UNIVERSITEIT GENT

FACULTY OF ENGINEERING AND ARCHITECTURE

SUrrogate Modeling (SUMO) Toolbox: Tutorial

Ivo Couckuyt

Tom Dhaene

http://sumo.intec.ugent.be

Outline

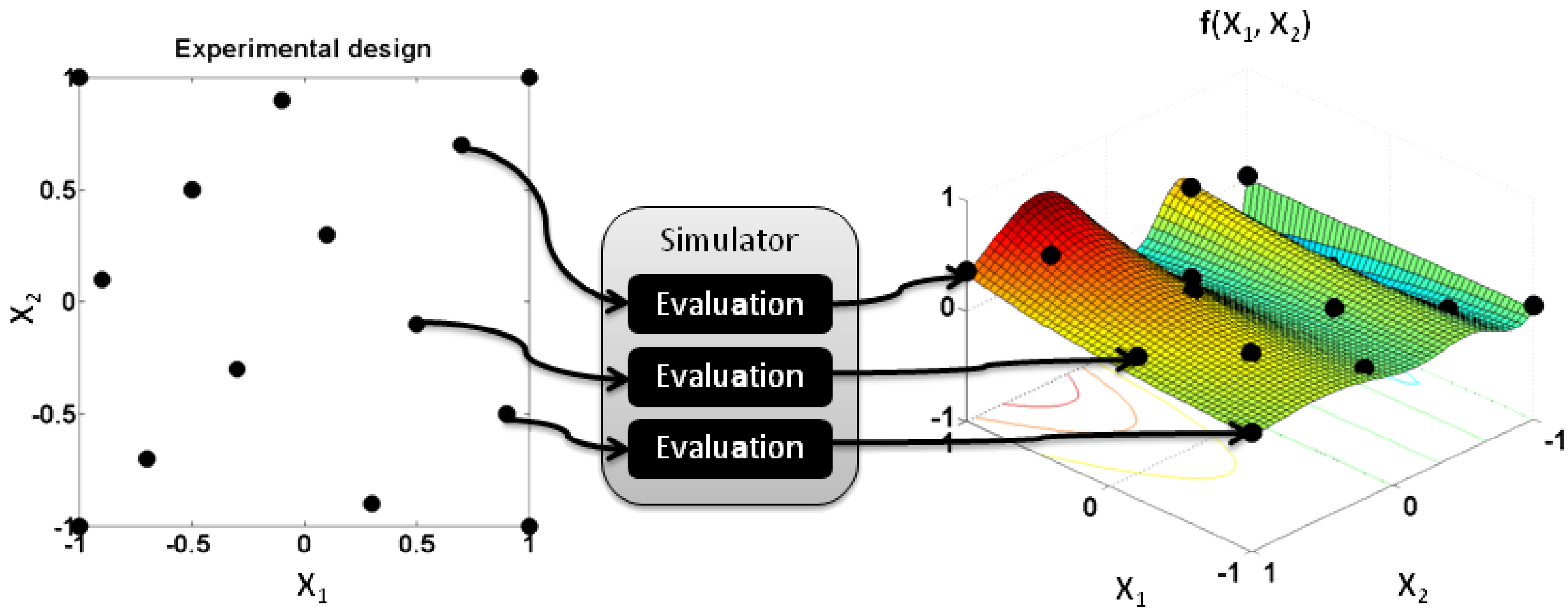
- **Surrogate modeling**
- **SUMO Toolbox**
- **Examples**
- **Conclusions**

Outline

- **Surrogate modeling**
  - Surrogate modeling
  - Sequential design
  - Adaptive surrogate modeling
- **SUMO Toolbox**
- **Examples**
- **Conclusions**

Outline

- **Surrogate modeling**
  - Surrogate modeling
  - Sequential design
  - Adaptive surrogate modeling
- **SUMO Toolbox**
- **Examples**
- **Conclusions**

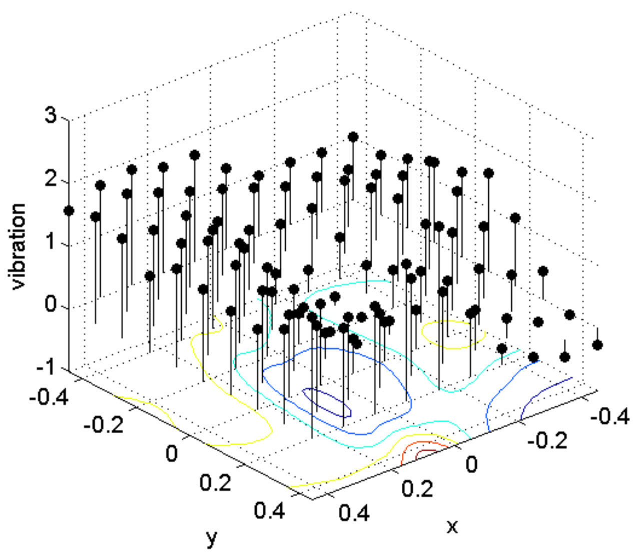


Experimental design
X2
X1
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1
Simulator
Evaluation
Evaluation
Evaluation
f(X1, X2)
1
0
-1
0
-1
1
0
-1
X1
X2

Simulator vs surrogate model

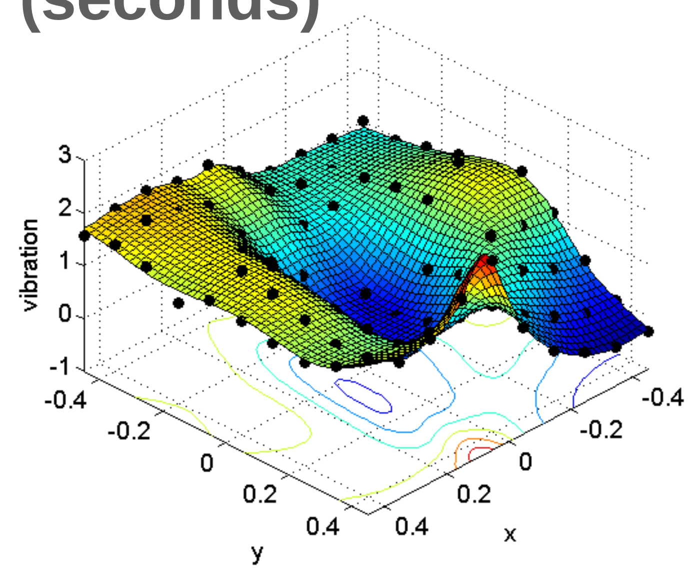
Simulator

- Based on physical equations
- Very accurate
- Slow (minutes or hours)

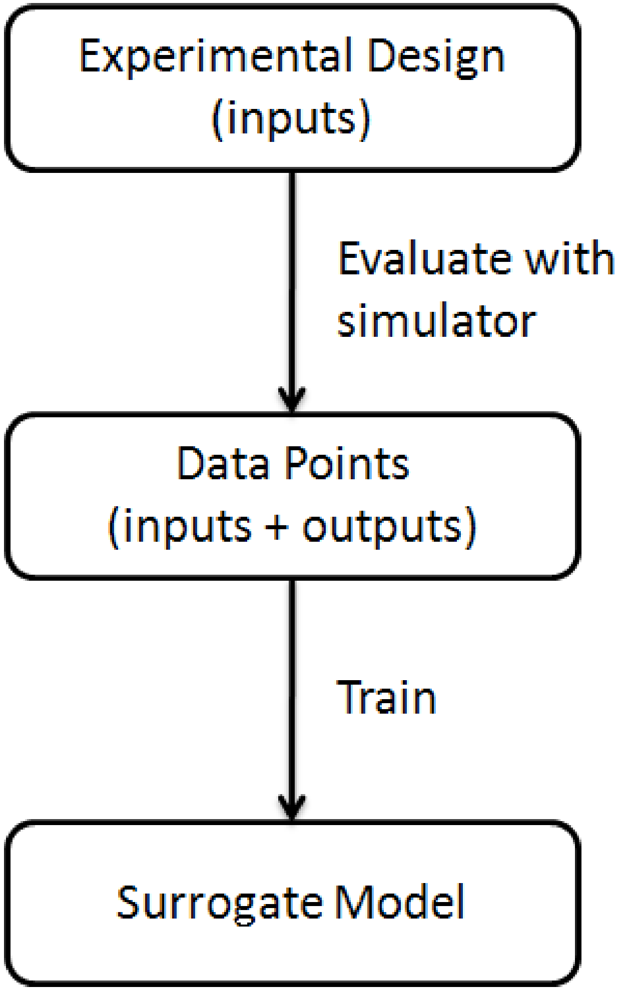


Surrogate model

- Based on maths equation
- Less accurate
- Extremely fast (seconds)



Surrogate modelling

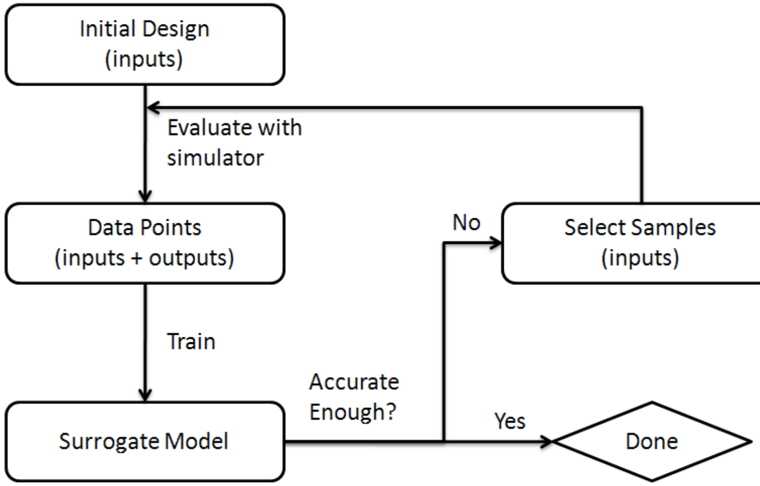


Outline

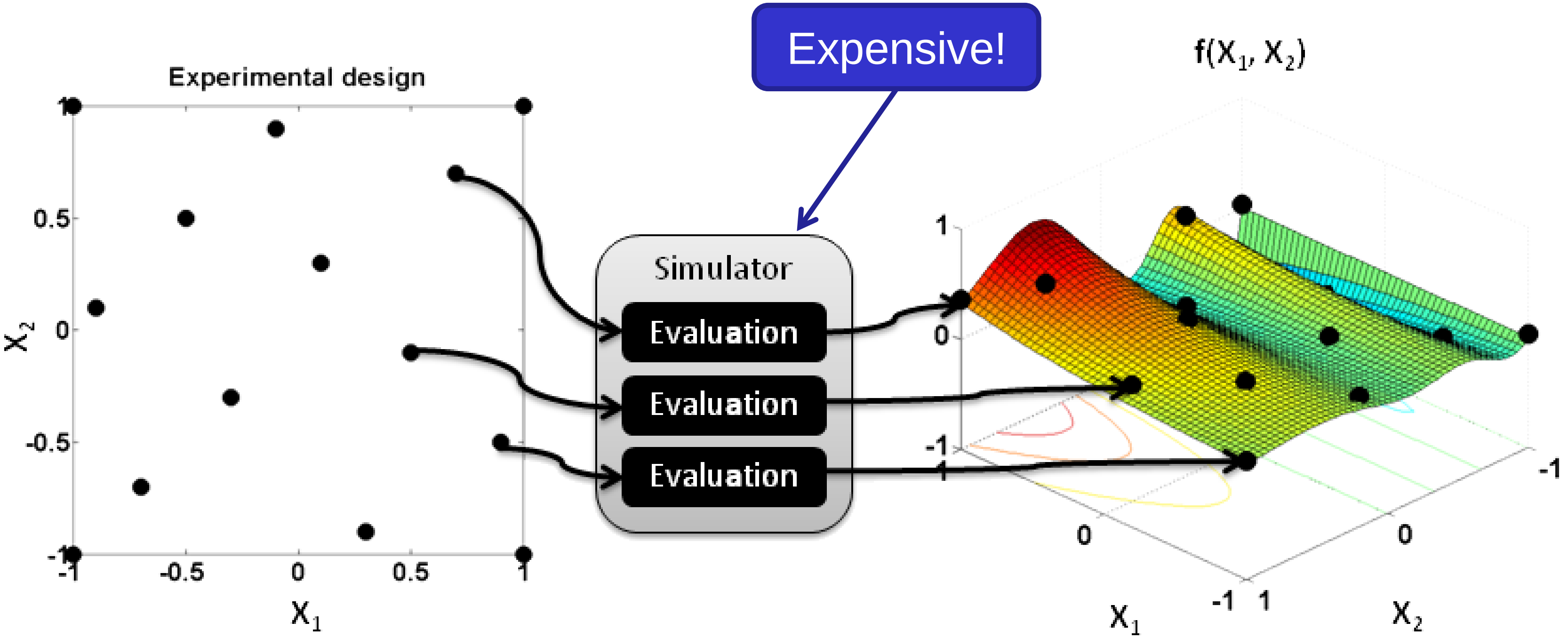
- **Surrogate modeling**
  - Surrogate modeling
  - Sequential design
  - Adaptive surrogate modeling
- **SUMO Toolbox**
- **Examples**
- **Conclusions**

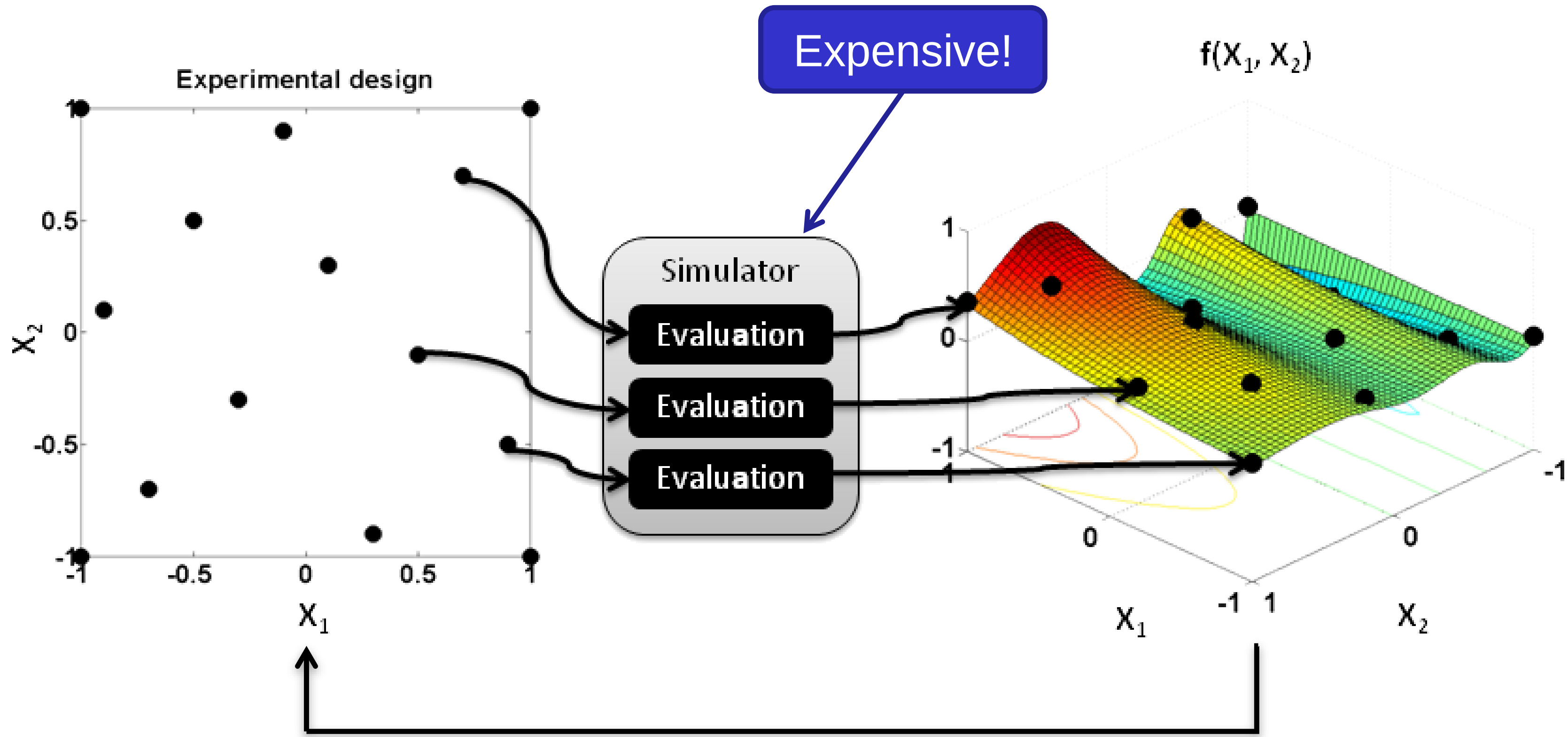
Sequential design

- Start with small set of initial simulations
- Build a surrogate model
  - Accurate enough? Stop
- Determine locations for additional simulations
- Repeat



Sequential design




Expensive!
Experimental design
$f(X_1, X_2)$
Simulator
Evaluation
Evaluation
Evaluation
X_1
X_2
X_1
X_2
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1
1
0
-1
-1
0
1
-1
0
1

Sequential vs one-shot design

Sequential design

- Samples 1 by 1
- No wasted simulations
- Use information from previous simulations to select new simulations more optimally

One-shot design

- Samples all at once
- Potential waste of simulations
- No information available to base experimental design on

Outline

- **Surrogate modeling**
- **SUMO Toolbox**
  - Installation
  - Walkthrough
  - Configuration
- **Examples**
- **Conclusions**

SUMO Toolbox

- **SUMO (SUrrogate MOdeling) Toolbox**
- **Adaptive surrogate modeling with sequential design**
  - Start with small set of initial samples
  - Sequentially select additional samples as required
  - After each sample selection, train a new surrogate model and adapt its model parameters to the data

SUMO Toolbox

- **Object-oriented Matlab interface**
  - Easy to use
- **Configuration through XML files**
  - Easy to configure
- **Pluggable and extensible framework**
  - Easy to tailor to your specific needs

Outline

- **Surrogate modeling**
- **SUMO Toolbox**
  - Installation
  - Walkthrough
  - Configuration
- **Examples**
- **Conclusions**

Installation

- **System requirements:**
  - Matlab 2008b (7.7) or later
    - **Use ‘ver’ command in Matlab to verify**
  - Java virtual machine (included in Matlab)
- **Optional Matlab toolboxes (recommended):**
  - Neural Network Toolbox
  - Genetic Algorithm and Direct Search Toolbox
  - Global Optimization Toolbox, Statistics Toolbox
  - Fuzzy Logic Toolbox

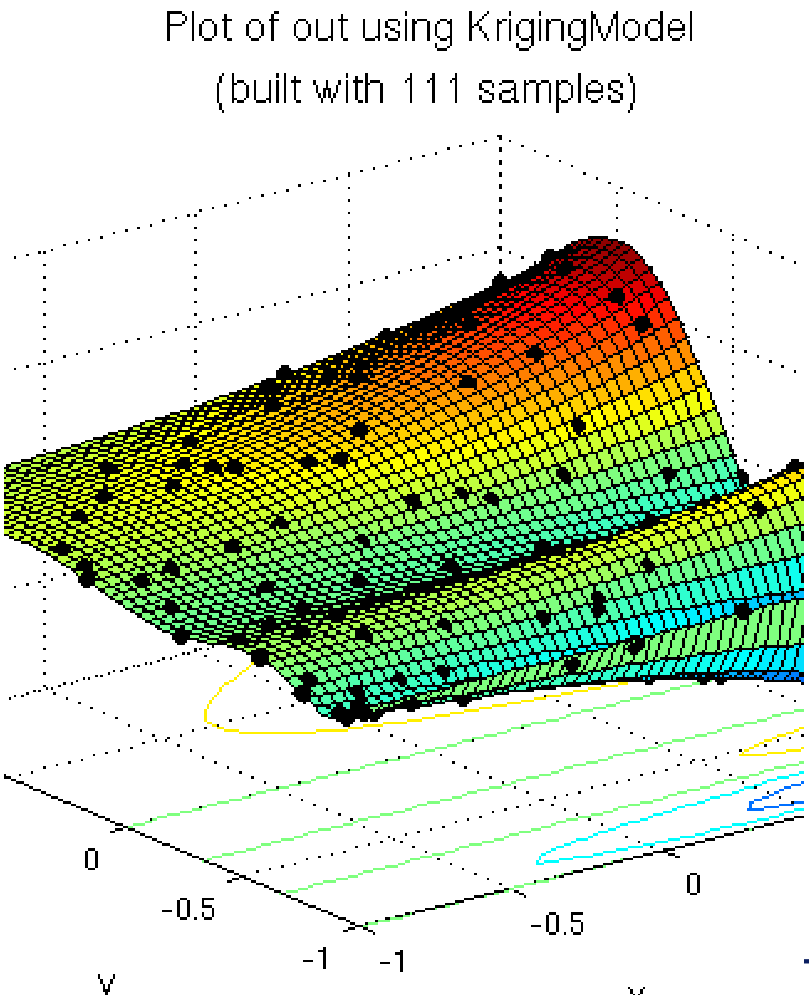
Installation

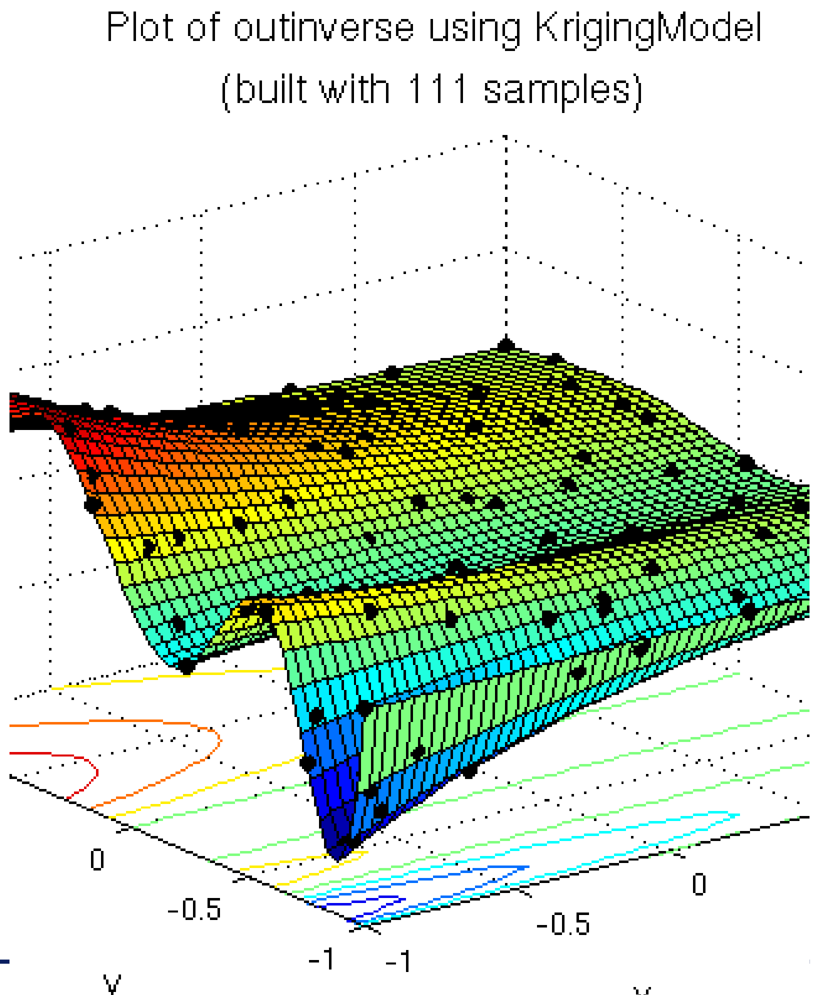
- **Download the toolbox zip file:**
  - http://www.sumo.intec.ugent.be/SUMO_download
- **Unzip on hard drive**
- **Start Matlab**
- **Inside Matlab:**
  - Navigate to the extracted SUMO Toolbox folder
  - Run the ‘startup’ command
  - This will configure the SUMO Toolbox

Installation

- **Type ‘go’**
  - SUMO should do a test run
  - Progress will be shown in the Command Window
  - A profiler window will open, displaying the various statistics of the test run
  - Two figures will be plotted showing the best model so far for “out” and “outinverse”
  - After a few minutes, the toolbox should halt

- **The final models should look something like this:**





Outline

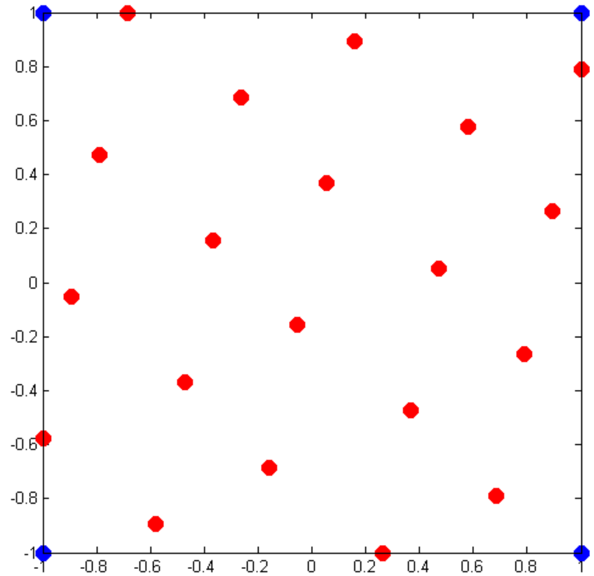
- **Surrogate modeling**
- **SUMO Toolbox**
  - Installation
  - Walkthrough
  - Configuration
- **Examples**
- **Conclusions**

- **SUMO uses two types of configuration files**
  - The main configuration file
    - **Just ‘go’ executes ‘config/default.xml’**
    - **To run a different configuration file: go(‘config/demo/demo-krigingAckley.xml’)**
  - The simulator configuration
    - **E.g., ‘examples/Math/Academic2DTwice.xml’**
    - **Defines the link between SUMO and the simulator**
      - E.g., 2 inputs and 2 outputs

Walkthrough

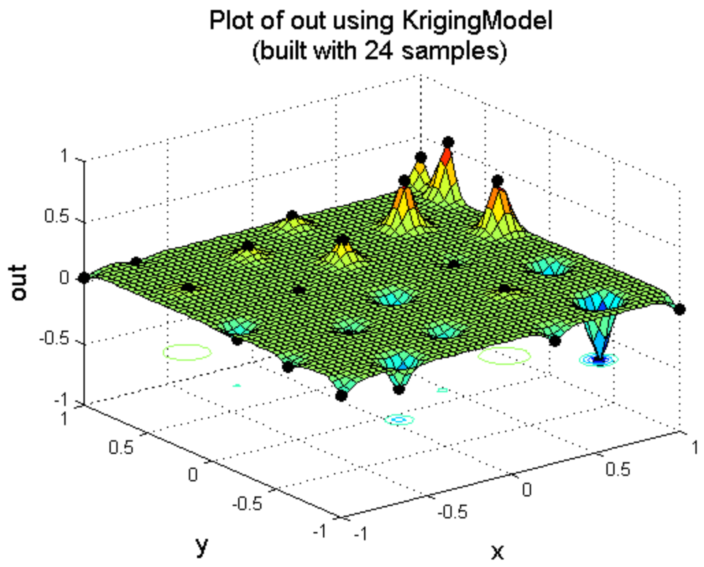
- **The components of 'default.xml'**
  - Surrogate model type: Kriging
    - **Good all-round model type**
  - Initial design: Latin hypercube with corner points
    - **Good coverage of entire 2D design space**
  - Sequential design strategy: LOLA-Voronoi
    - **Explores the design space, but focuses on nonlinear, difficult regions**
  - Simulator type: Matlab
    - **The simulator is a Matlab script**
    - **Can also be native executable, java code, dataset file**

- **The initial design is generated**
  - 20 points are selected in a Latin hypercube configuration
  - The 4 corner points are added



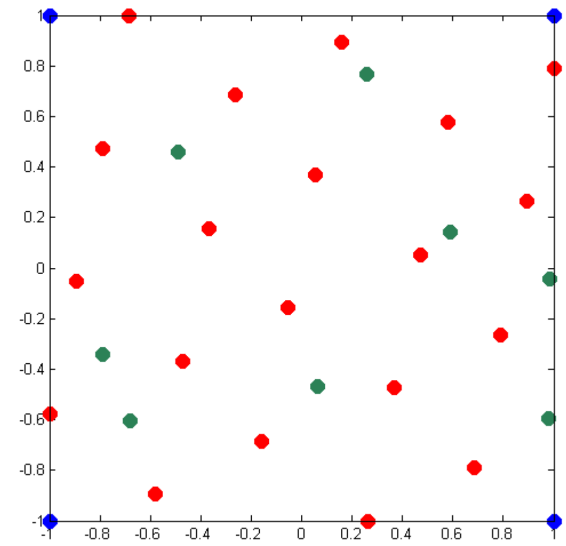
Walkthrough

- **Kriging models are built until no better model can be found with the current set of samples (= adaptive modeling iteration)**
  - The model parameter space is explored
  - The model parameters are adapted to suit the problem at hand



Walkthrough

- **New samples are selected using the sequential design strategy (= adaptive sampling iteration)**
  - Analyze error of previous models
  - Analyze previous samples to find important/difficult regions
  - Look for unexplored regions



Walkthrough

- **This process is repeated until one of the following is true:**
  - Minimum accuracy is reached (this case)
  - Maximum number of samples exceeded
  - Maximum run time exceeded

Walkthrough

- **A summary is printed**
  - Final model accuracy
  - Number of simulations performed (samples)
  - Elapsed time
- **Results are saved to disk**
  - All models built during the iterations
    - **plots and Matlab objects**
  - All samples evaluated
  - Detailed plots
    - **memory use, accuracy, minima/maxima, …**
- **Best model is plotted**

Outline

- **Surrogate modeling**
- **SUMO Toolbox**
  - Installation
  - Walkthrough
  - Configuration
- **Examples**
- **Conclusions**

- **Two configuration files:**
  - Simulator XML
    - **Defines number of inputs, outputs**
    - **Location of the Matlab script, or native executable, or dataset, etc**
    - **Constraints on the problem**
  - Main XML
    - **Different runs**
    - **Components used during each run**
    - **Configuration parameters for each component**
- **The .xml files can be edited with the Matlab editor (or any other editor)**

Main configuration

- **Example: config/default.xml**
- **Structure:**
  - The <plan> tag defines an experiment, and may contain multiple <run> tags
  - A <run> defines one run of the SUMO Toolbox as described in the walkthrough
  - For each run, a set of required components must be specified
  - These components can also be specified on the plan level, in which case they are used for all runs

Main configuration

- **The required components**
  - General options:
    - **<ContextConfig>**
    - **<SUMO>**
    - **<LevelPlot>**
  - Simulator link
    - **<Simulator>**
    - **<DataSource>**
  - Surrogate modeling algorithms
    - **<InitialDesign>**
    - **<ModelBuilder>**
    - **<SequentialDesign>**

Main configuration

- **The required components**
  - General options:
    - **<ContextConfig>**
    - **<SUMO>**
    - **<LevelPlot>**

    Can usually be left at default
  - Simulator
    - **<Simulator>**
    - **<DataSource>**
  - Surrogate modeling
    - **<InitialDesign>**
    - **<ModelBuilder>**
    - **<DataSource>**

Main configuration

- **Components**
  - <Simulator>
    - **Points to the Simulator directory (which contains the simulator XML)**
  - <InitialDesign>
    - **Defines the initial design**
  - <SequentialDesign>
    - **Defines the sequential design strategy**
  - <DataSource>
    - **Defines the data source: matlab script, dataset, …**
  - <ModelBuilder>
    - **Defines the model type and model parameter tuning strategy**

Main configuration

- **Each selected component points to a configuration section below the <plan> element**
  - For example: find “lhdWithCornerPoints”, the default <InitialDesign> setting
  - lhdWithCornerPoints is the composition of two other initial designs:
    - **Latin Hypercube design of 20 points**
    - **Factorial design of 2 points (= corner points)**

Main configuration

```
<Plan>
    <!-- Default components, these should normally not be changed unless you know what you are doing -->
    <ContextConfig>default</ContextConfig>
    <SUMO>default</SUMO>
    <LevelPlot>default</LevelPlot>

    <!-- This is the problem we are going to model, it refers to the name of a project
         directory in the examples/ folder.  It is also possible to specify an absolute
         path or to specify a particular xml file within a project directory -->
    <Simulator>Math/Academic/Academic2DTwice.xml</Simulator>

    <!--
    Runs can given a custom name by using the name attribute, a repeat attribute is
    also possible to repeat a run multiple times.  Placeholders available for run names include:
      #modelbuilder#
      #simulator#
      #sequentialdesign#
      #output#
      #measure#
    -->
    <Run name="" repeat="1">
        <!-- Enties listed here override those defined on plan level -->

        <!-- What experimental design to use for the very first set of samples -->
        <InitialDesign>lhdWithCornerPoints</InitialDesign>
```

...

```
    <!-- Specifies a combined Latin HyperCube and FactorialDesign -->
    <InitialDesign id="lhdWithCornerPoints" type="CombinedDesign">
        <!-- Select samples in a Latin Hypercube Design -->
        <InitialDesign type="TPLatinHypercubeDesign">
            <!-- how many points to generate -->
            <Option key="points" value="20"/>
            <!--<Option key="weight" value="0.5"/>-->
            <!--<Option key="coolingFactor" value="0.9"/>-->
            <!--<Option key="p" value="5.0"/>-->
        </InitialDesign>

        <InitialDesign type="FactorialDesign">
            <!-- how many points to generate for each dimension as a vector -->
            <!-- a scalar value (1) is the same as [1 1 ... 1] (length of input dimension) -->
            <Option key="levels" value="2" />
        </InitialDesign>
    </InitialDesign>
```

Simulator configuration

- **Example: 'Academic2DTwice.xml'**
  - Found in 'examples/Math/Academic'
  - 2 input parameters, named 'x' and 'y', both real-valued
    - **Only real-valued inputs are supported**
  - 2 output parameters, named 'out' and 'outinverse', both real-valued
    - **Real and complex outputs are supported**
  - A Matlab script that performs the simulation called Academic2DTwiceMatlab
    - **The Matlab script can be found in the same folder**
  - A grid dataset of 50x50 found in the same folder

Simulator configuration

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Simulator>
    <Name>Academic 2D Twice</Name>
    <Description>
        A predefined, two dimensional academic function with normal and inverse output.
    </Description>

    <!-- The input parameters -->
    <InputParameters>
        <Parameter name="x" type="real"/>
        <Parameter name="y" type="real"/>
    </InputParameters>

    <!-- The output parameters -->
    <OutputParameters>
        <Parameter name="out" type="real"/>
        <Parameter name="outinverse" type="real"/>
    </OutputParameters>

    <Implementation>

        <Executables>
            <Executable platform="matlab">Academic2DTwiceMatlab</Executable>
        </Executables>

        <DataFiles>
            <GriddedDataFile id="default" gridsize="50,50">Academic2DTwiceGrid</GriddedDataFile>
        </DataFiles>
    </Implementation>
</Simulator>
```

General information about the example

Information about the inputs

Information about the outputs

Datasets and scripts to generate data

Outline

- Surrogate modeling
- SUMO Toolbox
- Examples
- Conclusions

Example 1: how to run a different example

Example 1

- **Open Peaks.xml**
  - Found in examples/Math/Peaks
- **Observe:**
  - 2 inputs ‘x’ and ‘y’
  - 1 output named ‘out’
  - Matlab executable
  - 1 dataset
    - **Scattered**
    - **Can be used for validation**

Example 1

```xml
<Name>Peaks</Name>
<Description>
    Matlab's 2D Peaks demo function
</Description>

<!-- The input parameters -->
<InputParameters>
    <Parameter name="x" type="real" minimum="-5" maximum="5"/>
    <Parameter name="y" type="real" minimum="-5" maximum="5"/>
</InputParameters>

<!-- The output parameters -->
<OutputParameters>
    <Parameter name="out" type="real"/>
</OutputParameters>

<!-- A simulator may have multiple implementations: as an executable, a
java main class, a dataset, ...-->
<Implementation>

    <Executables>
        <Executable platform="matlab">PeaksSumo</Executable>
    </Executables>

    <DataFiles>
        <ScatteredDataFile id="default">Peaks2DScattered.txt</ScatteredDataFile>
    </DataFiles>
```

Example 1

- **You can change your configuration in two ways**
  - Replace an entire component
    - **by changing the reference in the <run> or <plan> tags**

  (= switching between components)
  - Modify the options of a component
    - **By changing the actual definition below the <plan> tags**

  (= fine-tuning of the component)

Example 1

- **Go back to default.xml**
- **Find <Simulator> in <plan>**
  - Change the path to Math/Peaks/Peaks.xml
    - **Name of xml file can be left out if it is the same as the folder name**
- **Find <outputs> in <run>**
  - Defines which outputs to model
  - Since Peaks.xml has has no output ‘outinverse’, delete this tag

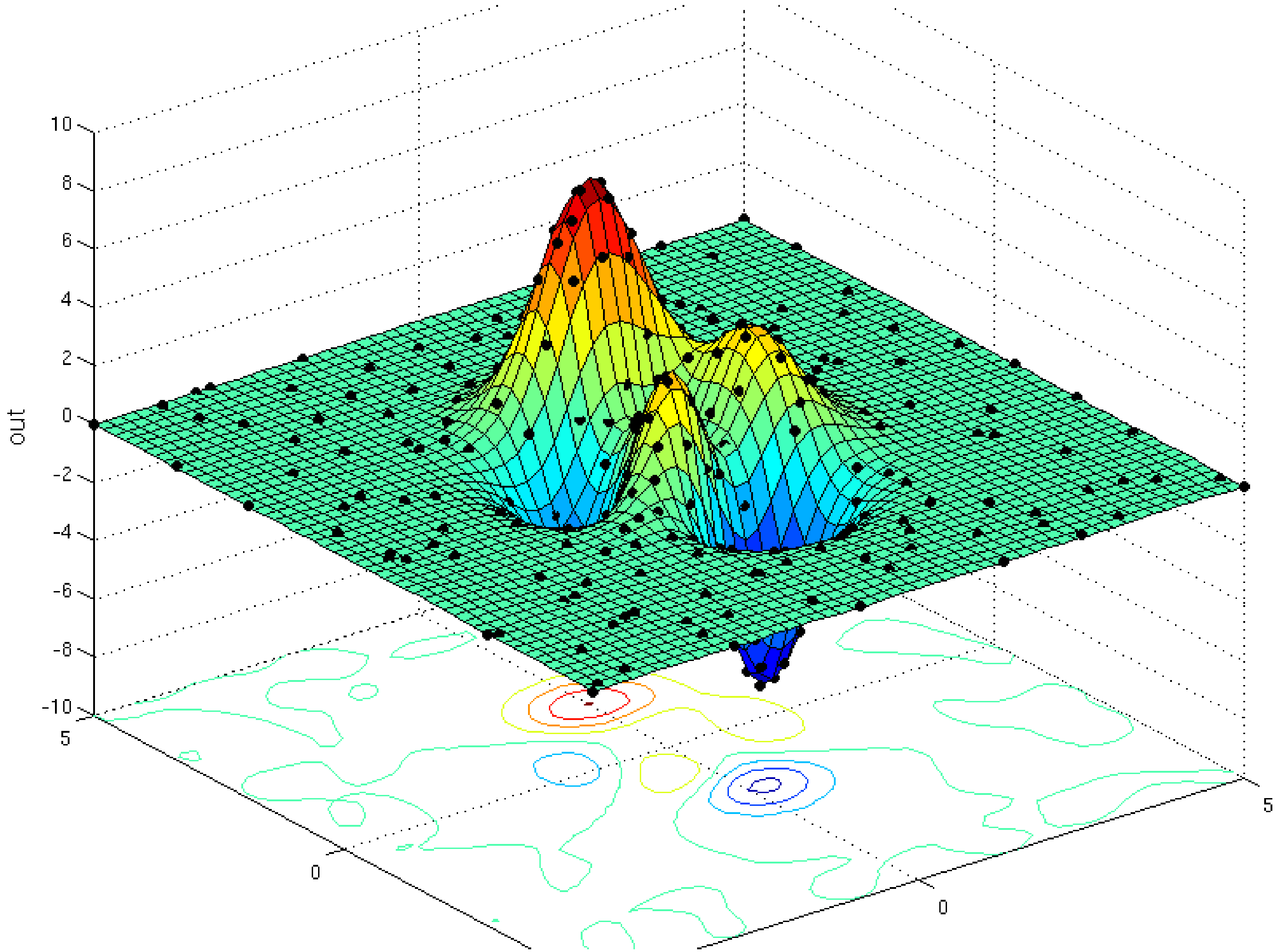
Example 1

```xml
<Plan>

    <ContextConfig>default</ContextConfig>
    <SUMO>default</SUMO>
    <LevelPlot>default</LevelPlot>

    <Simulator>Math/Peaks/Peaks.xml</Simulator>

    <Run name="" repeat="1">

        <InitialDesign>lhdWithCornerPoints</InitialDesign>

        <SequentialDesign>default</SequentialDesign>

        <DataSource>matlab</DataSource>

        <ModelBuilder>kriging</ModelBuilder>

        <Measure type="CrossValidation" target="0.01" errorFcn="rootRelativeSquareError" use="on" />

        <Outputs>
            <Output name="out">

            </Output>
        </Outputs>

    </Run>
</Plan>
```

Example 1

- **Save as default2.xml**
- **Navigate back to SUMO folder in Matlab**
- **Run go('config/default2.xml')**
- **Observe the output results**
  - Models converge slowly to 0.01 accuracy as more samples are selected
  - More samples are selected near the center, where there is a lot of nonlinearity (= lola-voronoi sample selector)
- **Abort the run by hitting ctrl+c**

Example 1: result



Example 2: how to configure a modelling run

Example 2

- **Change the sequential design to ‘density’**
  - Will uniformly spread points in design space
- **Change model builder to ‘ann’**
  - Artificial Neural Networks
  - Extremely accurate model, but slow to train
- **Run go(‘config/default2.xml’)**

Example 2

```
<Plan>

    <ContextConfig>default</ContextConfig>
    <SUMO>default</SUMO>
    <LevelPlot>default</LevelPlot>

    <Simulator>Math/Peaks/Peaks.xml</Simulator>


    <Run name="" repeat="1">

        <InitialDesign>lhdWithCornerPoints</InitialDesign>

        <SequentialDesign>density</SequentialDesign>

        <DataSource>matlab</DataSource>

        <ModelBuilder>ann</ModelBuilder>

        <Measure type="CrossValidation" target="0.01" errorFcn="rootRelativeSquareError" use="on" />


        <Outputs>
            <Output name="out">

            </Output>
        </Outputs>

    </Run>
</Plan>
```

Example 2

- **Observe**
  - Points are now spread out evenly due to ‘density’ sequential design strategy
  - Slow modelling speed due to ‘ann’
  - Higher accuracy than previous run

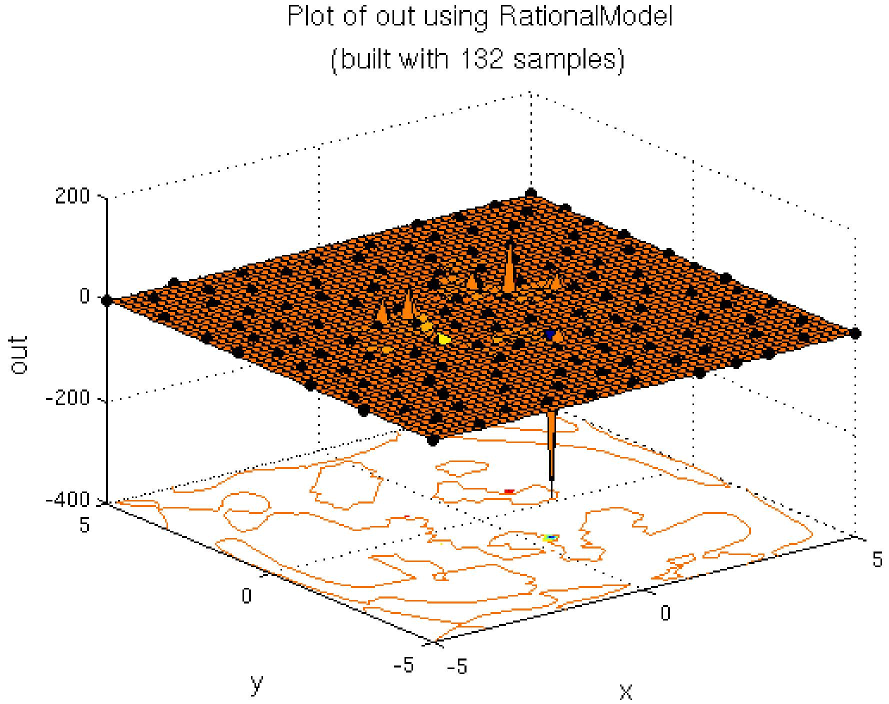
Example 3: running the rational model

Example 3

- **Keep the sequential design to ‘density’**
  - Will uniformly spread points in design space
- **Change model builder to ‘rational’**
  - Rational function
  - Unpredictable: can give very good and very bad results
- **Go to SUMO config (id ‘default’)**
  - Find option ‘minimumAdaptiveSamples’ and change it to 100 (all newly selected samples must be evaluated before new models are trained)
- **Run go(‘config/default2.xml’)**

Example 3

- **Observe**
  - Rational model builder fails to create good models



Example 4: visualizing the result and using the model

Example 4

- **Try simulator ‘ElectroMagnetics/StepDiscontinuity’**
  - 3 inputs
  - 4 outputs, pick ‘S11’
- **Model with rational**
- **Use density sample selector**

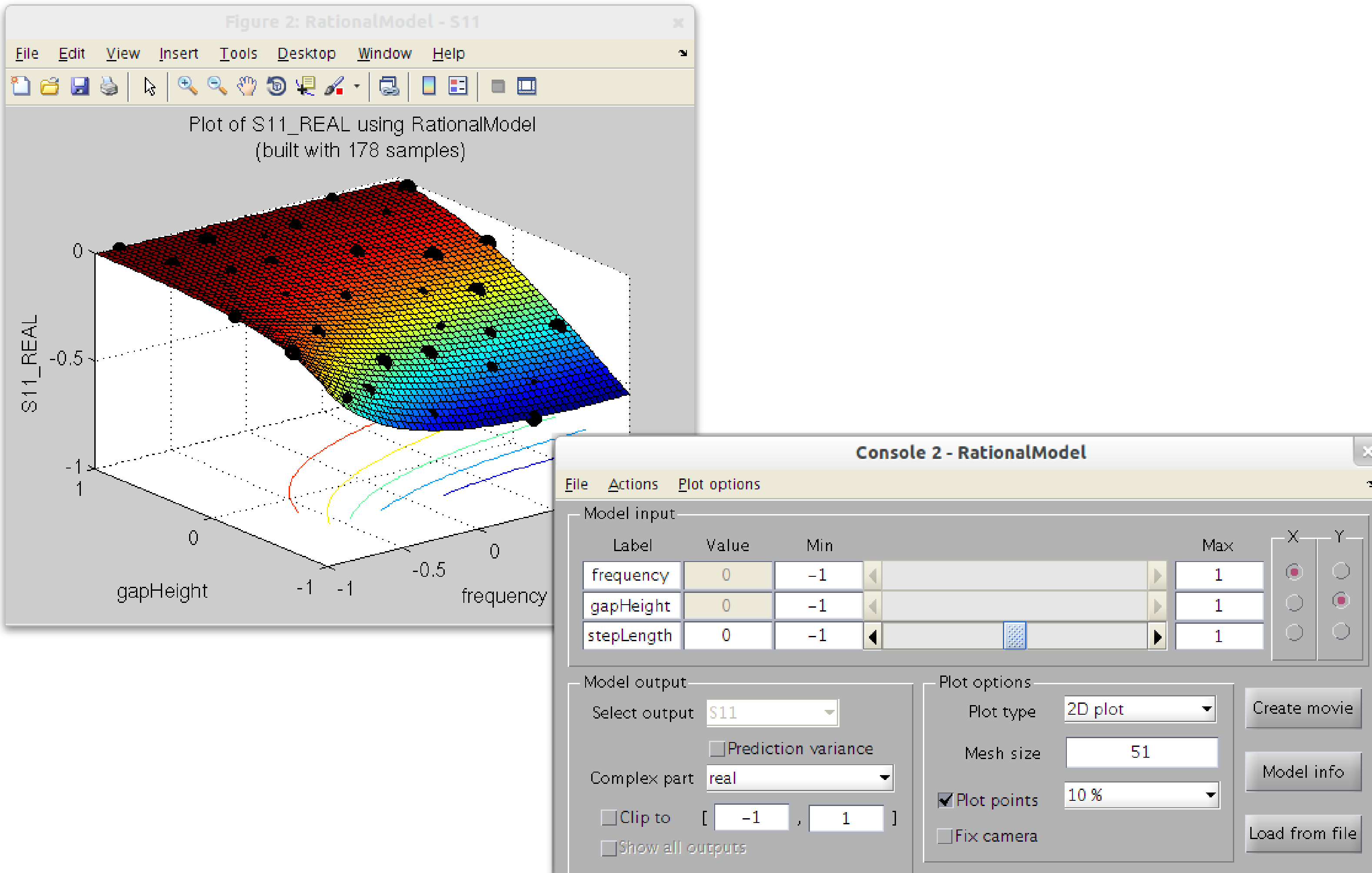
Example 4

- **Observe**
  - Rational works well on this problem
  - Visualisation of > 2D problems is tricky
  - SUMO can visualize slices of the data for higher dimensional problems
- **Browse to ElectroMagnetics/StepDiscontinuity/output/**
  - This directory contains all the runs performed for this example
  - The runs can be given custom names, but in this case the default name is used which is example_Model_dateStamp
  - Browse to the ‘best’ directory in directory of the current run

Example 4: exploring the model

- **Double-click model[S11].mat**
  - This loads the best model from the current run into the Matlab workspace with as variable name “model”
  - Type in the command windows “guiPlotModel(model)”
  - This will open a graphical user interface which allows you to explore the data. By adjusting the sliders different slice plots of the data will be shown.
- **Making an evaluation with the model**
  - Type in “y=model.evaluate([0.5,0.5,0.5; 0.7,0.7,0.7]))”
  - This will evaluate the model at (0.5,0.5,0.5) and (0.7,0.7,0.7)
  - The result should be close to: -0.0563 - 0.3780i and -0.0354 - 0.3294i
  - Type in ‘methods(model)’ or ‘model.<tab>’ to get a list of all functions available to this model

Example 4: exploring the model



Example 5: Surrogate-based optimization

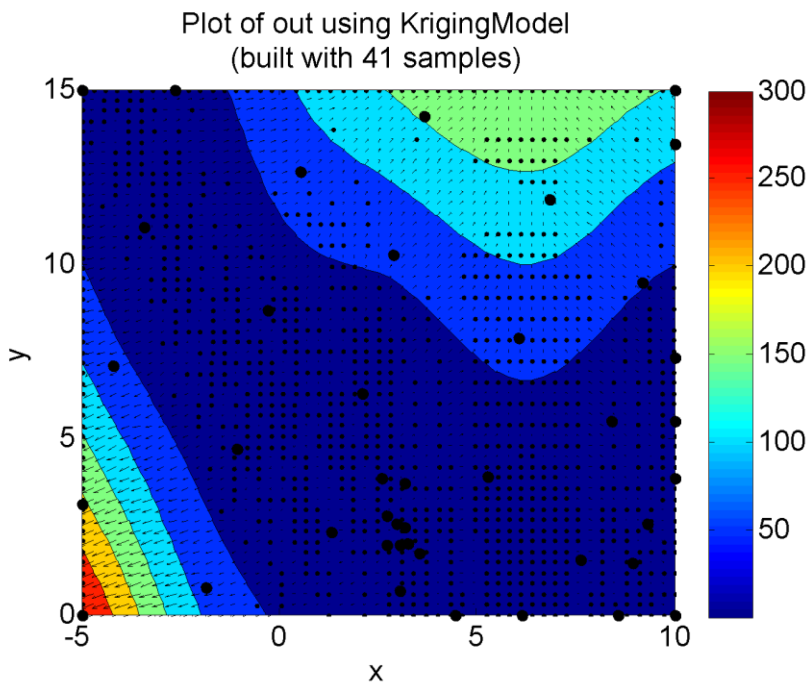
Example 5

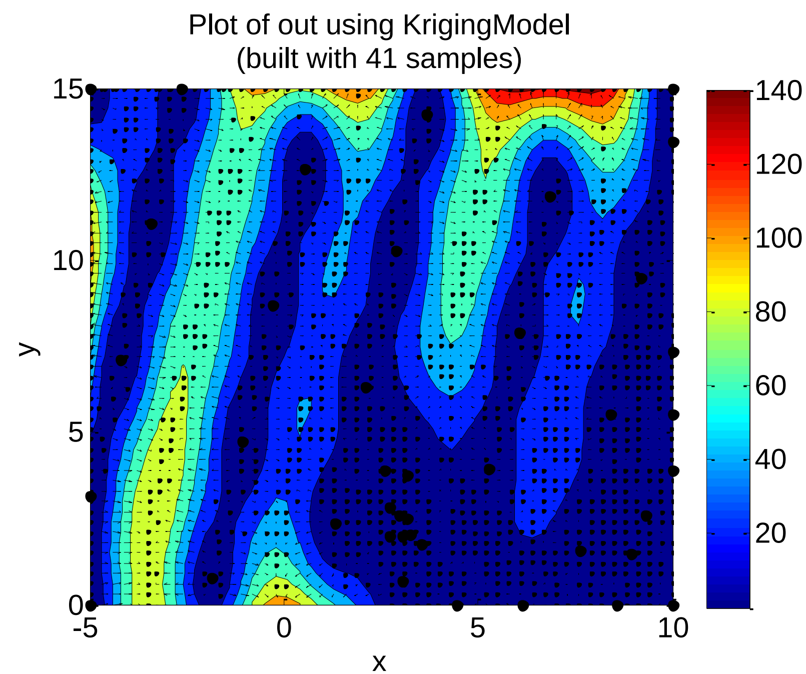
- **Try simulator ‘Math/Branin’**
  - 2 inputs
  - 1 output
  - Model with Kriging
- **Use expectedImprovement sequential design**
  - Set debug option to ‘on’
- **Kriging component**
  - Change BasisFunction option to ‘corrmatern32’
  - See list of available BasisFunctions

Example 5

- **Observe**
  - By selecting samples where the expected improvement is highest we can also optimize the simulator
    - **The surrogate model is a tool to an end, i.e., it is not necessarily accurate**
  - The debug plot shows the expected improvement criterion being optimized
  - Branin is an easy optimization problem
    - **But expected improvement is not so easy to optimize**
  - SampleMinimum profiler shows the progress of the optimization

- **‘guiPlotModel’ -> browse to and select kriging model**
  - Menu->Show->Derivatives
  - Check ‘Prediction variance’





Other things to try

- **Try different sequential designs**
  - Default (= 70% lola-voronoi + 30% error)
  - Error
    - **Select samples in locations where models disagree**
  - Density
    - **Spread out evenly**
  - Lola-voronoi
    - **Select samples in nonlinear regions**

Other things to try

- **Make your own sample selectors**
  - PipelineSequentialDesign
    - **Generates candidate samples (CandidateGenerator)**
    - **Score the candidates on some criteria (CandidateRankers)**
    - **Merge scores and select the best n candidates (MergeCriteria)**
  - OptimizeCriterion
    - **Optimizes a criteria (CandidateRankers)**
    - **Generate candidates (CandidateGenerator, optional)**
    - **Optimizes best candidate (Optimizer)**

Other things to try

- **Try different model types**
  - Kriging
    - **Interpolating**
    - **Default choice; works very well in most cases**
  - Rational
    - **Can be very accurate, but can also fail completely**
  - ANN (artificial neural networks)
    - **Very accurate, but extremely slow**
  - RBF (radial basis functions)
  - LSSVM (least squares support vector machines)
  - Heterogenetic
    - **Different models fight for survival, adapts the model type to the problem at hand**

Other things to try

- **<SUMO> settings**
  - minimumSamples/maximumSamples: number of samples selected at each sampling iteration
    - **Lower means more optimal sampling**
  - Stopping criteria: maximumTime, maximumTotalSamples, maxModelingIterations
  - minimumAdaptiveSamples: how much % of newly selected samples must be finished simulating before next modelling iteration starts