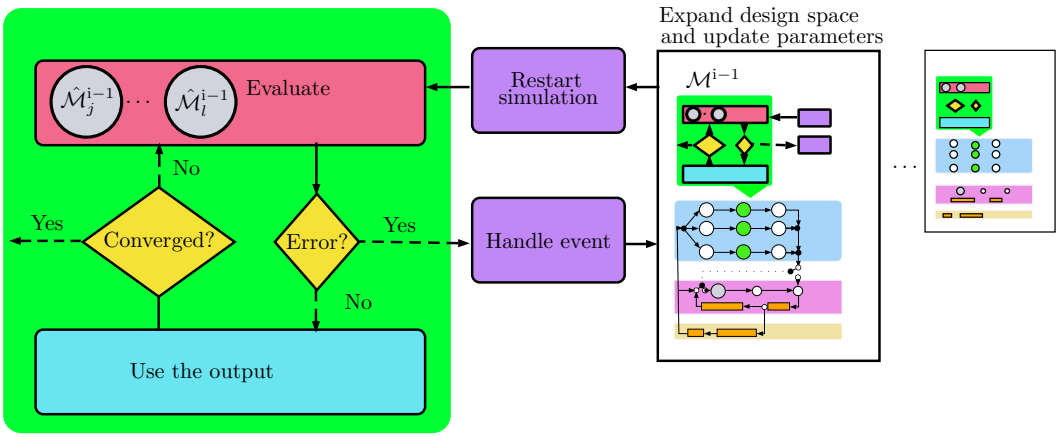

Figure 4: Illustration of a model (green) which employs surrogate models ($\hat{\mathcal{M}}$) for the lower scale, checking for errors after every evaluation. The parameter estimation framework handles errors using a model model based design of experiments procedure, which means evaluating surrogate models for yet another scale, and so on.

2.1 Illustration of the "life" of a surrogate model in a computational workflow

2.1.1 Initialisation

The initialisation of the surrogate model puts it into the database and thus makes it available for the other models in the application to use. When the execution of all the samples of the design space is

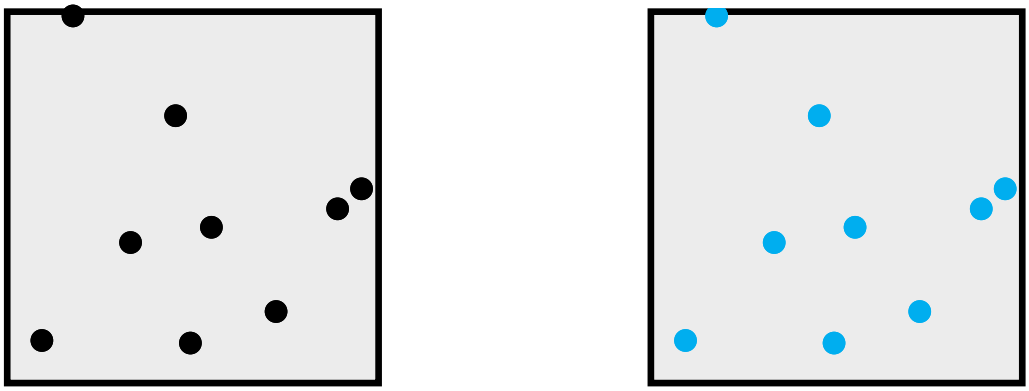

Figure 5: The figure illustrates the initialisation of a surrogate model by filling the design space of the model with **points** that should be explored further. The framework simulates the points and performs parameter fitting of the surrogate function using a subset of the simulations. The rest of the points are used in order to validate the surrogate model.

completed the parameters of the surrogate model are fitted to the input-output data.

2.1.2 Parameters rejected

There is no guarantee that the sample points are sufficient in order to tune the parameters of the surrogate model to accurately describe the behaviour of the complex model. When the parameter

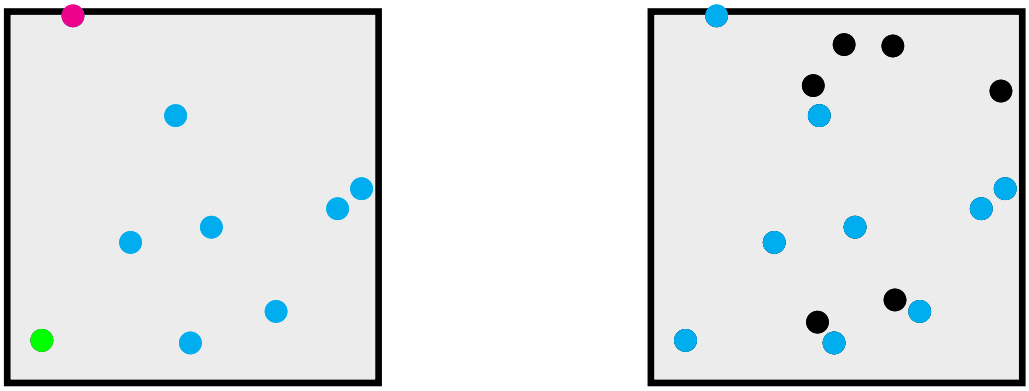

Figure 6: The figure illustrates how a subset of the simulated points are used for the validation of the surrogate model. The surrogate model and the simulation are compared and checked against an error criterion, and each point is either rejected or validated. In the case that any of the points are rejected, the framework will employ a strategy for improving the error of the surrogate model, adding **new points** to the domain.

fitting has been validated it can be used.

2.1.3 Execution

As illustrated in Figure 7, when the parameters of the surrogate model have been validated the surrogate model is executed by the application until it hits the boundary of the design space. In the event that

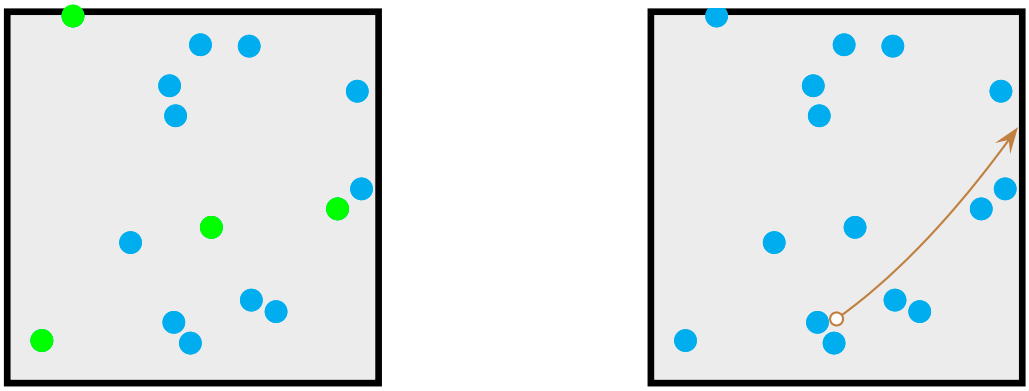

Figure 7: The figure illustrates that only a subset of the simulated points, not used for the parameter fitting of the surrogate function, is used for validation. When the surrogate model has been validated for a domain it can be evaluated, in this example allong a trajectory, untill it reaches the bounds of the domain.

the trajectory of the surrogate leads to the model being evaluated out of bounds it will throw an error that is caught by the software framework and automatically triggers the parameter estimation framework.

2.1.4 Out of bounds

When the surrogate model is used out of bounds the design space must be extended, the extended section must be sampled, and the MBDoE must be performed to re-fit the parameters globally.

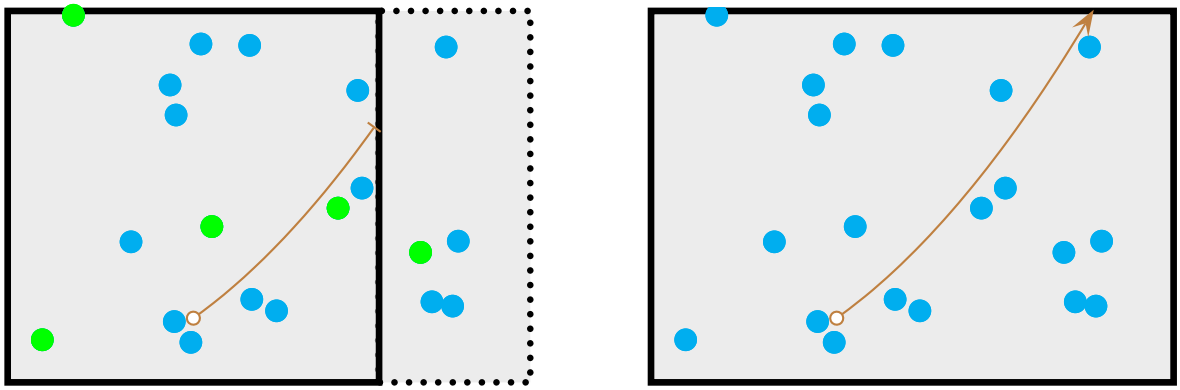

Figure 8: The figure illustrates the process of a simulation, where the surrogate function has been used along the trajectory, has been stopped, and the domain for the surrogate model is expanded to the dotted lines. New points are added to the domain and exact simulations are performed at every point before the surrogate function is fitted and validation globally across the entire domain. Afterwards, the simulation is re-started, and the process continues until the simulation is complete.

3 Implementation of the parameter estimation framework

The implementation of the backward mapping tool was explained from a technical perspective in deliverable 4.3 (Karolius, 2015), and the technical documentation is also available online (Rusche, 2015b). The backbone of the framework is the Python module FireWorks (Jain et al., 2015), which is employed as the workflow orchestrator for the MoDeNa software framework, and the data storage is facilitated by the NoSQL database MongoDB, and a Python API MongoEngine. The MoDeNa framework has been implemented in a object-oriented way, by formulating base-classes which inherits the properties from FireWorks and MongoEgine, depending on the job they have in the software.

The software framework can roughly be described as consisting of two parts: a "low-level" interface library, written in C, and a "high-level" model definition framework in PYTHON. The parameter fitting is a part of the high-level framework.

In order to make the framework flexible and easy to use the parameter estimation framework was implemented using a modular approach, which makes it easier for the developer of a scale-specific model to customize the model-based design of experiments algorithm from Figure 1.

3.1 Base classes

As mentioned in the introduction to the section the framework is in general built around a workflow management system, FireWorks, and the NoSQL database MongoDB. The strategies are classes inheriting properties from parent classes that are embeded into FireWoeks and MongoDB.

The base classes are the basic building blocks on which all the strategies are built, there is one base class for every strategy-type in the surrogate-model in Listing 2, and the different implemented strategies are further elaborated on in Section 5.

Initialisation Strategy that specifies how to instantiate the surrogate model.

Out of bounds Strategy which defines how to expand the range of validity.

Parameter estimation How the parameters of the surrogate model are determined.

Improve error Strategy defining how to go about attempting to improve the fit of the parameters.

3.2 Implementing new strategies

The modular approach of having interchangeable strategies based on the base classes leads to a framework that can easily be modified. A user implementing a new strategy must therefore create a new class, which inherits the property of the desired strategy type.

- Create a new class inheriting from the desired base class.
- Implement the algorithm as a method in the new class.

In practice it should not be necessary for the users to implement new strategies, both because it requires more in-depth knowledge of object-oriented programming, but also because the framework employs R in order to take advantage of the extensive statistical libraries. Specifically, the current version of the framework uses the packages "LHS" (Carnell, 2012) and "nlmrt" (Nash, 2014) for space filling and nonlinear regression respectively.

4 MoDeNa framework user-perspective summary

This section briefly summarises the basic necessary steps in order to formulate a surrogate model such that it can be used by the MoDeNa framework. The base classes for the data containers, which where thoroughly explained in Deliverable 4.2 (Karolius et al., 2015), are fundamentally different from the parameter estimation from the previous section. The data-containers are there to serve as data storage, i.e. it is where the parameter estimation framework retrieves and saves information.

4.1 Surrogate functions

The surrogate function base-class defines the global bounds, $\mathbf{U}_{max}$, of the inputs, outputs and parameters of the surrogate model. The code block in Listing 1 shows an example of the `CFunction` class, which is derived from surrogate function.

```
1  f = CFunction(
2  Ccode= '''
3  #include "modena.h"
4  #include "math.h"
5
6  void two_tank_flowRate
7  (
8  const modena_model_t* model,
9  const double* inputs,
10 double *outputs
11 )
12 {
13 {% block variables %}{% endblock %}
14
15 const double p1 = p0*inputs[3];
16
17 const double P0 = parameters[0];
18 const double P1 = parameters[1];
19
20 outputs[0] = M_PI*pow(D, 2.0)*P1*sqrt(P0*rho0*p0);
21 }
22 ''',
23   # These are global bounds for the function
24   inputs={
25     'D': { 'min': 0, 'max': 9e99 },
26     'rho0': { 'min': 0, 'max': 9e99 },
27     'p0': { 'min': 0, 'max': 9e99 },
28     'p1Byp0': { 'min': 0, 'max': 1.0 },
29   },
30   outputs={
31     'flowRate': { 'min': 9e99, 'max': -9e99, 'argPos': 0 },
32   },
33   parameters={
34     'param0': { 'min': 0.0, 'max': 10.0, 'argPos': 0 },
35     'param1': { 'min': 0.0, 'max': 10.0, 'argPos': 1 },
36   },
37 )
```

Listing 1: The code block shows how a user defines a surrogate function. There are four required parameters: the C-code in line 2, and the global domain of the inputs (line 24), outputs (line 30) and (line 33).

The purpose for all classes derived from surrogate function base-class should be to retain static information that is not changed throughout a simulation, because the surrogate model base-class is used explicitly by the parameter fitting module.

4.2 Surrogate model

The surrogate model base-class contains references to the parameter-estimation framework strategies. Moreover, as shown in Listing 2 it also contains the surrogate function from the previous section in line 3, and a reference to the exact model (line 4), which is not shown in this report. A function demonstration case can be found in Deliverable 5.5 (Rusche, 2015a), which tests the software framework.

```
1  m = BackwardMappingModel(
2    _id= 'flowRate',
3    surrogateFunction= f,
4    exactTask= FlowRateExactSim(),
5    substituteModels= [ ],
6    initialisationStrategy= Strategy.InitialPoints(
7    initialPoints=
8      {
9        'D': [0.01, 0.01, 0.01, 0.01],
10       'rho0': [3.4, 3.5, 3.4, 3.5],
11       'p0': [2.8e5, 3.2e5, 2.8e5, 3.2e5],
12       'p1Byp0': [0.03, 0.03, 0.04, 0.04],
13    },
14   ),
15   outOfBoundsStrategy= Strategy.ExtendSpaceStochasticSampling(
16     nNewPoints= 4
17   ),
18   parameterFittingStrategy= Strategy.NonLinFitWithErrorContol(
19     testDataPercentage= 0.2,
20     maxError= 0.05,
21     improveErrorStrategy= Strategy.StochasticSampling(
22       nNewPoints= 2
23     ),
24   ),
25 )
```

Listing 2: The code block shows an example of an instantiation of a backward mapping surrogate "flowRate". The surrogate model is instantiated using notation equivalent to a JSON document where every key has an associated value. The model defines a "surrogate function" in line 3, whose parameters will be fitted to the input-output data of the "exactTask" in line 4. The strategies in lines: 6, 15, 18 and 21 are triggered by events during the simulation.

The strategies in lines 6, 15, 18 and 21, shows how the parameter estimation framework is visible to the user in the MoDeNa software. They represents "slots" which the surrogate model base-class expects to be filled by the user, and can easily be switched out with corresponding strategies of the same type.

The next section summarises the available strategies in the current version of the MoDeNa framework as they would be used in the backward mapping surrogate model in Listing 2.

5 MoDeNa parameter estimation framework

This section summarises the current strategies, both strategy types, and the implemented algorithms in the MoDeNa framework. New strategies are added to the framework when requested by the users.

All the subsections represents base-classes for different types of strategies, and shows how they would be used in the surrogate model in Listing 2.

5.1 Initialisation

Initialisation is a necessary part of the life of any surrogate model, the goal is to fit the parameters of the model to a subspace of the global domain. There are many ways to instantiating the model, and the MoDeNa framework has currently implemented two strategies.

5.1.1 Initial Data

The initial data strategy is intended to be used in order to initialise the surrogate given sample points of both input and output data, e.g. from experiments. This means the user specifies both the input sample points and the corresponding output as shown below:

```
initialisationStrategy= Strategy.InitialData(
  initialData=
    {
      'D': [0.01, 0.01, 0.01, 0.01],
      'rho0': [3.4, 3.5, 3.4, 3.5],
      'p0': [2.8e5, 3.2e5, 2.8e5, 3.2e5],
      'p1Byp0': [0.03, 0.03, 0.04, 0.04],
      "flowRate" : [0.147002,0.150598,0.152887,0.148917],
   },
),
```

It can be verified that the keys includes all inputs and outputs by comparison with the surrogate function in Listing 1.

5.1.2 Initial Points

The initial points strategy initialises the surrogate model using a given set of input points. This means that the user provides the sample points for which the exact model is executed in order to provide output for the parameter fitting.

```
initialisationStrategy= Strategy.InitialPoints(
  initialPoints=
    {
      'D': [0.01, 0.01, 0.01, 0.01],
      'rho0': [3.4, 3.5, 3.4, 3.5],
      'p0': [2.8e5, 3.2e5, 2.8e5, 3.2e5],
      'p1Byp0': [0.03, 0.03, 0.04, 0.04],
   },
),
```

5.2 Out of bounds

The out of bounds base-class is used in order to expand the design space when the surrogate model is used out of bounds.

5.2.1 Expand design space using stochastic sampling

This strategy expands the design space a little bit beyond the point which caused the exception. The parameter `nNewPoints` determines how many samples from the extended space will be sampled.

```
outOfBoundsStrategy= Strategy.ExtendSpaceStochasticSampling(
  nNewPoints= 4
),
```

5.3 Parameter fitting

The parameter fitting base-class is responsible for parameter fitting and validation.

5.3.1 Nonlinear regression with error control

This strategy fits the parameters using nonlinear least squares regression on a randomly chosen subset of the sample points. The number of points that are used for the cross validation is determined by the parameter `testDataPercentage`, and which the model needs to satisfy is `maxError`. Finally there is a slot for yet another base-class of strategies that is used in order to select new samples in the event that the surrogate model is rejected.

```
parameterFittingStrategy= Strategy.NonLinFitWithErrorContol(
  testDataPercentage= 0.2,
  maxError= 0.05,
  improveErrorStrategy= Strategy.StochasticSampling(
    nNewPoints= 2
  ),
),
```

5.3.2 Nonlinear regression without error control

In the event that the user does not want the framework to attempt to reduce the error of the fit this strategy will perform a jackknife cross-validation of the parameter fitting. This means that if there are n samples there will be performed n regressions, each using $n-1$ variables for the fitting procedure and cross-validating using the point that is left out.

```
parameterFittingStrategy= Strategy.NonLinFitToPointWithSmallestError()
```

5.4 Improve parameter error

This base-class strategy is responsible for adding samples to a already existing design space in the event that a parameter fitting strategy fails.

5.4.1 Stochastic sampling

This strategy adds sample points to the design space using a latin hypercube sampling algorithm.

```
improveErrorStrategy= Strategy.StochasticSampling(
  nNewPoints= 2
),
```

6 Discussion and Conclusion

The parameter estimation framework in the MoDeNa software has been developed with a focus on flexibility, making it possible to adapt to future demands. The template syntax for defining surrogate models and specifying the strategies for parameter estimation shown in Listing 2 makes it easy for the end-user to change strategies without having to deal with the underlying framework.

The current state of the parameter estimation framework is ready for production release, and due to the use of `R` it can quickly be adapted by employing statistical algorithms that otherwise would have to be implemented.

Finally the parameter estimation framework as it stands in the MoDeNa software is not a standalone application, which could be an interesting direction to consider in the future. The reason for this is the advent of soft-modelling techniques, e.g. using partial least squares regression.

References

Carnell, R. (2012). *lhs: Latin Hypercube Samples*. R package version 0.10. 3.2

Franceschini, G. and Macchietto, S. (2008). Model-based design of experiments for parameter precision: State of the art. *Chemical Engineering Science*, 63(19):4846 – 4872. Model-Based Experimental Analysis. 1

Jain, A., Ong, S. P., Chen, W., Medasani, B., Qu, X., Kocher, M., Brafman, M., Petretto, G., Rignanese, G.-M., Hautier, G., Gunter, D., and Persson, K. A. (2015). Fireworks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, pages n/a–n/a. 3

Karolius, S. (2015). Modena - deliverable D4.3: Backward mapping tool. Technical report, FP7 MoDeNa. 3

Karolius, S., Birgen, C., Preisig, H. A., and Rusche, H. (2015). Modena - deliverable D4.2: Model and data containers. Technical report, FP7 MoDeNa. 4

Nash, J. C. (2014). *nlmrt: Functions for Nonlinear Least Squares Solutions*. R package version 2013-9.25. 3.2

Preisig, H. A., Thombre, M., and Karolius, S. (2014). Modena - deliverable D5.3: Parameter estimation framework. Technical report, FP7 MoDeNa. 2

Rusche, H. (2015a). Modena - deliverable D5.5: Testing of software framework v0.5. Technical report, FP7 MoDeNa. 4.2

Rusche, H. (2015b). *Software Framework MOdelling morphology of micro- and NAnostructures (MoDeNa) - technical documentation*. version 0.6. 3