Objective calibration of climate models using multivariate quadratic emulation

Software documentation and example

Institute for Atmospheric and Climate Science, ETH Zürich

References:

Bellprat, O., S. Kotlarski, D. Lüthi, and C. Schär (2012), Objective calibration of regional climate models, J. Geophys. Res., 117, D23115, doi:10.1029/2012JD018262.

Bellprat, O., S. Kotlarski, D. Lüthi, R. De Elía, A. Frigon, R. Laprise, and C. Schär, (2016), Objective Calibration of Regional Climate Models: Application over Europe and North America. J. Climate, 29, 819-838, https://doi.org/10.1175/JCLI-D-15-0302.1

Dissertation: Bellprat, O. (2013), Parameter uncertainty and calibration of regional climate models. (Available online: https://doi.org/10.3929/ethz-a-009915303)

Omar Bellprat
March 28, 2018

Chapter 1

Introduction

This document gives an introduction for a parameter estimation procedure for complex computer models with very large data output. The methodology relies on Bellprat et al. (2012a) and has originally been designed for an objective calibration of model parameters in physical parameterizations of regional climate models using a calibration theory published in Neelin et al. (2010). The document presents a software package writtin in Matlab© which is available online on the Github repository: `https://github.com/C2SM-RCM/calibration`.

The present framework aims at minimizing the computational costs to calibrate unconfined model parameters in climate models. Presently many state-of-the-art climate models are tuned using expert knowledge and hand-tuning without following a well defined strategy. This condition inhibits for instance model comparison using two different parameterizations since the model will always depend on how the tuning is performed. The proposed procedure therefore is not only an approach to improve model performance but also to compare model parameterizations which have undergone the same tuning efforts.

Nonetheless, expert knowledge is still required in the proposed framework for instance for the definition of model parameters which are calibrated and their associated uncertainty ranges. This selection is a choice of the user, as well as the definition of a cost-function which is optimized. The tool is hence a way to facilitate high-dimensional problem solving given certain definitions which allows for a transparent and re-producible calibration.

The following chapters will guide a user for application of the methodology. The first chapter gives the methodological background. The second chapter aims at providing an overview of the code package and the work flow for a potential application. The final section gives an overview of all functions available in the package for more in-depth understanding.

Chapter 2

Method

The methodology of the calibration framework is documented in the reference publication, but a short summary of the important equations is summarized here. The basic idea of the framework is to build a computationally efficient statistical model that approximates the model output fields for an N-dimensional parameter space (often termed as model emulator, model surrogate, or *metamodel*). Using such a metamodel the high dimensional problem can be solved efficiently with common optimization procedures that require typically O^{5-9} simulations for parameter problems with 10 or less parameters. These number of simulations are not possible to perform with a climate model due to computational constraints of high-performance computing centers.

There are numerous approaches how to approximate model parameter experiments (e.g. Neural Networks or Gaussian Processes) and several approaches have been applied to climate models as discussed in Annan and Hargreaves (2007). The present framework relies on a metamodel that approximates the parameter space using a multivariate quadratic regression. This choice is motivated by the low number of model simulations that are required to account for basis non-linear behavior and parameter interactions. The use of a quadratic regression further inhibits over-fitting and allows for analytical solutions of the parameter space.

In order to estimate the multivariate quadratic metamodel only the boarders of the n-dimensional space have to be sampled. The Fig.2.1 illustrates such a design for two parameters (P1,P2) where in the center the default values (cross) and each dimension a minimum and maximum parameter value is sampled (circles). Additionally, in order to take parameter interactions into account, one corner point needs to be simulated (triangle) which corresponds to N*2+N(N-1)/2 simulations for N parameters.

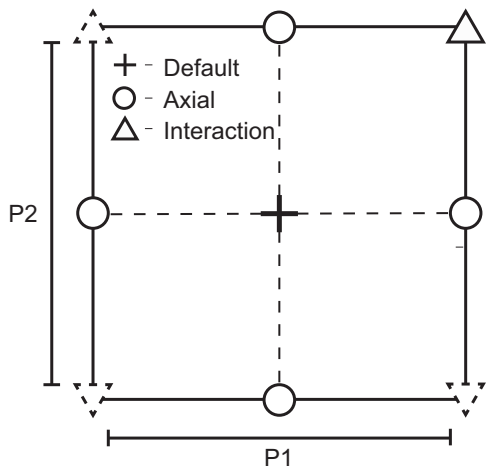

Figure 2.1: Koshal parameter experiment design required to estimate the parameters of the metamodel

Using the design points the following linear set of equations can be solved to estimate the linear term "a" and the non-linear term "B",

$$\Phi^* = \Phi_{ref} + \mu_*^T a + \mu_*^T B \mu_*. \tag{2.1}$$

where Φ corresponds to the data field which is approximated (Φ_{ref} is the reference field, Φ^* the predicted field by the metamodel) based on parameter vector $\mu*$ which describes the parameter values centered around the default value and normalized by the parameter range,

$$\mu_* = \frac{\mu_p - \mu_{def}}{max(\mu_p) - min(\mu_p)}. \tag{2.2}$$

Using centered parameter values has the advantage that the prediction of a parameter combination in one dimension is independent from the other parameter dimensions and therefore also their inaccuracies. The normalization avoids rounding errors.

The metamodel is by its definition quadratically smooth in space and only interaction between two parameters are allowed. This simplification is a good approximation of parameter experiments in climate models (Neelin et al., 2010, Bellprat et al., 2012b) but some inaccuracies because of this simplification can arise. In particular the estimation of the parameter interactions has proven to be mis-represented by this simplification. Possible reasons might be that three-way parameter are not determined even tough they seem to have important contributions in climate models (Rougier et al., 2009) or that the quadratic interaction term is not flexible enough to capture the true parameter interactions in the model. To constrain this uncertainty it is therefore recommended to use several simulation to estimate the interaction terms to achieve higher accuracy (Bellprat et al., 2012b). Other, more flexible regression models which have become popular are Gaussian Processes (GP) (Rougier et al., 2009, a Matlab code for GP emulation is provided here http://www.gaussianprocess.org/).

Using the fitted metamodel the agreement of the model with the observations can be maximized using performance function. This function is a choice of the user reflecting what he considers to be a "good" model. We choose a multivariate least-square estimation (Performance Score PS) considering uncertainty sources of predictability and observations as defined in Bellprat et al. (2012a).

Using a definition of the model performance the parameter space can be sampled to identify optimal parameter configurations. Finding such optimal configurations is, albeit the emulation of climate model, a tedious task in high dimensional parameter spaces. The most challenging problem is that within the parameter space local maxima may exist which need to be identified. Sampling the parameter space (with N number of parameters) using a grid design is inefficient in this case as the number of parameter combinations required grows with the exponent of parameters (power N). A further key limitation of grid sampling is the "collapsing" property: multiple parameter configurations have the same parameter value projected on the parameter axis. This limits the sampling in space and thus the small scale non-linearities i.e. local maxima are misrepresented.

Again numerous approaches from optimization theory exist to efficiently find these maxima. Here we use a Latin hypercube sampling (LHS) approach to reduce the dimensionality of the parameter problem. The approach has been introduced by McKay et al. (2000) and is widely used for determination of computer experiments. To illustrate the concept a comparison between a grid sample and a Latin hypercube sample is shown in Fig.2.2 again for two parameters. The baseline of the LHS is the number of parameter combinations M which are sampled. In case of a grid design for two parameters using a minimum, a center and maximum parameter value the number of parameter combinations corresponds to $M = 3^2 = 9$ experiments. Instead of sampling each combination for a fix number of intervals (grid design) the number of intervals in a LHS corresponds to the number of experiments M. The sample is subsequently drawn as such that each parameter combination uses a different parameter value as any other combination. This approach avoids the "collapsing" property of a grid design and makes the sampling independent of the number of parameters considered.

There are multiple samples which can be drawn for such a design corresponding to $(M!)^{N-1}$. The common approach to select one these samples is to use the sample with highest sum of Euclidean dis-

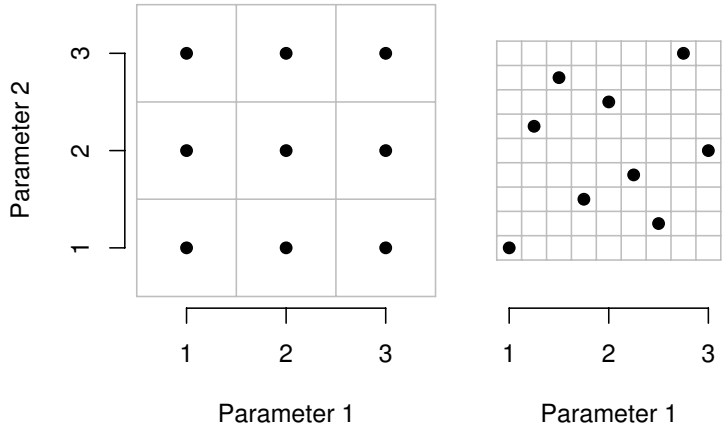

Figure 2.2: Comparison of grid-sampling and Latin hypercube sampling from Urban and Fricker (2010)

tances or to minimize the correlation structure between all parameter combinations. The LHS is used in the reference publication to find optimal parameter configurations using a 1 Million LHS for five parameters (i.e. 1 Million different values for each parameter sampled). Further a LHS sample is used to draw an independent parameter sample for a validation ensemble of the metamodel is used.

Alternative procedures for the calibration of the model parameters are possible using machine learning theory as Genetic Algorithms (Goldberg, 1989, toolbox integrated in Matlab as `gatool`) or Bayesian statistics as a Marcov Chain Monte Carlo (MCMC, Matlab code available here http://helios.fmi.fi/ lainema/mcmc/) integration. Generally these approaches are more efficient but require a multitude of additional assumptions. A further approach to constrain the parameter uncertainty without a calibration procedure is history matching, recently also applied in climate models (Williamson et al., 2013).

Chapter 3

Code structure and working flow

The structure of the program code is shown in Fig.3.1. Four main operation tasks are defined in square blocks, the definition of calibration suite, the estimation of the metamodel, the validation of the metamodel and the optimization of the model parameters. Relying on these main task, optional routines can be called which are shown in ellipsoid areas. The data is stored and accessed by three matlab structures termed *parameters* (contains information on parameters and experiments), *datamatrix* (contains simulated data of parameter experiments and validation sets) and *metamodel* (contains parameters and information on the fitted metamodel). All matlab routines can be called using these data structures, which are described with more detail in the next sections.

3.1 Definitions and data

3.1.1 *Parameter*

Within the *parameter* structure all information about the model parameters are stored. The individual fields describe for each parameter in *name* a string of the parameter name, *name_tex* the same name string using LaTeX standards for plotting, in *range* a vector with the minimum and maximum parameter value, in *default* the reference parameter value, in *experiments* the parameter values of all experiments used to fit the metamodel, in *constrain* additional parameter experiments outside of the Koshal design to further constrain the metamodel, and in *validation* the parameter values of the independent experiments used to validate the metamodel.

```
parameters =

1xN struct array with fields:
    name
    name_tex
    range
    default
    experiments
    constrain
    validation
```

3.1.2 *Datamatrix*

The *datamatrix* contains all data from the simulations which is needed to estimate and validate the metamodel, and to compute model scores if they are not emulated directly. A function to read-in data must be built by the user, an example how to read NetCDF data is provided. The individual fields of this structure contain in *moddata* all model data of the parameter experiments used to fit the metamodel, in *refdata* the data of the reference simulation, in *valdata* the data of the independent simulations for the validation of the metamodel, in *obsdata* the observations which are used to compute a model score if specified any, in *score* the name of the function that computes the model score (an example using *ps* is provided), in *constrain* the data of the simulations needed to additionally constrain

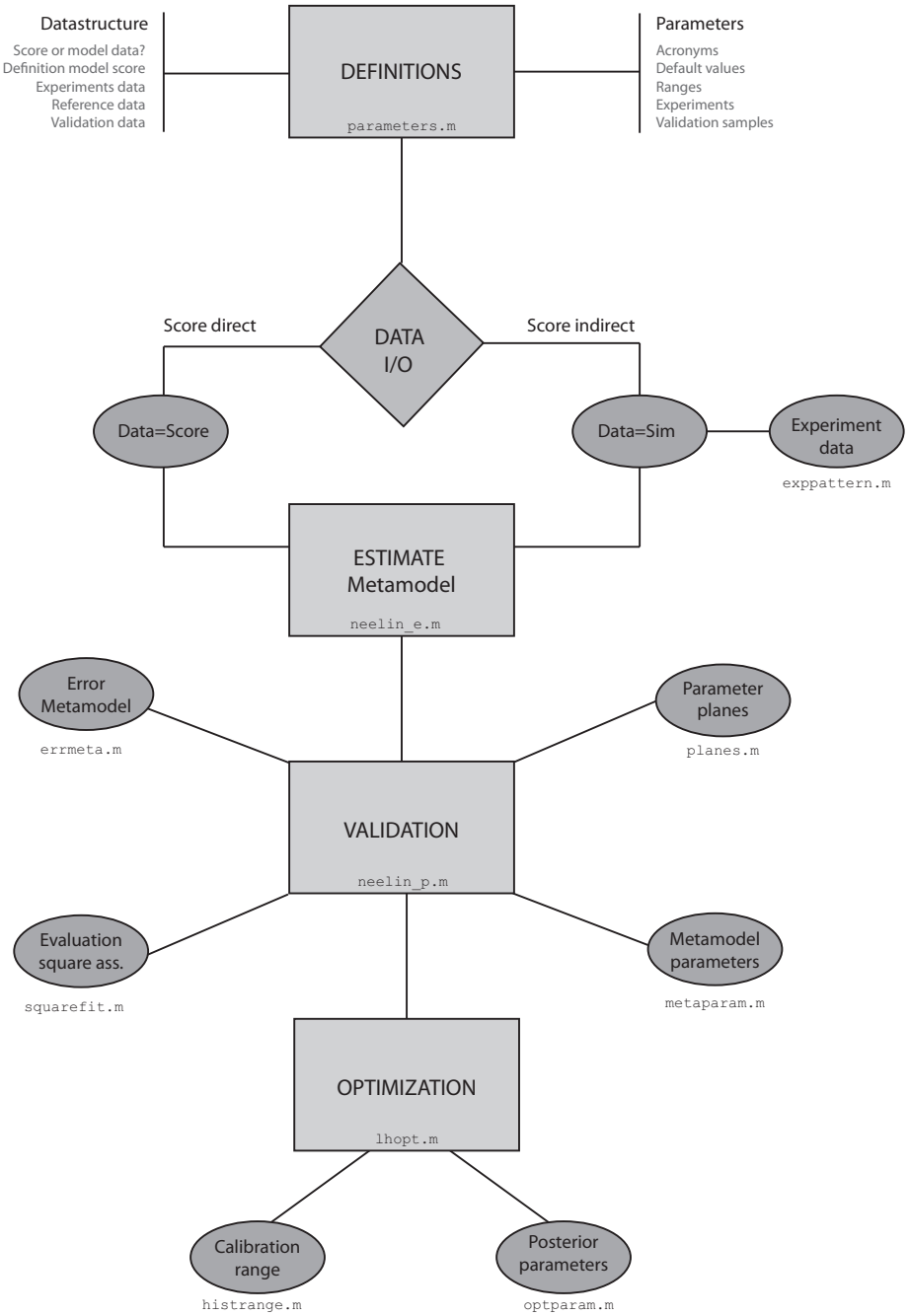

Figure 3.1: Schematic figure on the calibration framework. The framework is divided in four main tasks illustrated with square blocks: the definition of the calibration suite, the estimation of the metamodel, the validation of the metamodel and the optimization of the model parameters. For each task different functions are provided which are called using three data structures of the framework: the *datamatrix* (containing the data of the simulations and its definitions), the *parameters* (containing the information about the parameters and its values of the experiments), and *metamodel* (containing the estimated metamodel parameters). All functions need either all of these data structures or only two of them. More information about the functions are provided in the function alphabet of this document.

the metamodel. Further information of physical variables can be specified which are listed in the field *variables* that describes in the first term the index within the data structure that corresponds to different physical variables and in the continuation the strings of the variable names. Further physical limits for these variables can be set in *limits*, with a lower and upper bound for which each prediction with the metamodel is constrained.

```
datamatrix =

      moddata: [n+1-D double]
      refdata: [n-D double]
      valdata: [5-D double]
      obsdata: [4-D double]
        score: 'ps'
    constrain: [5-D double]
    variables: {[4]  'T2M'  'PR'  'CLCT'}
       limits: {[-Inf Inf]  [0 Inf]  [0 100]}
```

3.1.3 *Metamodel*

The estimated metamodel parameters are stored in the *metamodel* structure. For each data point a linear a and non-linear matrix B is estimated, the dimensions of field a and B therefore correspond to the dimensions of the data (n) plus a dimension for all parameters. In case additional simulations are used to constrain the metamodel a constant term c is used, otherwise only the data of the reference simulation is used as a constant term.

```
metamodel =
            a: [n+1-D double]
            B: [n+2-D double]
            c: [n+1-D double]
```

3.2 Metamodel estimation

The estimation solves the linear set of equations resulting from equation 2.1. The data matrix is for this purpose linearized such that the estimation is independent on the number of dimensions. This allows that the metamodel can be estimated on multivariate model data or integrated data of for instance a performance score directly (one value per experiment). The estimation of the metamodel is called using `neelin_e` which gives as an output the estimated metamodel.

```
metamodel = neelin_e(datamatrix,parameters)
```

The linear set of equation is solved by using the axial and default experiments (See Fig.2.1) for a quadratic regression in each parameter dimension, which can subsequently used to estimate the interaction term using the interaction experiments.

Depending on the number of experiments provided different approaches for the metamodel estimation are called. If the number of experiments is smaller than the number required for a Koshal design (2*N+N(N-1)) the estimation of the interaction terms is omitted and only the linear and quadratic term is estimated. If the number of experiments is larger than the linear set of equations the additional experiments are used to further estimate the interaction terms using a least square estimation. In case the number of experiments is smaller than 2*N the estimation can not be performed.
In case additional experiments are desired to use to further constrain the metamodel linear and squared parameters the function `neelin_c` can be used, which uses the definitions of datamatrix.constrain and parameters.constrain to narrow the uncertainty of the metamodel. This function should be called after the initial estimation of the metamodel as the following:

```
metamodel = neelin_c(datamatrix,parameters,metamodel)
```

3.3 Metamodel validation

Once the metamodel is estimated the metamodel can be evaluated with a number different analysis functions. The base function for this purpose is metamodel prediction function `neelin_p` which predicts the data of a given parameter experiment using the three base structures (metamodel, parameters, datamatrix) and a vector of parameter values for which the data should be predicted.

```
datamatrix = neelin_p(metamodel,parameters,datamatrix,pvector)
```

Given the prediction function independent simulations not used to estimate the metamodel can be used to assess the accuracy of the metamodel. This error of the metamodel can be estimated using the function:

```
[errstd]=errmeta(metamodel,parameters,datamatrix)
```

The function estimates the standard error to predict a number of experiments defined in datamatrix.valdata with parameter values defined in parameters.validation. Further a scatter plot of the simulated and predicted data points is produced.

Further metamodel analysis tools are the `planes` function, that plots pair-wise parameter planes predicted by the metamodel, `metaparam` which plots the metamodel parameters normalized by the variability of the data, and `squarefit` which allows to visualize the quadratic regression averaged for all the data points. If datamatrix.variables is specified, referring to physically different variables in the data, all the aforementioned figures are separated for these variables.

3.4 Parameter optimization

If the accuracy of the metamodel satisfies the needs of the problem the parameter space can be sampled in order to determine optimal model parameters. The calibration package offers for this purpose a Latin hypercube sampling that can be called using:

```
[lhscore lhexp popt]=lhopt(metamodel,parameters,datamatrix,lhacc)
```

The number `lhacc` determines the number of samples drawn from the LHS. The optimization yields for each sample a performance score (`lhscore` depending on the definition of the score), the experiment of the parameters (`lhexp`) and the optimal model parameter setting (`popt`). The optimal parameter corresponds to the parameter sets which according to the metamodel yields the highest score. However, given that the metamodel is associated with an uncertainty it is more reasonable to determine a distribution of parameter combinations which yield best model performance.

This distribution of optimized model parameters can be computed using the following function:

```
optparam(parameters,lhscore,lhexp,popt,errstd)
```

Additionally to visualize the whole performance space captured by the metamodel, a empirical distribution of the performance score can be visualized using the following command:

```
histplot(lhscore,datamatrix)
```

Chapter 4

Example

Given the defintion of all the functionalities of the code an example is provided based on an application of the methodology for the calibration of a regional climate model over Europe and North America ("Objective calibration of regional climate models: Application over Europe and North America"). The application bases on the calibration of Bellprat et al. 2012 but includes four additional parameters giving a total of eigth (N=8) parameters.

4.1 Defintions of parameters (structure: *parameters*)

We start the calibration by defining the parameter structure by giving names to parameter, default values, ranges and experiments. For this example the following definitions have been made in a matlab file that we use as a main file (e.g. calmo.m). The defintion begin with the names of each parameter for normal (`paramn`) and latex compatible (`paramnt`) format. The next matrix `range` defines for each parameter a range with a minimum and maximum value (e.g. the minimum value of the parameter rlam_heat is 0.1). The default vector of the parameters is given next (e.g. the default parameter of rlam_heat is 1).

```
%--------------------------------------------------------------------
% DEFINE Calibration structures (Parameters,Datamatrix)
%--------------------------------------------------------------------

paramn={'rlam_heat','entr_sc','qi0','uc1','rootdp','tkhmin','radfac','soilhyd'};

range={[0.1 10]; % Parameter ranges (min,default,max)
        [log(3E-5) log(3E-3)];
        [0 0.0001];
        [0 1.6];
        [0.5 1.5];
        [0.01 2];
        [0.3 0.9];
        [1 6]}

default={[1 roundn(log(3E-4),-4) 0.00000 0.8 1 1 0.5 1]};

expval=dlmread('parameterexp.txt'); % Experiment values to fit metamodel
valcon=dlmread('parametercon.txt'); % Experiment values to constrain the metamodel
valval=dlmread('parameterexp.txt'); % Experiment values to validate the metamodel

parameters=struct('name',paramn,'range',range,'default',default,
  'experiments', expval, 'constrain',valcon,'validation',valval,'name_tex',paramnt)
```

The parameter configurations used to estimate the metamodel are in this example read from a text file `'parameterlist.txt'` to avoid long script length. In this file each line consists of the paramater values chosen to estimate the metamodel. The structure of the experiments is currently fixed and has to follow these defintions: (1) The same order of parameters has to be kept as defined in the parameter structure. (2) First the minimum and maximum parameter experiment of all parameters have to be

listed. (3) After the mimimum and maximum experiment the interaction experiments are listed. So for two parameters p1 and p2 the experiments in the parameterlist.txt would need to be:
min_p1 p2_default
max_p1 p2_default
p1_default min_p2
p1_default max_p2
min_p1 max_p2 (or any other combination of interaction)

This list is much larger in the case of the current example since eigth parameters are considered. For the minimumum and maximum parameter experiment a total 2*N=16 and for the interaction N*(N-1)/2=28 parameter experiments are required. Further in the example all four interaction combinations are simulated which gives a total of 2*N+4*(N-1)=128 parameter combinations.

```
>cat parameterlist.txt
0.1     0.0003  0       0.8     1       1       0.8     1
10      0.0003  0       0.8     1       1       0.5     1
1       3e-05   0       0.8     1       1       0.5     1
.... (128 lines)
```

Optionally, further experiments can be defined to validate the metamodel (`valval`) and to further constrain the linear and quadratic terms of the metamodel (`valcon`) (instead of only using min, max and default values, additional points in between can be set).

4.2 Defintions of datamatrix (structure: *datamatrix*)

Having defined the parameters the data of the simulations need to be read. Reading this data into matlab has to be done by user as the data structure may differ from one application to another. There are two options to read the data, either to read the raw model data or to read a computed score for each simulation. However, reading only the score of the data reduces the accuarcy of the metamodel (as scores typically introduce further non-linearities) and hence it is recommenended to read the model data directly. An example how the defintion could look like is shown in here:

```

moddata=% Function to read experiment data to fit the metamodel
condata=% Optional: Function to read validation data to fit the metamodel
valdata=% Optional: Function to read validation data to fit the metamodel
obsdata=% Function to read to observations to compute a score
refdata=% Function to read the reference simulation;
variables={4,'T2M','PR','CLCT'} % Optional: Index of variables
  in moddata and name of each model variable
limits={[-Inf Inf],[0 Inf],[0 100]} % Optional: Physical limits of variables defined.
scoren='ps'; % Name of the function that computes the score.
datamatrix=struct('moddata',moddata,'refdata',refdata,'valdata',
  valdata,'obsdata',obsdata,'score',scoren,'variables',variables,'limits',limits)
```

For the example the data is already read and stored in ./data/datamatrix_example.mat. We therefore load this structure directly in to matlab using the following command

```
load('./data/datamatrix_example.mat')
datamatrix =
      moddata: [5-D double]
      refdata: [4-D double]
      valdata: [5-D double]
      obsdata: [4-D double]
        score: 'ps'
    constrain: [5-D double]
    variables: {[4]  'T2M'  'PR'  'CLCT'}
       limits: {[-Inf Inf]  [0 Inf]  [0 100]}
```

As shown by the summary of the datamatrix the data of the simulations have 4 dimensions (years, months, regions, variables). Further a score has been defined ('ps') which transforms the data to a performance score. If a different score is desired a new function that computes the desired score has to be formulated and called in the functions `planes.m`, `hist.m`, and `lhopt.m` (suggestions for a further development is an automatic integration of the score into all functionalities would be preferable). The

vector of the variables defines at which index of the data ('4') the different physical variables are stored. The following strings define the names of the variables which will be used e.g. in the function `errmeta.m`. (suggestion for the development is an automatic defintion of the variables bound to defitnions in the data I/O)

4.3 Estimation of the metamodel

Using the structures `parameteres` and `datamatrix` the metamodel can be estimated using the function `neelin_e.m` (neelin estimation)

```
metamodel=neelin_e(parameters,datamatrix)
```

This functions writes a new structure which contains the parameters of the metamodel as described in the previous section 3. Since the metamodel is estimated for every data point the parameters of the metamodel have the dimensions of the data plus one dimension for each parameter in the linear term in "a" and plus two dimensions in the quadratic term in "B" following the defintions of equation 2.1

```
metamodel =

    a: [5-D double]
    B: [6-D double]
```

In the current example all four interaction points are sampled (see schematic figure 2.1). As a result the linear system is overdetermined and the additional simulations are used to constrain the interaction term of the metamodel to increase its accuracy using least squares.
Having defined and computed all necessary datastructures all the following functions can be called using these structures (parameter, datamatrix and metamodel)

4.4 Evaluation and validation of the metamodel

Different functions are provided to evaluate and validate the metamodel. The functions are summarized in Fig.3.1 and in the function alphabet. The most important step to proceed with the calibration is to validate the metamodel to ensure that the approximation is accurate enough to determine an optimal configuration of the original model. To perform such a validation one ore more independent simulations are required which are predicted with the metamodel. The data of these simulations need to be stored in datamatrix.valdata and the the parameter values of these simulations in parameters.valval.
The function which determines the accuracy of the prediction is `errmeta` which produces scatter plots as shown in Fig.5.2. The function also computes the difference from the predicted and the simulated data which can be used to infer the uncertainty of the metamodel

```

% Error and standard error of the metamodel for each variable

[errpred stderr] = errmeta(metamodel,parameters,datamatrix)
```

If the level accuracy of the metamodel satisfies the requirements of the specific application one can proceed with the calibration. Otherwise the metamodel has to be constrained on additional regressions points (see `neelin_e.m` and `neelin_c.m`). If the metamodel remains inaccurate to predict the model data the metamodel is not suited for the application. As a consequence the problem has to be redefined or alternatively a more flexibile emulator has to be choosen (e.g. Gaussian Process).

4.5 Determining optimal parameter values

All necessary steps to perform the calibration are set and the parameter space can be sampled to find an optimal configuration. The sampling of the parameter space is here done using a Latin hypercube which linearises the parameter dimensions and allows for an efficicent sampling of high-dimensional spaces. The number of samples of the hypercube have to be specified in the variable `lhacc` which allows a maximum sampling aouround 10 6 depending on the memory allocated to Matlab. If the parameter space is very large (e.g. more than 10 parameters) is recommmended to use a sequential approach like

Marcov Chain Monte Carlo (MCMC) or Genetic Algorithms which will determine optimal parameter configurations more efficiently.
The optimization unsing the Latin hypercube can be called as follows:

```

% Number of samples
lhacc=1000000;
[lhscore lhexp popt]=lhopt(metamodel,parameters,datamatrix,lhacc)
```

The function generates for all samples the score (lhscore) the parameter configurations (lhexp) and the determines the parameter configuration which reaches the highest score (popt). Using output the calibrated parameter distributions can be visualized with the following function.

```
% Number of samples
```

Chapter 5

Function Alphabet

The available functions of the package are briefly described here with a short description of the functionality, an example of the produced figure (if any produced) and the description from the header of the code. The figures are based on an example of a calibration of the regional climate model COSMO-CLM for 8 parameters. The underlying description of the variables and observations selected are described in Bellprat et al. (2012a). Additionally, to the functions of the calibration framework, a very simple toy model `calmo_toy` is provided which allows to test the effect of noise in the experiment data and effect of structural error of the quadratic assumption.

5.1 exppattern.m

In order to visualize the effect of the model parameters from the different experiments a plot procedure is provided which summarizes the the difference of the experiments with respect to the reference.

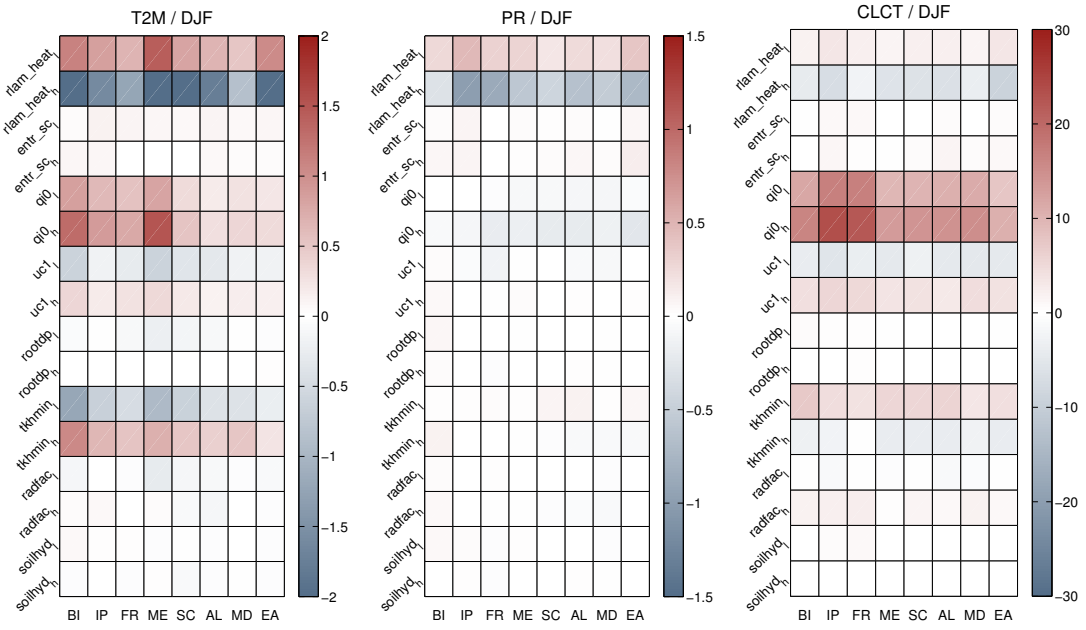

Figure 5.1: Sensitivity of parameter experiments divided in to variables, regions and seasons.

Header:

```
% Plot routine to visualize experiments for a Neelin fit
% NAME
%   exppattern
% PURPOSE
%   Create mosaic plots for dimesensions simulation, region, and varable
% INPUTS
%   From the structure datamatrix and paramters the
%   following fields are
%   processed (mind the same naming in the input)
%
%   datamatrix.moddata:
%
%           Model data for all experiments
%
%   parameters.name:
%
%           Name of parameter, parameter experiments
%
% OUTUTS
%   Pcolor plots given the number of model variables
% HISTORY
%   First version: 11.10.2013
```

5.2 errmeta.m

When using a validation set of simulations the function `errmeta.m` allows the user to estimate the prediction error of the metamodel. The error of the metamodel is further visualized using the scatter plots of the simulated and predicted data points as shown in Fig.5.2.

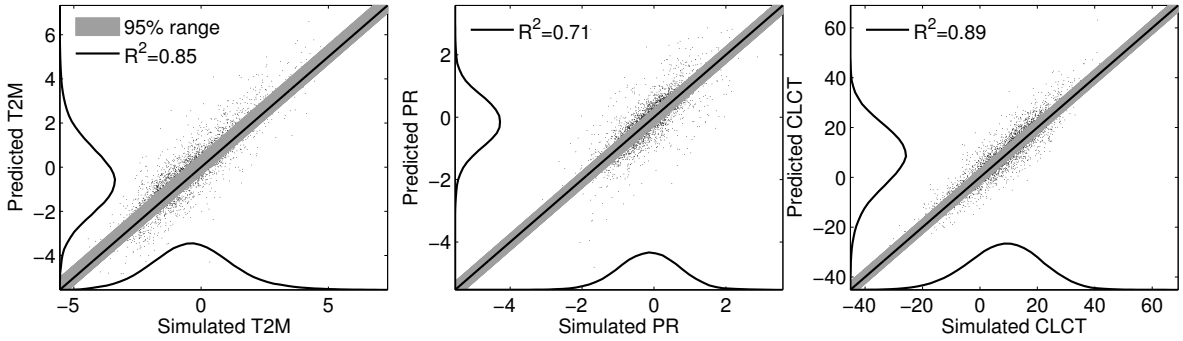

Figure 5.2: Simulated and predicted data points with the metamodel divided into different variables.

Header:

```
% Estimate the error of the metamodel to predict indpendent model information
% NAME
%   errmeta
% PURPOSE
%   Predict modeldata on of independt simulations and estimate the
%   standard error of the metamodel
%
% INPUTS
%   The structure metamodel, parameters and datamatrix
%   are used for the inpute of neelin_p. Addionally the parameter
%   matrix is read
%
%   parameters.experiments:
%
%           Parameter matrix of experiments on which metamodel is
%           estimated
%   phyd:
%
%           Index in the model data along which the prediction
%           error is physically seperated (ex: ind of different
%           model variabiles (temperature,precipitation,clouds))
% OUTUTS
%   Plot: Scatter plot for each defined variable of simulated and
%   predicted points
%   errstd: Standard error to predict the model data
%   errps: Standard error to predict the model score
% HISTORY
% First version: 11.10.2013
```

5.3 histplot.m

This functions allows to characterize the full calibration range of the parameter space in the for the specified performance function. Further the level of the reference function and the level of performance of the optimized configuration is plotted as shown in Fig.5.3.

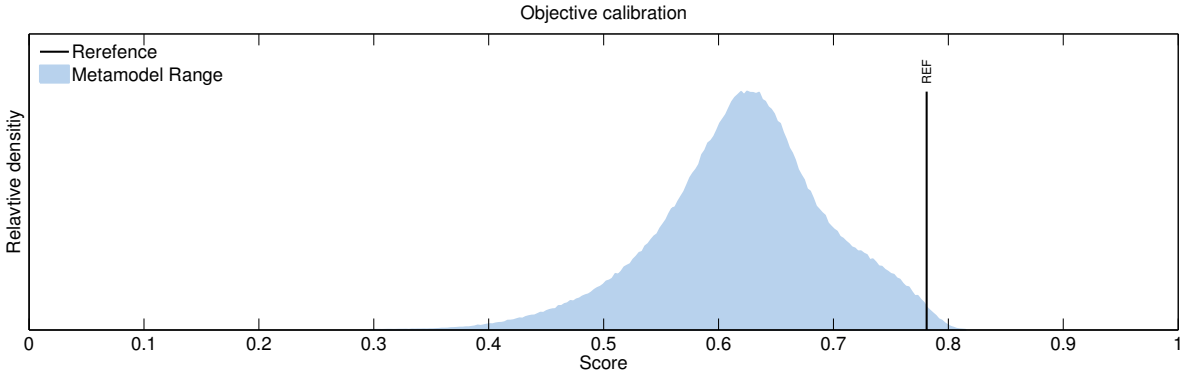

Figure 5.3: Calibration range of the Latin hypercube sample along the performance score (PS) in comparison to the reference (REF) simulation. The blue area above the reference simulation indicates the potential of the calibration to improve the model performance.

Header:

```
%   Plot full performance range sampled with the Latin hypercube
%   experiment as a histogramm
% NAME
%   neelin_p
% PURPOSE
%   Predict data using the metamodel for a parameter matrix
% INPUTS
%   From the structure metamodel, parameters and datamatrix the following fields are
%   processed (mind the same naming in the input)
%
%   datamatrix.reffdata:
%
%            Modeldata using default parameter settings to
%            determine/compute the model score of the reference
% OUTUTS
%   Plot: Histogram plot
% HISTORY
% First version: 11.10.2013
```

5.4 lhopt.m

The optimization of the optimal parameter configurations is done using a Latin hypercube sampling described in `lhopt.m`. For a given number of samples the function computes the predicted performance score of each sample and looks for the best combination. The output of the function is then further used to for `histplot.m` and `optparam.m`.

Header:

```
% Optimise model parameters using a Latin hypercube sampling
% NAME
%   neelin_p
% PURPOSE
%   Create a sample of parameters using a Latin hypercube design
%   and predict the model performance of the sample using the
%   metamodel.
% INPUTS
%   From the structure metamodel, parameters and datamatrix the following fields are
%   processed (mind the same naming in the input)
%
%   metamodel.a:
%
%          Vector of linear terms of the metamodel [...,N,1] additional
%          data dimensions possible (ex:a~[Regions,Variables,Time,N,1])
%
%   metamodel.B:
%
%           Matrix of quadratic and interactions terms [...,N,N] additional
%           data dimensions possible (ex:a~[Regions,Variables,Time,N,N])
%
%   parameters.range:
%
%            Range of values for each paramter to normalize the
%            scale.
%
%   parameters.default:
%
%            Default values of parameters to center the scale
%
%   datamatrix.reffdata:
%
%            Modeldata of when using default parameter values to
%            to center the datamatrix
%
%   pvector: Parameter values for one experiment with the
%            dimension of [N,1] N=Number parameters
% OUTUTS
%   dmatrix: Predicted data for parameter experiment
% HISTORY
% First version: 11.10.2013
```

5.5 metaparam.m

To assess the linear, quadratic and inter-action terms of the metamodel a function is provided which shows these terms when averaging for the whole data structure as shown in Fig.5.4.

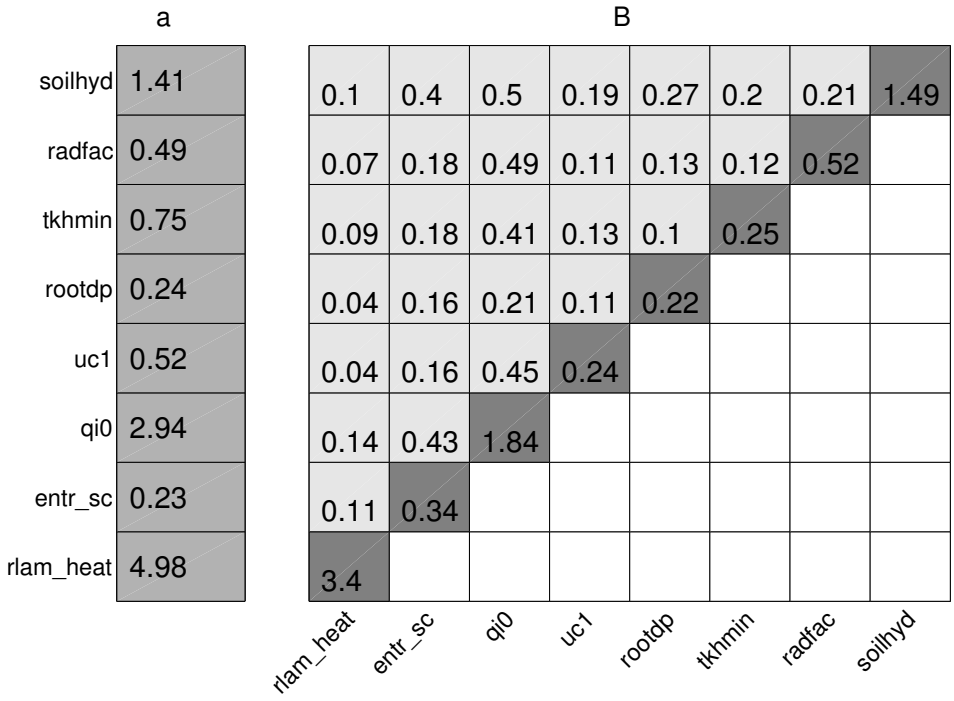

Figure 5.4: Parameter values of the metamodel divided into the linear contribution (first column on the left "a"), the quadratic contribution (diagonal values in the matrix "B"), and interaction values (off-diagonal values in the matrix "B"). The values are normalized by the range of the parameter values in order to compare the contribution of each term between the parameters.

Header:

```
% Visualize fitted metamodel parameters
% NAME
%   metaparam
% PURPOSE
%   Show normalized linear,quadratic and interaction terms for each
%   parameter and interaction
% INPUTS
%   The structure metamodel, parameters,datamatrix are used
% OUTUTS
%   Plot: Histogram of the difference between predicted and actual
%   model data
% HISTORY
% First version: 11.10.2013
```

5.6 neelin_e.m

Using the definitions in section 3.2 this function allows to estimate the metamodel creating a metamodel structure which contains all information to make a prediction of a given parameter set.

Header:

```
% Quadratic regression metamodel as described in Neelin et al. (2010) PNAS
% NAME
%   neelin_f
% PURPOSE
%   Estimate a mutlivariate quadratic metamodel which estimates quadratic
%   regressions in each parameter dimensions and computes interaction
%   terms for all pair of parameter experiments
% INPUTS
%   From the structure parameters and datamatrix the following fields are
%   processed (mind the same naming in the input)
%
%   parameters.experiments:
%
%            Parameter values for each experiment with the
%            dimension of [N, 2*N+N*(N-1)/2]
%            The structure NEEDS to be as follows
%            Example for 2 parameters (p1,p2)
%
%            [p1_l dp2 ] ! Low parameter value for p1 default dp2
%            [p1_h dp2 ] ! High parameter value for p1 default dp2
%            [dp1  p2_l] ! Low parameter value for p2 default dp1
%            [dp1  p2_h] ! Hihg parameter value for p2 default dp1
%            [p1_l p2_h] ! Experiments with interaction (no default)
%                        ! Additional experiments used to
%                        constrain interaction terms
%   parameters.range:
%
%            Range of values for each paramter to normalize the
%            scale.
%
%   parameters.default:
%
%            Default values of parameters to center the scale
%
%   datamatrix.moddata:
%
%            Modeldata corresponding to the dimenoins of
%            parameter.experiments
%
%   datamatrix.reffdata:
%
%            Modeldata of when using default parameter values to
%            to center the datamatrix
%            fitted. Not needed if metamodel fits score data directly
% OUTUTS
%   structure metamodel.
%   a: Metamodel parameter for linear terms [N,1]
%   B: Metamodel parameter for quadratic and interaction terms
%      [N,N]. Quadratic terms in the diagonal, interaction terms
%      in the off-diagonal. Matrix symetric, B(i,j)=B(j,i).
% HISTORY
% First version: 11.10.2013
```

5.7 neelin_p.m

Based on an estimated metamodel and a parameter set of chosen this function predicts the data that would result from a simulation with the specified parameter set.

Header:

```
1
2  % Forecast using regression metamodel as described in Neelin et al. (2010) PNAS
3  % NAME
4  %   neelin_p
5  % PURPOSE
6  %   Predict data using the metamodel for a parameter matrix
7  % INPUTS
8  %   From the structure metamodel, parameters and datamatrix the following fields are
9  %   processed (mind the same naming in the input)
10 %
11 %   metamodel.a:
12 %
13 %           Vector of linear terms of the metamodel [...,N,1] additional
14 %           data dimensions possible (ex:a~[Regions,Variables,Time,N,1])
15 %
16 %   metamodel.B:
17 %
18 %            Matrix of quadratic and interactions terms [...,N,N] additional
19 %            data dimensions possible (ex:a~[Regions,Variables,Time,N,N])
20 %
21 %   parameters.range:
22 %
23 %             Range of values for each paramter to normalize the
24 %             scale.
25 %
26 %   parameters.default:
27 %
28 %             Default values of parameters to center the scale
29 %
30 %   datamatrix.reffdata:
31 %
32 %             Modeldata of when using default parameter values to
33 %             to center the datamatrix
34 %
35 %   pvector: Parameter values for one experiment with the
36 %             dimension of [N,1] N=Number parameters
37 % OUTUTS
38 %   dmatrix: Predicted data for parameter experiment
39 % HISTORY
40 % First version: 11.10.2013
```

5.8 neelin_c.m

Additional simulations than needed to estimate the metamodel can be used to further constrain the linear and quadratic terms of the metamodel. By defining these parameter configuration and data in the structures `parameters` and `datamatrix` this function can be called as described in section 3.2

Header:

```
% Constrain linear and quadratic metamodel terms with additinal simulations
% NAME
%   neelin_c
% PURPOSE
%   Use additional simulations to narrow uncertainty of the
%   metamodel paramters, particulary if strong unequal distancies
%   between default and min/max values.
% INPUTS
%   From the structure parameters and datamatrix the following fields are
%   processed (mind the same naming in the input)
%
%   parameters.constrain:
%
%            Parameter values for each experiment for
%            additional parameter samplilng
%
%   parameters.range:
%
%            Range of values for each paramter to normalize the
%            scale.
%
%   parameters.default:
%
%            Default values of parameters to center the scale
%
%   datamatrix.moddata:
%
%            Modeldata corresponding to the dimenoins of
%            parameter.experiments
%
%   datamatrix.reffdata:
%
%            Modeldata of when using default parameter values to
%            to center the datamatrix
%            fitted. Not needed if metamodel fits score data
%            directly
%
%   datamatrix.constrain
%
%            Simulation data of experiments used to further sample
%            parameter ranges
%
% OUTUTS
%   Updatated metamodel structure
% HISTORY
% First version: 4.11.2013
```

5.9 `optparam.m`

To empirically estimate the posterior parameter distributions the current functions uses the uncertainty of the metamodel (one standard deviation from `errmeta.m`) and selects the best parameter combinations from `lhopt` which lie within this uncertainty. The following figure is then produced.

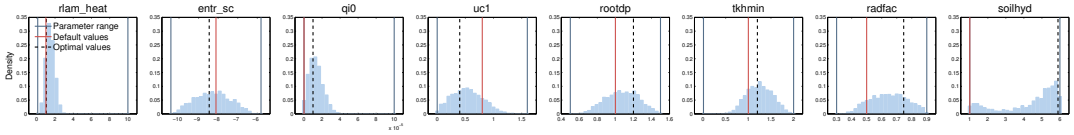

Figure 5.5: Distribution of the model parameters which lead to best model results. The distribution is computed from a sample of parameter combinations which, given the uncertainty of the metamodel (`errstd`), give the best model results. From this sub-sample of the Latin hypercube sample the empirical distribution of the values are plotted. This approach is an approximation of determination of a posterior parameter distribution without the definition of distribution priors.

Header:

```
1  %   Plot posterior parameter distributions given the uncertainty of
2  %   the metamodel prescribed in errm
3  % NAME
4  %   optparam
5  % PURPOSE
6  %   Plot parameter distributions which lead to best model results
7  %   given the uncertainty of the metamodel
8  % INPUTS
9  %   From the structure metamodel, parameters and datamatrix the following fields are
10 %   processed (mind the same naming in the input)
11 %
12 %   datamatrix.reffdata:
13 %
14 %            Modeldata using default parameter settings to
15 %            determine/compute the model score of the reference
16 % OUTUTS
17 %   Plot: Histogram plot
18 % HISTORY
19 % First version: 11.10.2013
```

5.10 planes.m

This routine allows to show the variations in the specified model performance along all planes of parameter pairs.

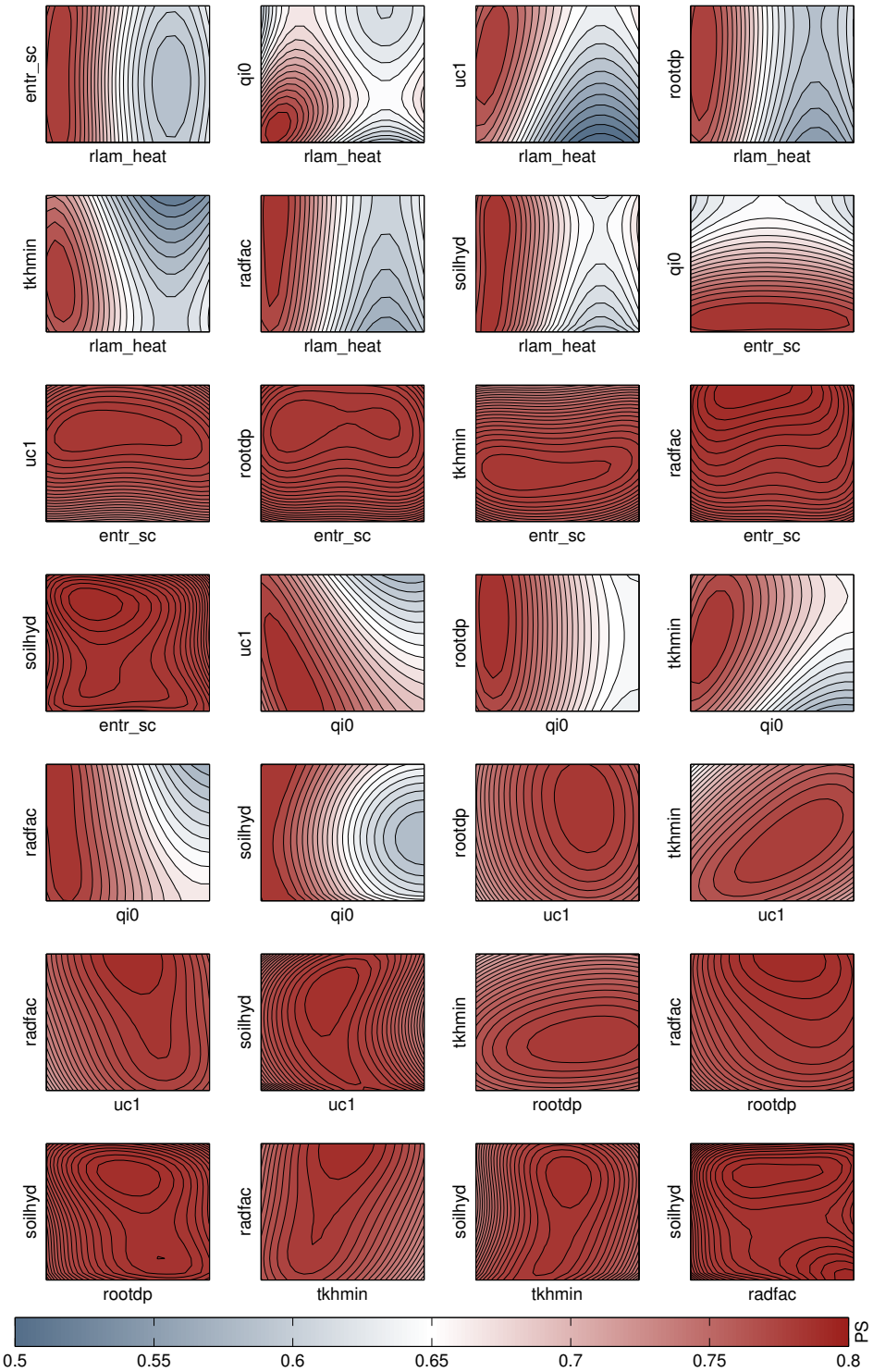

Figure 5.6: Contours of the model performance as predicted by the metamodel along planes of all possible parameter pairs.

Header:

```
% Plot performance planes for each parameter pair predicted
% by the metamodel
%
% NAME
%   planes
%
% PURPOSE
%
%   Predict performance surface for a parameter pairs between
%   the design points to visualize performance non-linearities.
%
% INPUTS
%
%   From the structure metamodel, parameters and datamatrix the following fields are
%   processed (mind the same naming in the input)
%
%   metamodel.a:
%
%          Vector of linear terms of the metamodel [...,N,1] additional
%          data dimensions possible (ex:a~[Regions,Variables,Time,N,1])
%
%   metamodel.B:
%
%           Matrix of quadratic and interactions terms [...,N,N] additional
%           data dimensions possible (ex:a~[Regions,Variables,Time,N,N])
%
%   parameters.range:
%
%            Range of values for each paramter to normalize the
%            scale.
%
%   parameters.default:
%
%            Default values of parameters to center the scale
%
%   datamatrix.reffdata:
%
%            Modeldata of when using default parameter values to
%            to center the datamatrix
%
%   pvector: Parameter values for one experiment with the
%            dimension of [N,1] N=Number parameters
% OUTUTS
%  pi:      Performance index for all data points
%  ps:      PS value for each simulation
% HISTORY
% First version: 11.10.2013
```

Bibliography

Annan, J. D., and J. C. Hargreaves. 2007. Efficient estimation and ensemble generation in climate modelling. *Philosophical Transactions of the Royal Society A-mathematical Physical and Engineering Sciences* **365**, 2077–2088.

Bellprat, O., S. Kotlarski, D. Luethi, and C. Schaer. 2012a. Exploring perturbed physics ensembles in a regional climate model. *Journal of Climate* **25**, 4582–4599.

Bellprat, O., S. Kotlarski, D. Lüthi, and C. Schär. 2012b. Objective calibration of regional climate models. *Journal of Geophysical Research*. submitted.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional.

McKay, M. D., R. J. Beckman, and W. J. Conover. 2000. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics* **42**, 55–61.

Neelin, J. D., A. Bracco, H. Luo, J. C. McWilliams, and J. E. Meyerson. 2010. Considerations for parameter optimization and sensitivity in climate models. *Proceedings of the National Academy of Sciences of the United States of America* **107**, 21349–21354.

Rougier, J., D. M. H. Sexton, J. M. Murphy, and D. Stainforth. 2009. Analyzing the Climate Sensitivity of the HadSM3 Climate Model Using Ensembles from Different but Related Experiments. *Journal of Climate* **22**, 3540–3557.

Urban, N. M., and T. E. Fricker. 2010. A comparison of Latin hypercube and grid ensemble designs for the multivariate emulation of an Earth system model. *Computers & Geosciences* **36**, 746–755.

Williamson, D., M. Goldstein, L. Allison, A. Blaker, P. Challenor, L. Jackson, and K. Yamazaki. 2013. History matching for exploring and reducing climate model parameter space using observations and a large perturbed physics ensemble. *Climate Dynamics* **41**, 1703–1729.