Using Emulators to Estimate Uncertainty in Complex Models

Peter Challenor and the MUCM Team

National Oceanography Centre

NATURAL ENVIRONMENT RESEARCH COUNCIL

MUCM

Managing Uncertainty in Complex Models

The CENTURY model

- Consider CENTURY, a model of soil carbon processes
- Initial conditions: 8 carbon pools
- Other contextual data: 3 soil texture inputs
  - Sand, clay, silt
- Exogenous data: 3 climate inputs for each monthly time step
- Parameters: coded in differential equations

Input uncertainty

Input uncertainty

- We are typically uncertain about the values of many of the inputs
  - Measurement error, lack of knowledge
  - E.g. CENTURY
    - Texture, initial carbon pools, (future) climate

Input uncertainty

- We are typically uncertain about the values of many of the inputs
  - Measurement error, lack of knowledge
  - E.g. CENTURY
    - Texture, initial carbon pools, (future) climate
- Input uncertainty should be expressed as a probability distribution
  - Across all uncertain inputs
  - Model users are often reluctant to specify more than plausible bounds
    - Inadequate to characterise output uncertainty

Output uncertainty

Output uncertainty

- Input uncertainty induces uncertainty in the output y

Output uncertainty

- Input uncertainty induces uncertainty in the output y
- It also has a probability distribution
- In theory, this is completely determined by
  - the probability distribution on x
  - and the model f
- In practice, finding this distribution and its properties is not straightforward

A trivial model

A trivial model

- Suppose we have just two inputs and a simple linear model

$$y = x_1 + 3x_2$$

A trivial model

- Suppose we have just two inputs and a simple linear model

$$y = x_1 + 3x_2$$

- Suppose that x_1 and x_2 have independent uniform distributions over [0, 1]
  - i.e. they define a point that is equally likely to be anywhere in the unit square

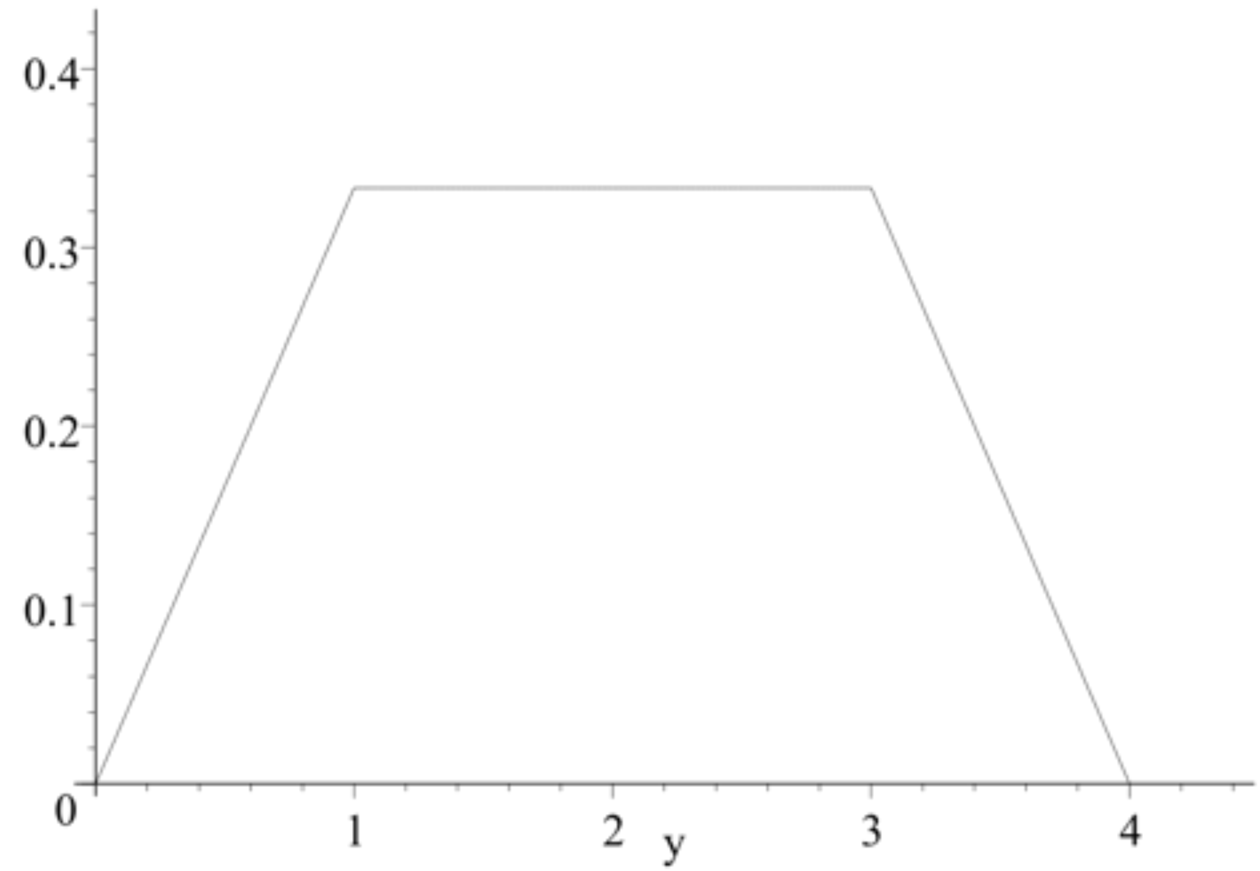
A trivial model

- Suppose we have just two inputs and a simple linear model

  $$y = x_1 + 3x_2$$

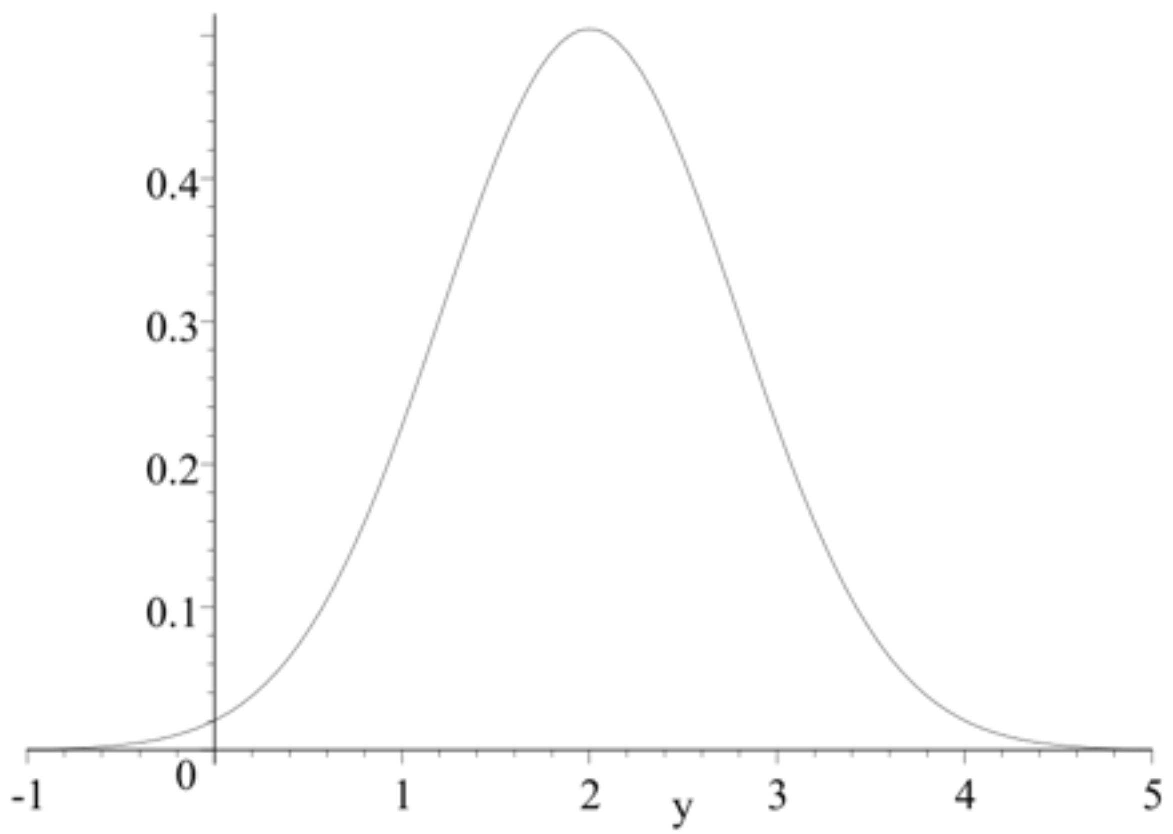
- Suppose that x_1 and x_2 have independent uniform distributions over [0, 1]
  - i.e. they define a point that is equally likely to be anywhere in the unit square
- Then we can determine the distribution of y exactly

Trivial model – output distribution



- The distribution of y has this trapezium form

Trivial model – normal inputs



- If x_1 and x_2 have normal distributions $N(0.5, 0.25^2)$ we get a normal output

A slightly less trivial model

A slightly less trivial model

- Now consider the simple nonlinear model

  $$y = sin(x_1)/\{1+exp(x_1+x_2)\}$$

- We still have only 2 inputs and quite a simple equation

A slightly less trivial model

- Now consider the simple nonlinear model

  $$y = sin(x_1)/\{1+exp(x_1+x_2)\}$$

- We still have only 2 inputs and quite a simple equation
- But even for nice input distributions we cannot get the output distribution exactly

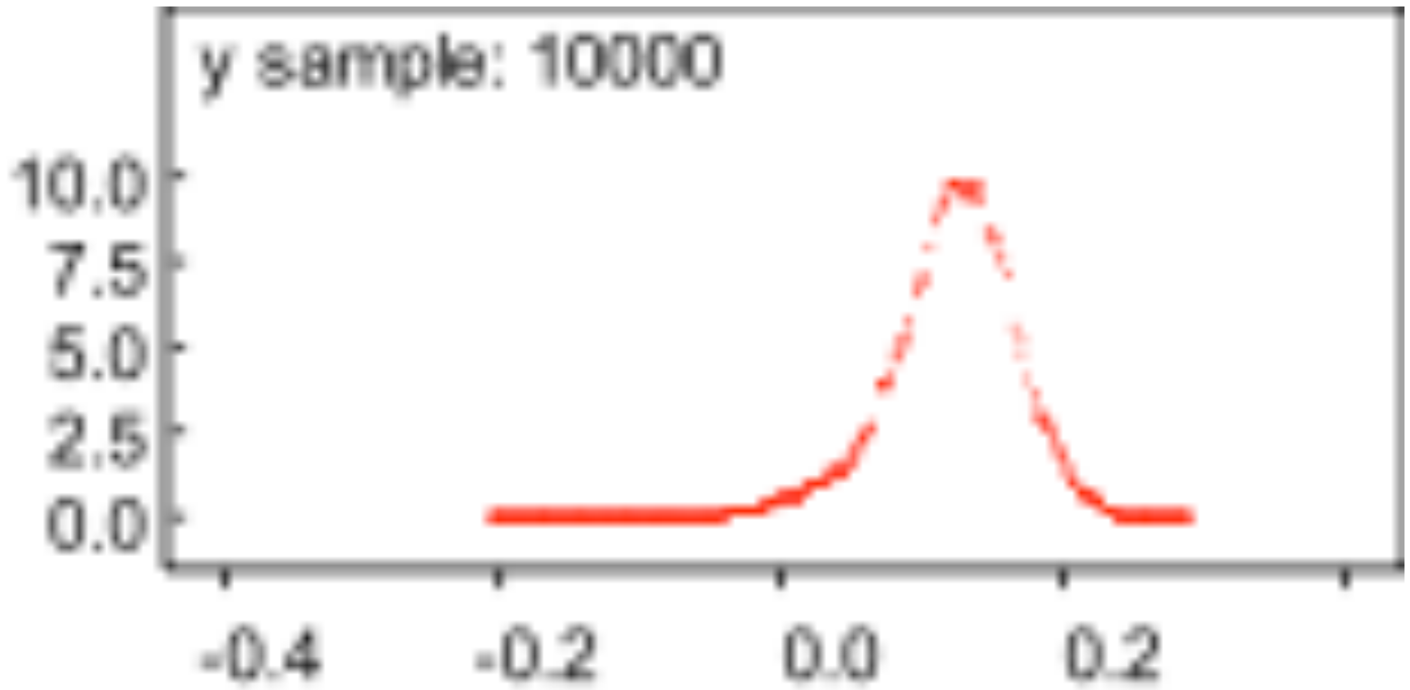
A slightly less trivial model

- Now consider the simple nonlinear model

$$y = sin(x_1)/\{1+exp(x_1+x_2)\}$$

- We still have only 2 inputs and quite a simple equation
- But even for nice input distributions we cannot get the output distribution exactly
- The simplest way to compute it would be by Monte Carlo

Monte Carlo output distribution



- This is for the normal inputs
- 10,000 random normal pairs were generated and y calculated for each pair

Uncertainty analysis (UA)

Uncertainty analysis (UA)

- The process of characterising the distribution of the output y is called **uncertainty analysis**

Uncertainty analysis (UA)

- The process of characterising the distribution of the output y is called **uncertainty analysis**
- Plotting the distribution is a good graphical way to characterise it

Uncertainty analysis (UA)

- The process of characterising the distribution of the output y is called **uncertainty analysis**
- Plotting the distribution is a good graphical way to characterise it
- Quantitative summaries are often more important
  - Mean, median
  - Standard deviation, quartiles
  - Probability intervals

UA versus plug-in

UA versus plug-in

- Even if we just want to estimate y, UA does better than the "plug-in" approach of running the model for estimated values of x
  - For the simple nonlinear model, the central estimates of x_1 and x_2 are 0.5, but

    $sin(0.5)/(1+exp(1))$ = 0.129

    is a slightly too high estimate of y compared with the mean of 0.117 or median of 0.122

UA versus plug-in

- Even if we just want to estimate y, UA does better than the "plug-in" approach of running the model for estimated values of x
  - For the simple nonlinear model, the central estimates of x_1 and x_2 are 0.5, but

    $sin(0.5)/(1+exp(1))$ = 0.129

    is a slightly too high estimate of y compared with the mean of 0.117 or median of 0.122
- The difference can be much more marked for highly nonlinear models
  - As is often the case with serious simulators

Example: UK carbon flux in 2000

Example: UK carbon flux in 2000

- Vegetation model predicts carbon exchange from each of 700 pixels over England & Wales in 2000
  - Principal output is Net Biosphere Production

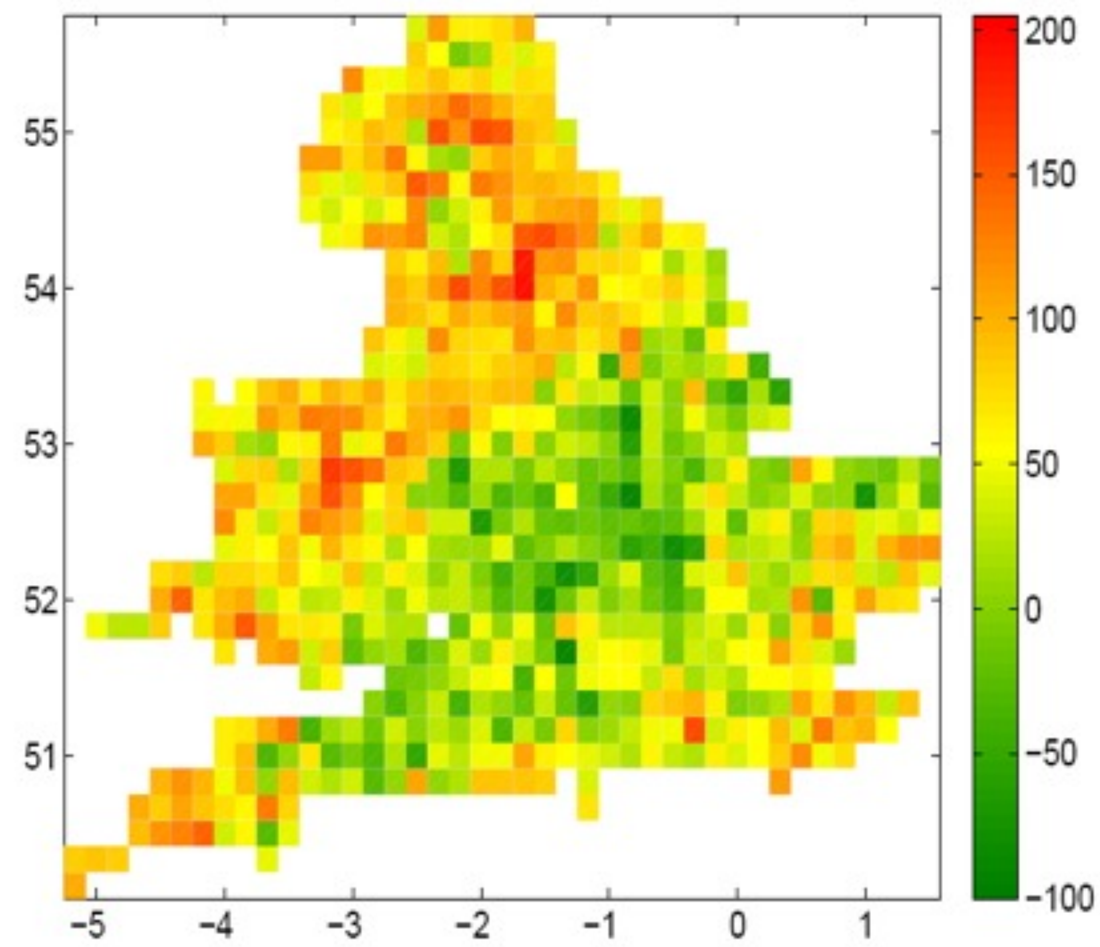
Example: UK carbon flux in 2000

- Vegetation model predicts carbon exchange from each of 700 pixels over England & Wales in 2000
  - Principal output is Net Biosphere Production
- Accounting for uncertainty in inputs
  - Soil properties
  - Properties of different types of vegetation
  - Land usage
  - (Not structural uncertainty)

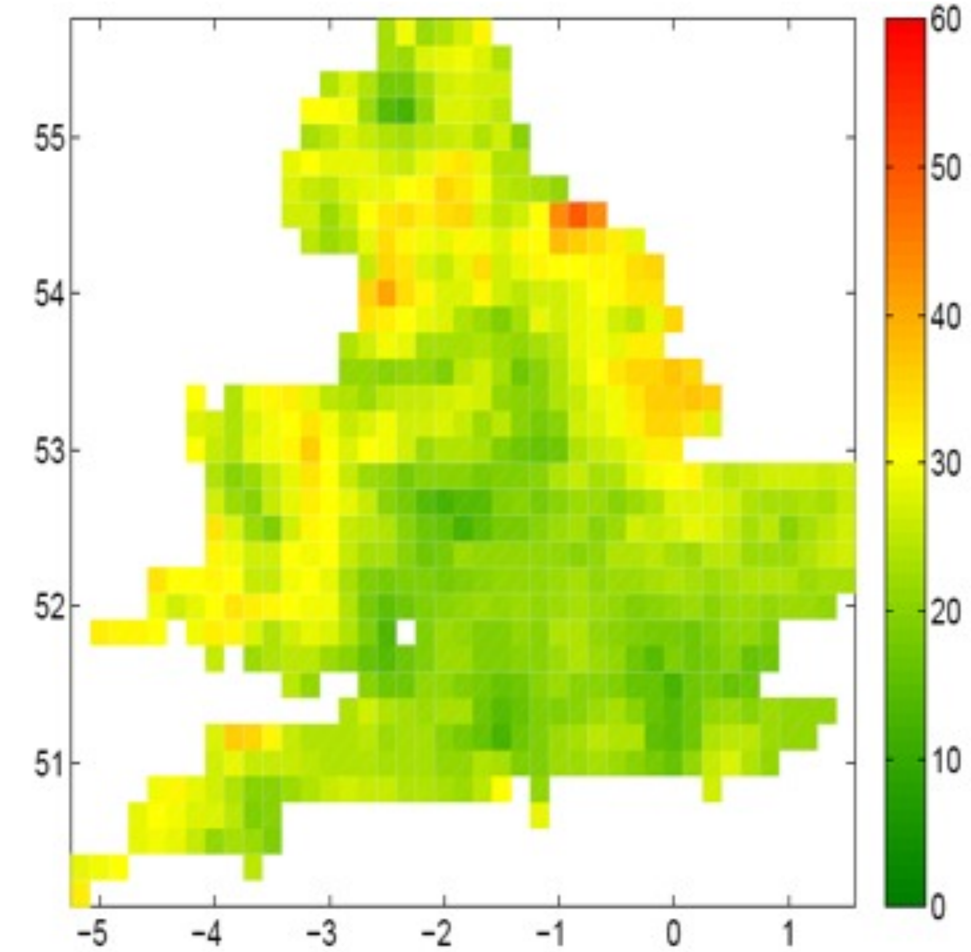
Example: UK carbon flux in 2000

- Vegetation model predicts carbon exchange from each of 700 pixels over England & Wales in 2000
  - Principal output is Net Biosphere Production
- Accounting for uncertainty in inputs
  - Soil properties
  - Properties of different types of vegetation
  - Land usage
  - (Not structural uncertainty)
- Aggregated to England & Wales total
  - Allowing for correlations
  - Estimate 7.46 Mt C
  - Std deviation 0.54 Mt C

Maps



Mean NBP



Standard deviation

England & Wales aggregate

PFT	Plug-in estimate (MtC)	Mean (MtC)	Variance (MtC^2)
Grass	5.28	4.37	0.2453
Crop	0.85	0.43	0.0327
Deciduous	2.13	1.80	0.0221
Evergreen	0.80	0.86	0.0048
Covariances			-0.0081
Total	9.06	7.46	0.2968

Reducing uncertainty

Reducing uncertainty

- To reduce uncertainty, get more information!

Reducing uncertainty

- To reduce uncertainty, get more information!
- Informal – more/better science
  - Tighten $p(x)$ through improved understanding
  - Tighten $p(z-y)$ through improved modelling or programming

Reducing uncertainty

- To reduce uncertainty, get more information!
- Informal – more/better science
  - Tighten $p(x)$ through improved understanding
  - Tighten $p(z-y)$ through improved modelling or programming
- Formal – using real-world data
  - Calibration – learn about model parameters
  - Data assimilation – learn about the state variables
  - Learn about structural error $z-y$
  - Validation

So far, so good, but

- In principle, all this is straightforward
- In practice, there are many technical difficulties
  - Formulating uncertainty on inputs
    - Elicitation of expert judgements
  - Propagating input uncertainty
  - Modelling structural error
  - Anything involving observational data!
    - The last two are intricately linked
  - And *computation*

The problem of big models

The problem of big models

- Tasks like uncertainty propagation and calibration require us to run the simulator many times

The problem of big models

- Tasks like uncertainty propagation and calibration require us to run the simulator many times
- Uncertainty propagation
  - Implicitly, we need to run $f(x)$ at all possible x
  - Monte Carlo works by taking a sample of x from $p(x)$
  - Typically needs thousands of simulator runs

The problem of big models

- Tasks like uncertainty propagation and calibration require us to run the simulator many times
- Uncertainty propagation
  - Implicitly, we need to run $f(x)$ at all possible x
  - Monte Carlo works by taking a sample of x from $p(x)$
  - Typically needs thousands of simulator runs
- Calibration
  - Traditionally done by searching x space for good fits to the data

The problem of big models

- Tasks like uncertainty propagation and calibration require us to run the simulator many times
- Uncertainty propagation
  - Implicitly, we need to run $f(x)$ at all possible x
  - Monte Carlo works by taking a sample of x from $p(x)$
  - Typically needs thousands of simulator runs
- Calibration
  - Traditionally done by searching x space for good fits to the data
- Both become impractical if the simulator takes more than a few seconds to run
  - 10,000 runs at 1 minute each takes a week of computer time
  - We need a more efficient technique

Gaussian process representation

Gaussian process representation

- More efficient approach
  - First work in early 1980s (DACE)

Gaussian process representation

- More efficient approach
  - First work in early 1980s (DACE)
- Represent the code as an unknown function
  - $f(.)$ becomes a random process
  - We generally represent it as a Gaussian process (GP)
    - Or its second-order moment version (so called Bayes Linear)

Gaussian process representation

- More efficient approach
  - First work in early 1980s (DACE)
- Represent the code as an unknown function
  - $f(.)$ becomes a random process
  - We generally represent it as a Gaussian process (GP)
    - Or its second-order moment version (so called Bayes Linear)
- Training runs
  - Run simulator for sample of x values
  - Condition GP on observed data
  - Typically requires many fewer runs than Monte Carlo
    - And the x values don't need to be chosen randomly

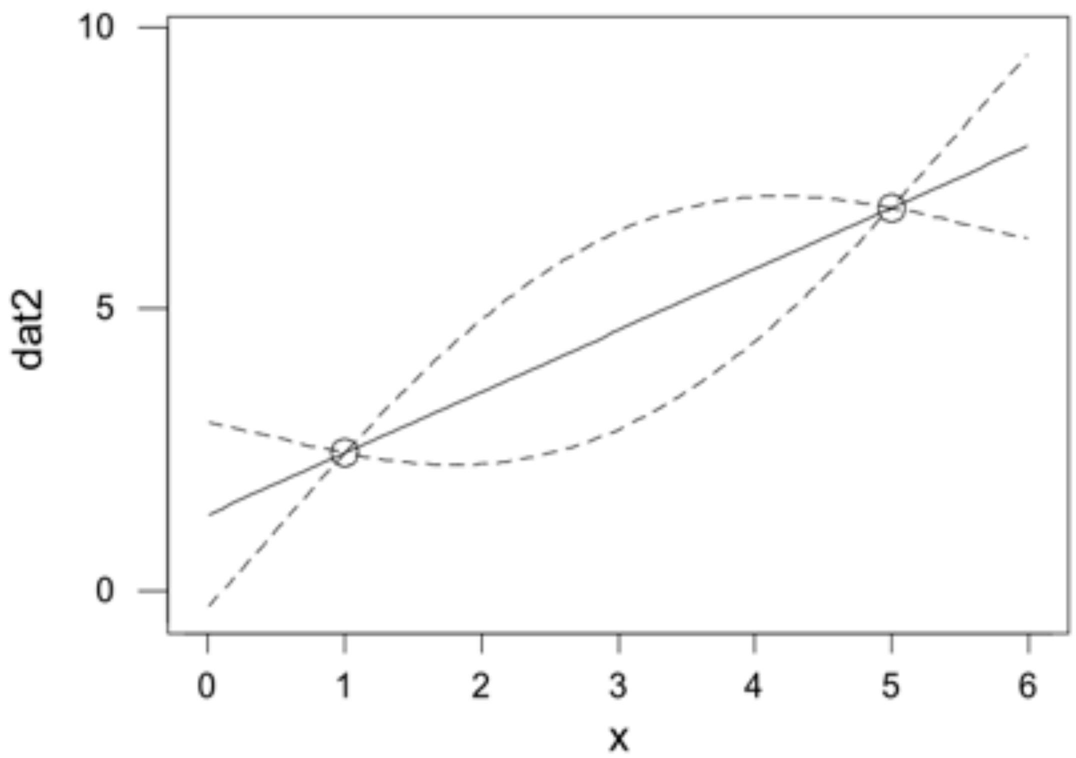
Emulation

- Analysis is completed by prior distributions for, and posterior estimation of, hyperparameters

- The posterior distribution is known as an **emulator** of the computer simulator
  - Posterior mean estimates what the simulator would produce for any untried x (prediction)
  - With uncertainty about that prediction given by posterior variance
  - Correctly reproduces training data

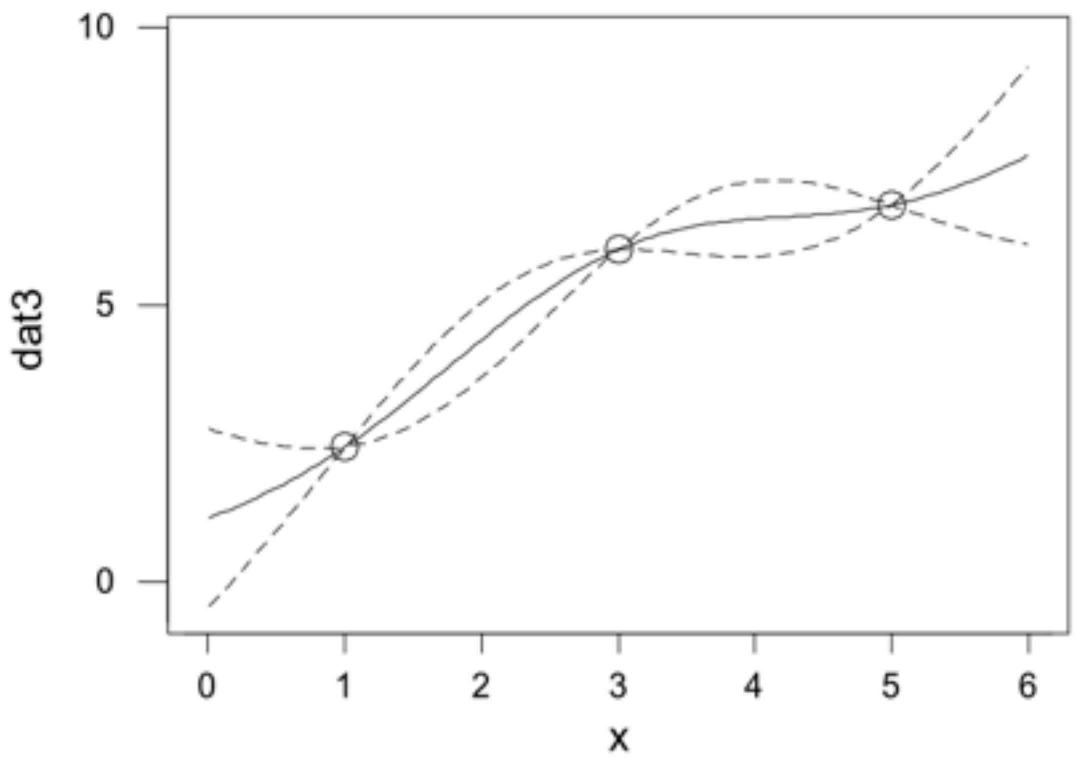
2 code runs

- Consider one input and one output
- Emulator estimate interpolates data
- Emulator uncertainty grows between data points



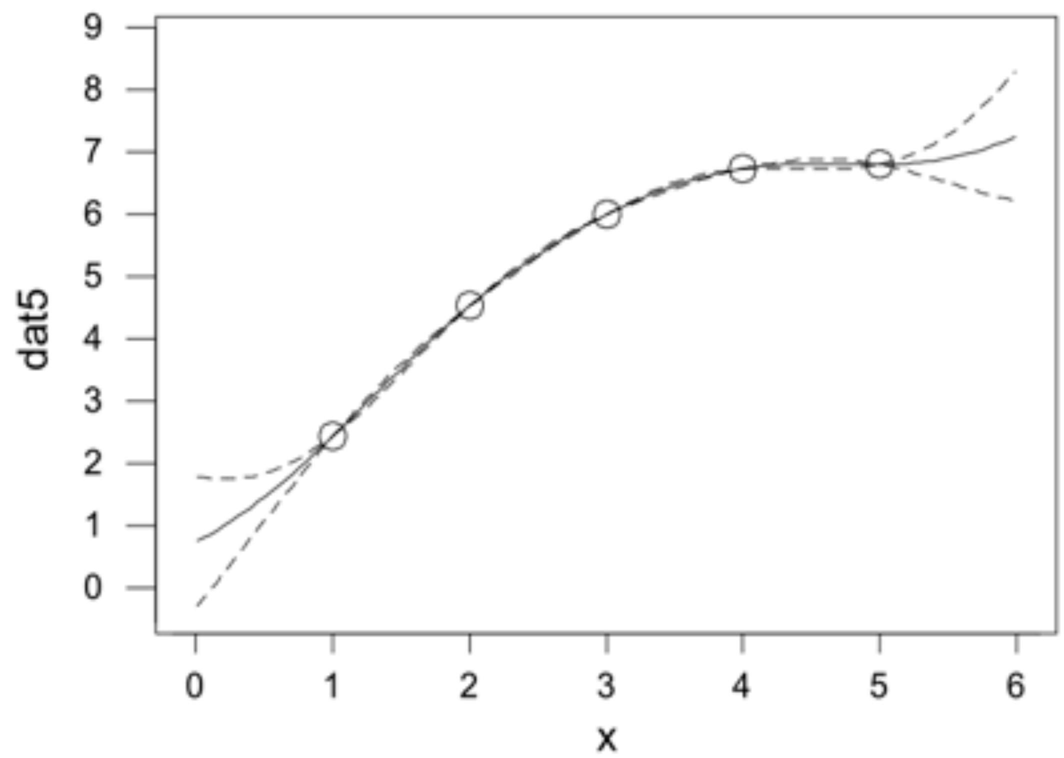
3 code runs

- Adding another point changes estimate and reduces uncertainty



5 code runs

- And so on



Then what?

Then what?

- Given enough training data points we can in principle emulate any simulator output accurately
  - So that posterior variance is small "everywhere"
  - Typically, this can be done with orders of magnitude fewer model runs than traditional methods
    - At least in relatively low-dimensional problems

Then what?

- Given enough training data points we can in principle emulate any simulator output accurately
  - So that posterior variance is small "everywhere"
  - Typically, this can be done with orders of magnitude fewer model runs than traditional methods
    - At least in relatively low-dimensional problems
- Use the emulator to make inference about other things of interest
  - E.g. uncertainty analysis, calibration

Then what?

- Given enough training data points we can in principle emulate any simulator output accurately
  - So that posterior variance is small "everywhere"
  - Typically, this can be done with orders of magnitude fewer model runs than traditional methods
    - At least in relatively low-dimensional problems
- Use the emulator to make inference about other things of interest
  - E.g. uncertainty analysis, calibration
- Conceptually very straightforward in the Bayesian framework
  - But of course can be computationally hard

MUCM

MUCM

- Managing Uncertainty in Complex Models
  - Universities of Sheffield, Aston, Durham, LSE, NOC
  - Large 4-year research grant
  - June 2006 to September 2010
  - 7 postdoctoral research associates
  - 4 project PhD students
  - Objective to develop BACCO methods into a basic technology, usable and widely applicable

MUCM

- Managing Uncertainty in Complex Models
  - Universities of Sheffield, Aston, Durham, LSE, NOC
  - Large 4-year research grant
  - June 2006 to September 2010
  - 7 postdoctoral research associates
  - 4 project PhD students
  - Objective to develop BACCO methods into a basic technology, usable and widely applicable
- MUCM2: New directions for MUCM
  - Smaller 2-year grant to September 2012
  - Scoping and developing research proposals

Primary MUCM deliverables

Primary MUCM deliverables

- Methodology and papers moving the technology forward
  - Papers both in statistics and application area journals

Primary MUCM deliverables

- Methodology and papers moving the technology forward
  - Papers both in statistics and application area journals
- The MUCM toolkit
  - Documentation of the methods and how to use them
  - With emphasis on what is found to work reliably across a range of modelling areas
  - Web-based

Primary MUCM deliverables

- Methodology and papers moving the technology forward
  - Papers both in statistics and application area journals
- The MUCM toolkit
  - Documentation of the methods and how to use them
  - With emphasis on what is found to work reliably across a range of modelling areas
  - Web-based
- Case studies
  - Three substantial case studies
  - Showcasing methods and best practice
  - Linked to toolkit

Primary MUCM deliverables

- Methodology and papers moving the technology forward
  - Papers both in statistics and application area journals
- The MUCM toolkit
  - Documentation of the methods and how to use them
  - With emphasis on what is found to work reliably across a range of modelling areas
  - Web-based
- Case studies
  - Three substantial case studies
  - Showcasing methods and best practice
  - Linked to toolkit
- Events
  - Workshops – conceptual and hands-on
  - Short courses
  - Conferences – UCM 2010, UCM 2012

Focus on the toolkit

- The toolkit is a ‘recipe book’
  - The good sort that encourages you to experiment
  - There are recipes (procedures) but also lots of explanation of concepts and discussion of choices
- It is not a software package
  - Software packages are great if they are in your favourite language
    - But it probably wouldn’t be!
  - Packages are dangerous without basic understanding
- The purpose of the toolkit is to build that understanding
  - And it enables you to easily develop your own code
  - Over 300 pages

MUCM Toolkit

Managing Uncertainty in Complex Models

Toolkit Home
Tutorial
Toolkit Structure
Threads
Case Studies
Page List
Notation
Comments

The MUCM Toolkit, release 7

Welcome to the MUCM Toolkit. The toolkit is a resource for people interested in quantifying and managing uncertainty in the outputs of mathematical models of complex real-world processes. We refer to such a model as a simulation model or a simulator.

The toolkit is a large, interconnected set of webpages and one way to use it is just to browse more or less randomly through it. However, we have also provided some organised starting points and threads through the toolkit.

- We have an introductory tutorial on MUCM methods and uncertainty in simulator outputs here.
- You can read about the toolkit structure.
- The various threads, each of which sets out in a series of steps how to use the MUCM approach to build an emulator of a simulator and to use it to address some standard problems faced by modellers and users of simulation models. This release contains the following threads:
    - ThreadCoreGP, which deals with the simplest emulation scenario, called the core problem, using the Gaussian process approach;
    - ThreadCoreBL, which also deals with the core problem, but follows the Bayes linear approach. A simple guide to the differences between the two approaches can be found in the alternatives page on Gaussian process or Bayes Linear Emulator (AltGPorBLEmulator);
    - ThreadVariantMultipleOutputs, which extends the core problem to address the case where we wish to emulate two or more simulator outputs;
    - ThreadVariantDynamic, which extends the core analysis in a different direction, where we wish to emulate the time series output of a dynamic simulator;
    - ThreadVariantTwoLevelEmulation, which considers the situation where we have two simulators of the same real-world phenomenon, a slow but relatively accurate simulator whose output we wish to emulate, and a quick and relatively crude simulator. This thread discusses how to use many runs of the fast simulator to build an informative

prior model for the slow simulator, so that fewer training runs of the slow simulator are needed;
  - ThreadVariantWithDerivatives, which extends the core analysis for the case where we can obtain derivatives of the simulator output with respect to the various inputs, to use as training data;
  - ThreadVariantModelDiscrepancy, which deals with modelling the relationship between the simulator outputs and the real-world process being simulated. Recognising this model discrepancy is a crucial step in making useful predictions from simulators, in calibrating simulation models and handling multiple models.
  - ThreadGenericMultipleEmulators, which deals with combining two or more emulators to produce emulation of some combination of the respective simulator outputs;
  - ThreadGenericEmulateDerivatives, which shows how to use an emulator to predict the values of derivatives of the simulator output;
  - ThreadGenericHistoryMatching, which deals with iteratively narrowing down the region of possible input values for which the simulator would produce outputs acceptably close to observed data.
  - ThreadGenericCalibration, which addresses how to learn in a fully Bayesian way from observations of the real-world process.
  - ThreadTopicSensitivityAnalysis, which is a topic thread providing more detailed background on the topic of sensitivity analysis, and linking together the various procedures for such techniques in the other toolkit threads.
  - ThreadTopicScreening, which provides a broad view of the idea of screening the simulator inputs to reduce their dimensionality.
  - ThreadTopicExperimentalDesign, which gives a detailed overview of the methods of experimental design that are relevant to MUCM, particularly those relating to the design of a training sample.

- Another important feature of the toolkit is the MUCM Case Studies. The Case Studies are demonstrations of the MUCM methodology applied to address substantive challenges faced by users of real simulation models. The techniques that they use are all described in the toolkit and there are appropriate links from each Case Study to the relevant pages in the toolkit. The Case Studies generally are accessed from the page MetaCaseStudies, and from the menu bar.

Later releases of the toolkit will add more threads and other material, including more extensive examples to guide the toolkit user and further Case Studies. In each release we also add more detail to some of the existing threads. For instance, in this release we have a substantial reworking of the variant thread on model discrepancy, a new thread on calibration and two new Case Studies.

Last modified: 12 January 2011 18:55:29.

Toolkit Structure

- Built around a number of ‘threads’
- Two core threads (GP and BL)
- 5 variant threads
- 4 generic threads
- 3 topic threads
- Over 300 pages in all

GP and Bayes Linear

- Two fundamental approaches to building emulators
- GP - Gaussian process. Assumes a Gaussian distribution. Requires distributional priors but these can be improper
- BL – Bayes Linear. No distributional assumptions. Only first and second moments. Priors only in terms of moments. Fast.

The Core Problem

- This includes only one simulator
- The simulator has only one output
- The output is deterministic
- We do not have observations of the real world process against which to compare the simulator
- We do not wish to make statements about the real world process
- We cannot directly observe simulator derivatives

Steps in building an emulator

- Specify the Gaussian process (or BL)model
- Select the prior distributions for the GP hyperparameters
- Choose a design for training and validation
- Fit the emulator to the simulator runs
- Validate and re-fit if needed

MUCM Toolkit

Managing Uncertainty in Complex Models

Thread: Analysis of the core model using Gaussian Process methods

Overview

The principal user entry points to the MUCM toolkit are the various *threads*, as explained in the Toolkit Structure. The main threads give detailed instructions for building and using emulators in various contexts.

This thread takes the user through the analysis of the most basic kind of problem, using the fully Bayesian approach based on a Gaussian process (GP) emulator. We characterise a core problem or model as follows:

- We are only concerned with one simulator.
- The simulator only produces one output, or (more realistically) we are only interested in one output.
- The output is deterministic.
- We do not have observations of the real world process against which to compare the simulator.
- We do not wish to make statements about the real world process.
- We cannot directly observe derivatives of the simulator.

Each of these aspects of the core problem is discussed further in page DiscCore.

The fully Bayesian approach has a further restriction:

- We are prepared to represent the simulator as a Gaussian process.

See also the discussion page on the Gaussian assumption (DiscGaussianAssumption).

This thread comprises a number of key stages in developing and using the emulator.

Active inputs

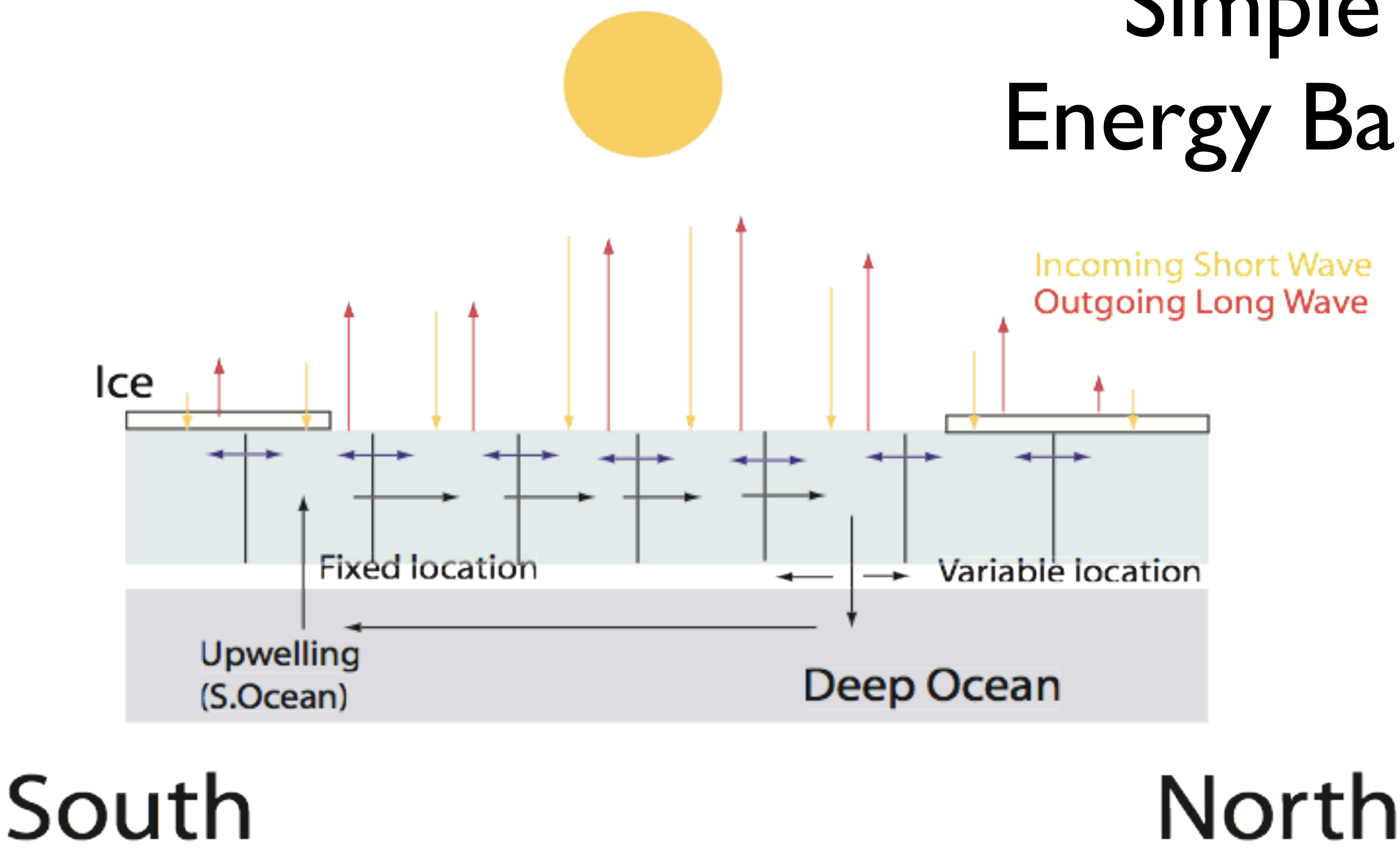
Before beginning to develop the emulator, it is necessary to decide what inputs to the simulator will be varied. Complex simulators often have many inputs and many outputs. In the core problem, only one output is of interest and we assume that this has already been identified. It may also be necessary to restrict the number of inputs that will be represented in the emulator. The distinction between active inputs and inactive inputs is considered in the discussion page DiscActiveInputs.

Once the active inputs have been determined, we will refer to these simply as the inputs, and we denote the number of (active) inputs by p.

The GP model

The first stage in building the emulator is to model the mean and covariance structures of the Gaussian process that is to represent the simulator. As

Simple Example: Energy Balance Model



Inputs: 18 initial surface temperatures +8 others

Outputs: 18 final temperatures + 4 others

We use the mean surface temperature as our output

Vary solar constant

Specifying the GP model

- The first step in building an emulator is the specification of the Gaussian process model
- This consists of specifying the mean

$$\mathrm{E}[f(x)] = h_\beta(x)$$

- ... alternatives for the mean function are given in page AltMeanFunction
- and the specification of the correlation function

$$\mathrm{Cov}[f(x), f(x')] = \sigma^2 c_\delta(x, x')$$

- ... alternatives for the correlation function are given in page AltCorrelationFunction

MUCM Toolkit

Managing Uncertainty in Complex Models

Alternatives: Emulator prior mean function

Overview

The process of building an emulator of a simulator involves first specifying prior beliefs about the simulator and then updating this using a training sample of simulator runs. Prior specification may be either using the fully Bayesian approach in the form of a Gaussian process or using the Bayes linear approach in the form of first and second order moments. The basics of building an emulator using these two approaches are set out in the two core threads: the thread for the analysis of core model using Gaussian process methods (ThreadCoreGP) and the thread for the Bayes linear emulation for the core model (ThreadCoreBL).

In either approach it is necessary to specify a mean function and covariance function. We consider here the various alternative forms of mean function that are dealt with in the MUCM toolkit. An extension to the case of a vector mean function as required by the thread for the analysis of a simulator with multiple outputs using Gaussian methods (ThreadVariantMultipleOutputs) can be found in a companion page to this one, dealing with alternatives for multi-output mean functions (AltMeanFunctionMultivariate).

Choosing the Alternatives

The mean function gives the prior expectation for the simulator output at any given set of input values. We assume here that only one output is of interest, as in the core problem.

In general, the mean function will be specified in a form that depends on a number of hyperparameters. Thus, if the vector of hyperparameters for the mean function is β then we denote the mean function by $m(\cdot)$, so that $m(x)$ is the prior expectation of the simulator output for vector x of input values.

In principle, this should entail the analyst thinking about what simulator output would be expected for every separate possible input vector x. In practice, of course, this is not possible. Instead, $m(\cdot)$ represents the general shape of how the analyst expects the simulator output to respond to changes in the inputs. The use of the unknown hyperparameters allows the emulator to learn their values from the training sample data. So the key task in specifying the mean function is to think generally about how the output will respond to the inputs.

Having specified $m(\cdot)$, the subsequent steps involved in building and using the emulator are described in ThreadCoreGP / ThreadCoreBL.

The Nature of the Alternatives

The linear form

It is usual, and convenient in terms of subsequent building and use of the emulator, to specify a mean function of the form

$$m(x) = \beta^T h(x)$$

MUCM Toolkit

Managing Uncertainty
in Complex Models

Toolkit Ho
Tutorial
Toolkit Str
Threads
Case Studi
Page List
Notation
Comments

MUCM Toolkit

Managing Uncertainty
in Complex Models

Alternatives: Emulator prior correlation function

Overview

The process of building an emulator of a simulator involves first specifying prior beliefs about the simulator and then updating this using a training sample of simulator runs. Prior specification may be either using the fully Bayesian approach in the form of a Gaussian process (GP) or using the Bayes linear approach in the form of first and second order moments. The basics of building an emulator using these two approaches are set out in the two core threads: the thread for the analysis of the core model using Gaussian process methods (ThreadCoreGP) and the thread for the Bayes linear emulation for the core model (ThreadCoreBL).

In either approach it is necessary to specify a covariance function. The formulation of a covariance function is considered in the discussion page on the GP covariance function (DiscCovarianceFunction). Within the MUCM toolkit, the covariance function is generally assumed to have the form of a variance (or covariance matrix) multiplied by a correlation function. We present here some alternative forms for the correlation function $c(\cdot,\cdot)$, dependent on hyperparameters δ.

Choosing the Alternatives

The correlation function $c(x,x')$ expresses the correlation between the simulator outputs at input configurations x and x', and represents the extent to which we believe the outputs at those two points should be similar. In practice, it is formulated to express the idea that we believe the simulator output to be a relatively smooth and continuous function of its inputs. Formally, this means the correlation will be high between points that are close together in the input space, but low between points that are far apart. The various correlation functions that we will consider in this discussion of alternatives all have this property.

The Nature of the Alternatives

The Gaussian form

It is common, and convenient in terms of subsequent building and use of the emulator, to specify a covariance function of the form

$$c(x,x') = \exp\{-(x-x')^T C(x-x')\}, \quad (1)$$

where C is a diagonal matrix whose diagonal elements are the inverse squares of the elements of the δ vector. Hence, if there are p inputs and the i-th elements of the input vectors x and x' and the correlation length vector δ are respectively x_i, x_i' and δ_i, we can write

$$c(x,x') = \exp\left[-\sum_{i=1}^{p}\{(x_i - x_i')/\delta_i\}^2\right] = \prod_{i=1}^{p}\exp\left[-\{(x_i - x_i')/\delta_i\}^2\right]. \quad (2)$$

This formula shows the role of the correlation length hyperparameter δ_i. The smaller its value, the closer together x_i and x_i' must be in order for the outputs at x and x' to be highly correlated. Large/small values of δ_i therefore mean that the output values are correlated over a wide/narrow range of the i-th input

Example: The GP model

- Mean function

$$\mathrm{E}[f(x)] = h(x)\beta = [1, x]\beta$$

- Correlation function

$$\mathrm{Cov}[f(x)] = \sigma^2 c(x, x')$$

$$c(x, x') = \exp\left\{-\frac{|x - x'|^2}{\delta^2}\right\}$$

Priors

- The GP model depends on a number of hyperparameters, which typically are:
- β : mean function hyperparameters
- σ^2 : GP variance
- δ : Correlation lengths
- A fully Bayesian approach requires the specification of prior distributions for each of these hyperparameters (BL only needs moments)

Design

- We need to design a set of simulator runs to *span* input space
- Common designs are optimised Latin Hypercubes and quasi-Monte Carlo sequences (AltCoreDesign or ThreadTopicExperimentalDesign)
- For our example these reduce to equally spaced points

MUCM Toolkit

Managing Uncertainty in Complex Models

Procedure: Build Gaussian process emulator for the core problem

Description and Background

The preparation for building a Gaussian process (GP) emulator for the core problem involves defining the prior mean and covariance functions, identifying prior distributions for hyperparameters, creating a design for the training sample, then running the simulator at the input configurations specified in the design. All of this is described in the thread for the analysis of the core model using Gaussian process methods (ThreadCoreGP). The procedure here is for taking those various ingredients and creating the GP emulator.

Inputs

- GP prior mean function $m(\cdot)$ depending on hyperparameters β
- GP prior correlation function $c(\cdot,\cdot)$ depending on hyperparameters δ
- Prior distribution $\pi(\cdot,\cdot,\cdot)$ for β,σ^2 and δ, where σ^2 is the process variance hyperparameter
- Design D comprising points $\{x_1,x_2,\ldots,x_n\}$ in the input space
- Output vector $f(D)=(f(x_1),f(x_2),\ldots,f(x_n))^T$, where $f(x_j)$ is the simulator output from input point x_j

Outputs

A GP-based emulator in one of the forms presented in the discussion page on GP emulator forms (DiscGPBasedEmulator).

In the case of general prior mean and correlation functions and general prior distribution:

- A GP posterior conditional distribution with mean function $m^*(\cdot)$ and covariance function $v^*(\cdot,\cdot)$ conditional on $\theta=\{\beta,\sigma^2,\delta\}$
- A posterior representation for θ

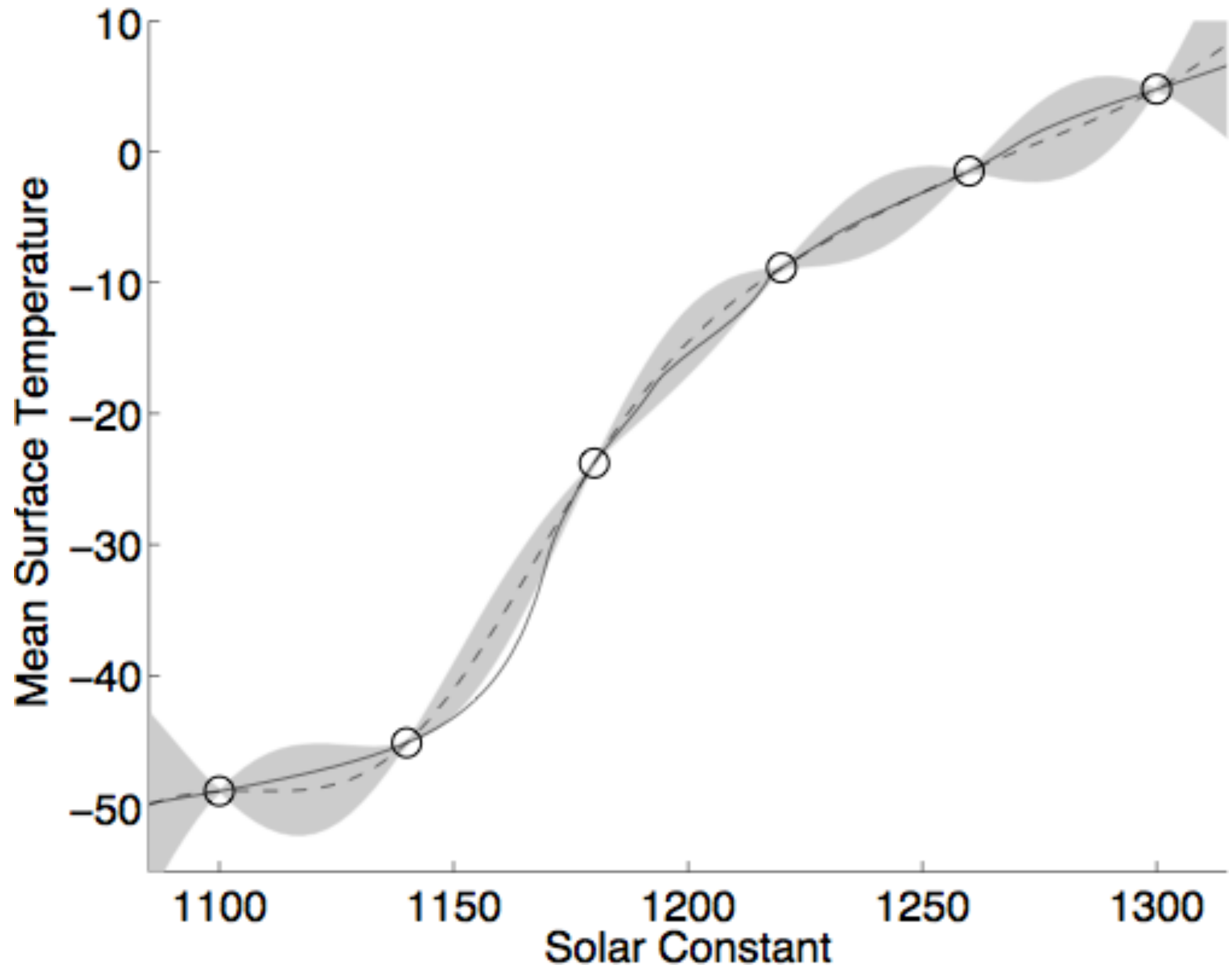
In the case of linear mean function, general correlation function, weak prior information on β,σ^2 and general prior distribution for δ:

- A t process posterior conditional distribution with mean function $m^*(\cdot)$, covariance function $v^*(\cdot,\cdot)$ and degrees of freedom b^* conditional on δ
- A posterior representation for δ

As explained in DiscGPBasedEmulator, the "posterior representation" for the hyperparameters is formally the posterior distribution for those hyperparameters, but for computational purposes this distribution is represented by a sample of hyperparameter values. In either case, the outputs define the emulator and allow all necessary computations for tasks such as prediction of the simulator output, uncertainty analysis or sensitivity analysis.

Procedure

Example



10
0
−10
−20
−30
−40
−50
Mean Surface Temperature
1100
1150
1200
1250
1300
Solar Constant

Validation

- Produce new set of simulator runs
- Compare true values with the emulator
- Generalisation of regression diagnostics (Bastos and O'Hagan, 2009)
- Over confident and under confident emulators

MUCM Toolkit

Managing Uncertainty in Complex Models

Procedure: Validate a Gaussian process emulator

Description and Background

Once an emulator has been built, under the fully Bayesian Gaussian process approach, using the procedure in page ProcBuildCoreGP, it is important to validate it. Validation involves checking whether the predictions that the emulator makes about the simulator output accord with actual observation of runs of the simulator. Since the emulator has been built using a training sample of runs, it will inevitably predict those correctly. Hence validation uses an additional set of runs, the validation sample.

We describe here the process of setting up a validation sample, using the validation data to test the emulator and interpreting the results of the tests.

We consider here an emulator for the core problem, and in particular we are only concerned with one simulator output.

Inputs

- Emulator, as derived in page ProcBuildCoreGP.
- The input configurations $D = \{x_1, x_2, \ldots, x_n\}$ at which the simulator was run to produce the training data from which the emulator was built.

Outputs

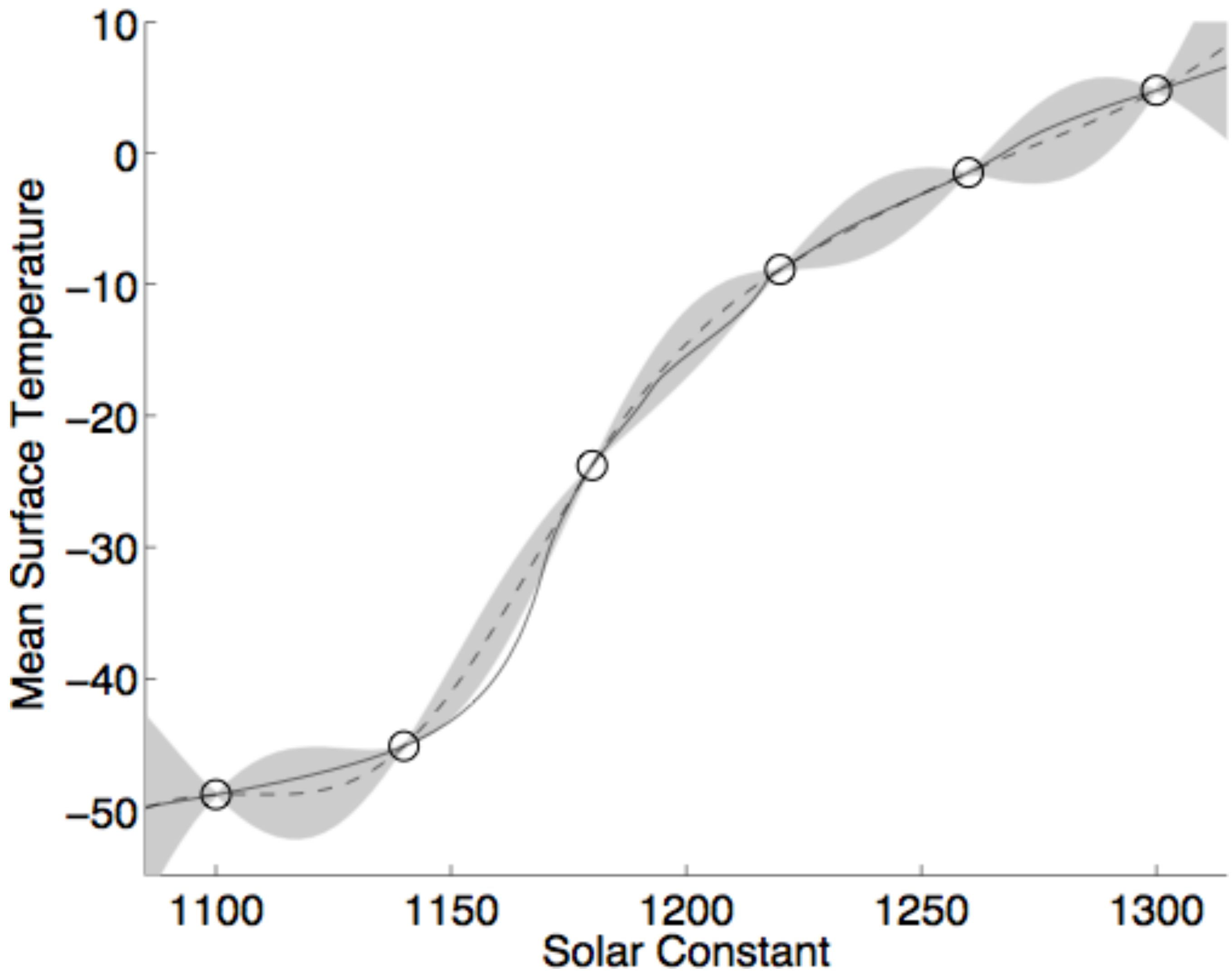
- A conclusion, either that the emulator is valid or that it is not valid.
- If the emulator is deemed not valid, then indications for how to improve it.

Procedure

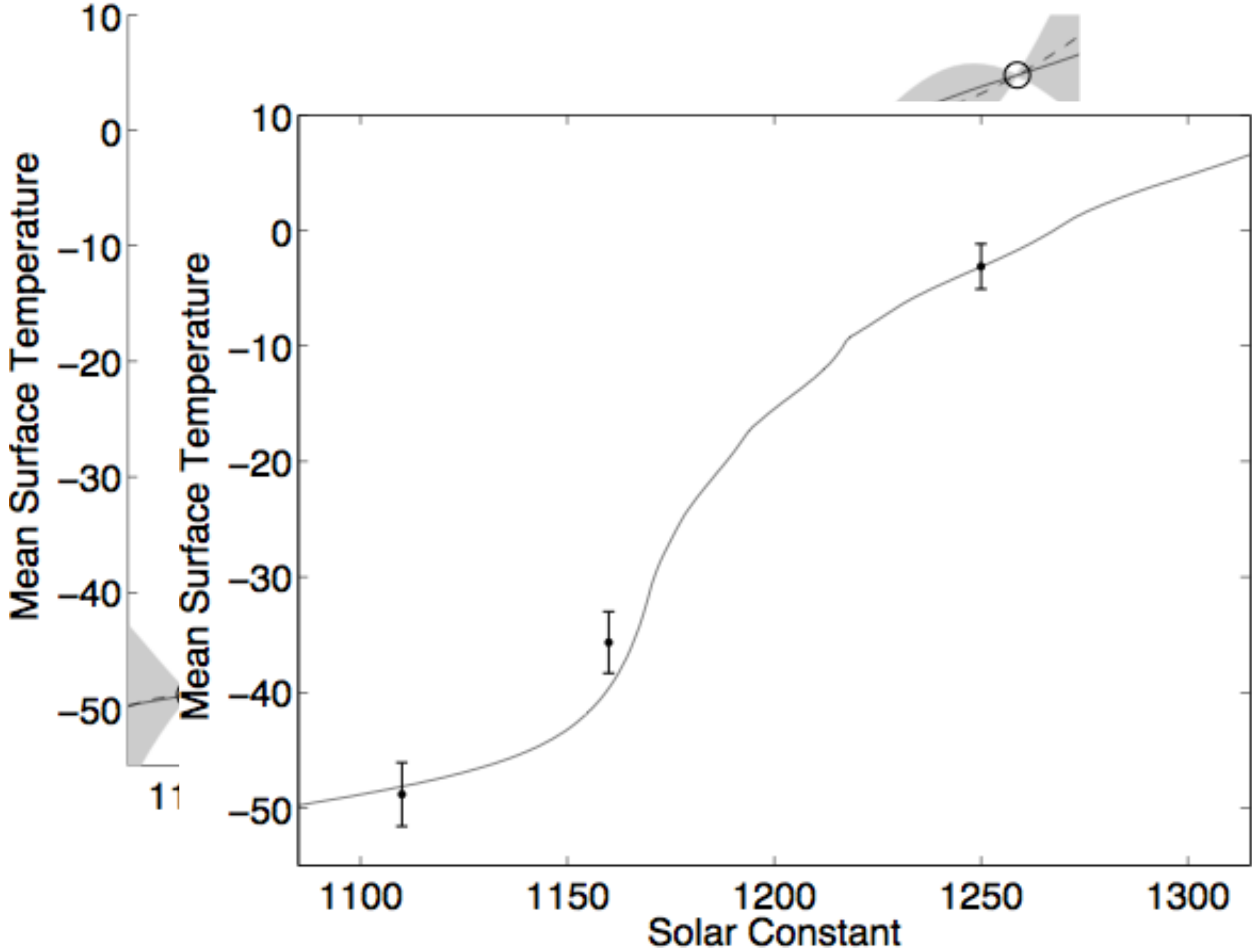
The validation sample

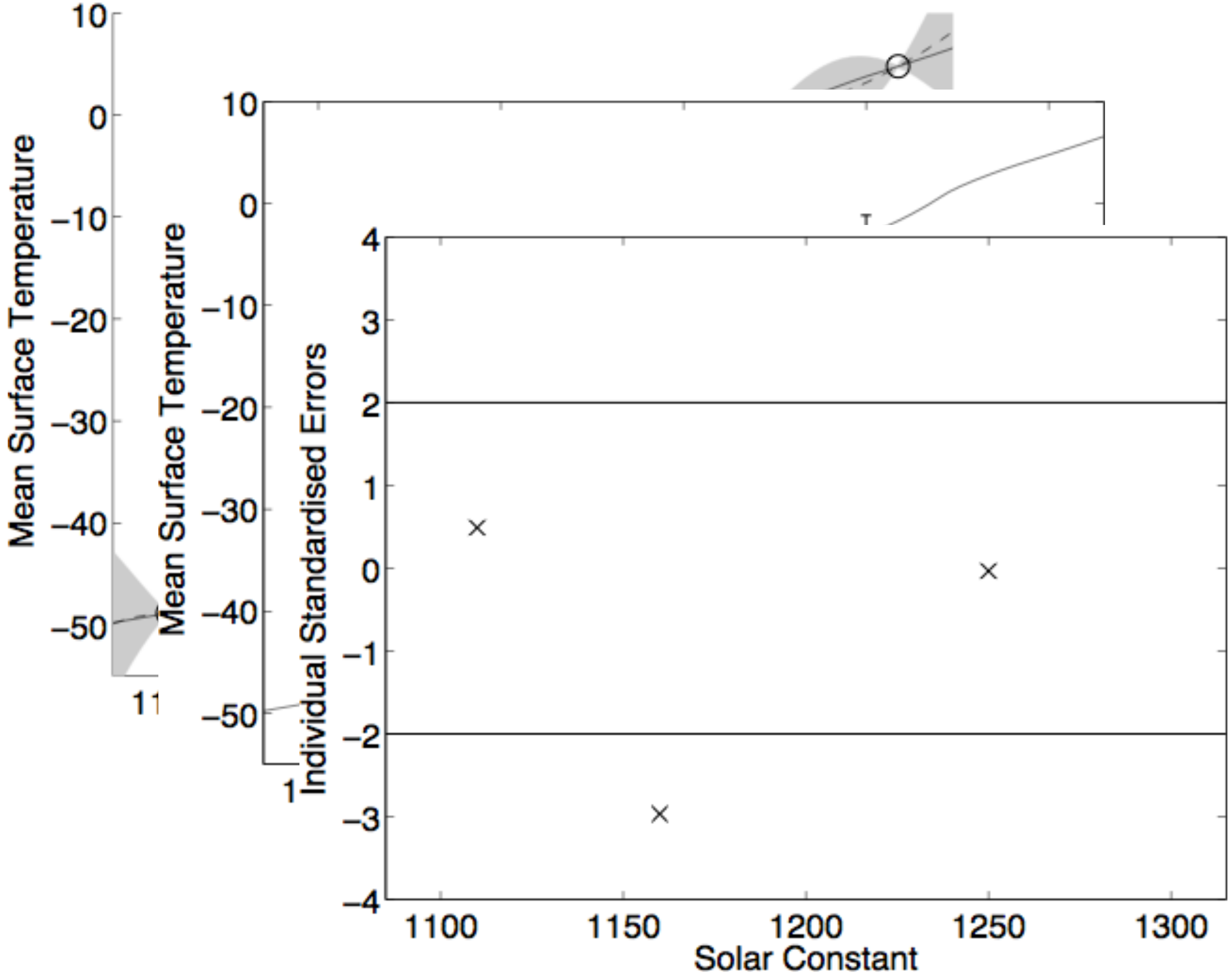
The validation sample must be distinct from the training sample that was used to build the emulator. One approach is to reserve part of the training data for validation, and to build the emulator only using the rest of the training data. However, the usual approach to designing a training sample (typically to use points that are well spread out, through some kind of space-filling design, see the alternatives page on training sample design (AltCoreDesign)) does not generally provide subsets that are good for validation. It is preferable to develop a validation sample design after building the emulator, taking into account the training sample design D and the estimated values of the correlation function hyperparameters δ.

Validation sample design is discussed in page DiscCoreValidationDesign. We denote the validation design by $D' = \{x'_1, x'_2, \ldots, x'_{n'}\}$, with n' points. The simulator is run at each of the validation points to produce the output vector $f(D') = (f(x'_1), f(x'_2), \ldots f(x'_{n'})^T)$, where $f(x'_j)$ is the simulator output from the run with input vector x'_j.

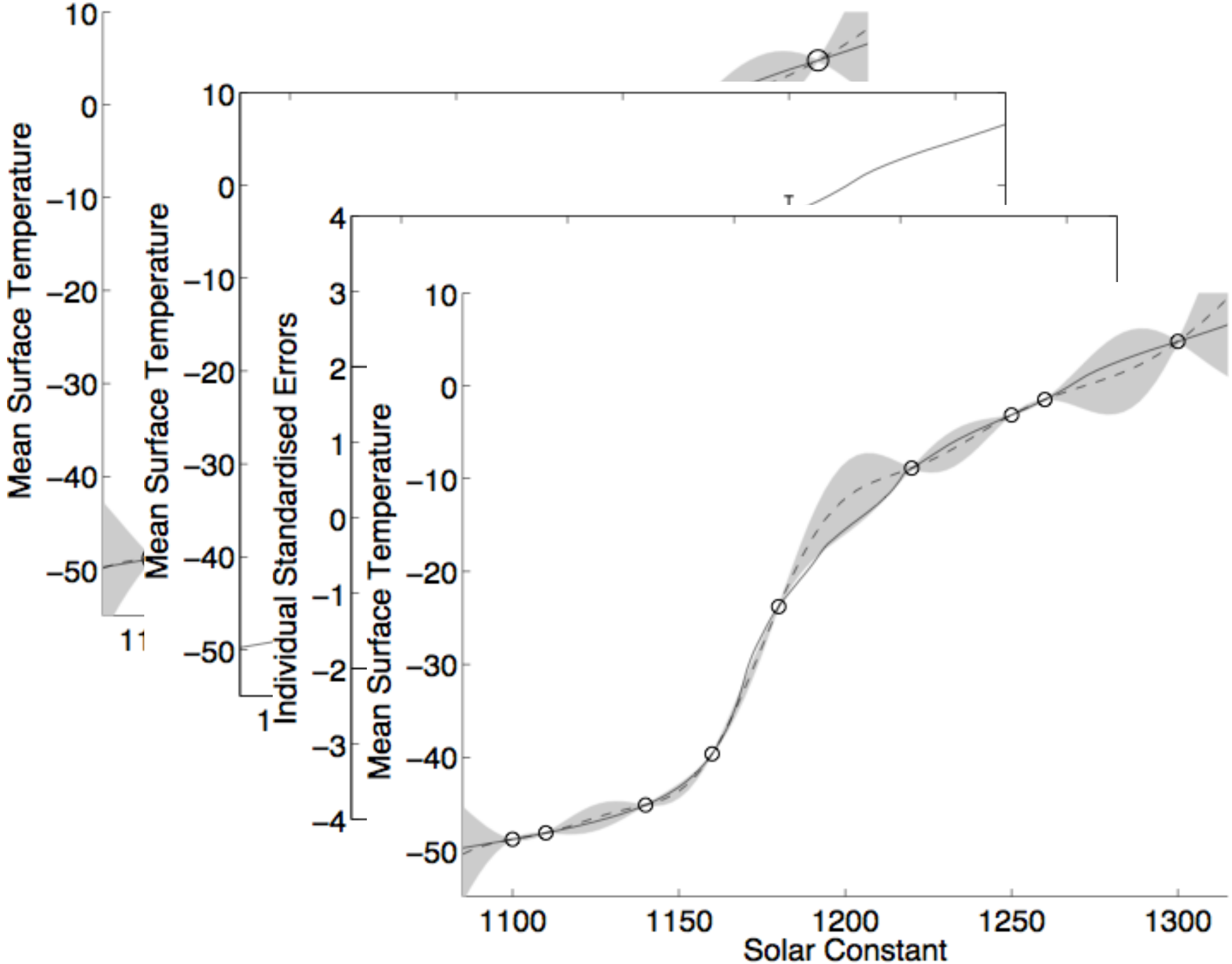


10
0
−10
−20
−30
−40
−50
Mean Surface Temperature
1100
1150
1200
1250
1300
Solar Constant

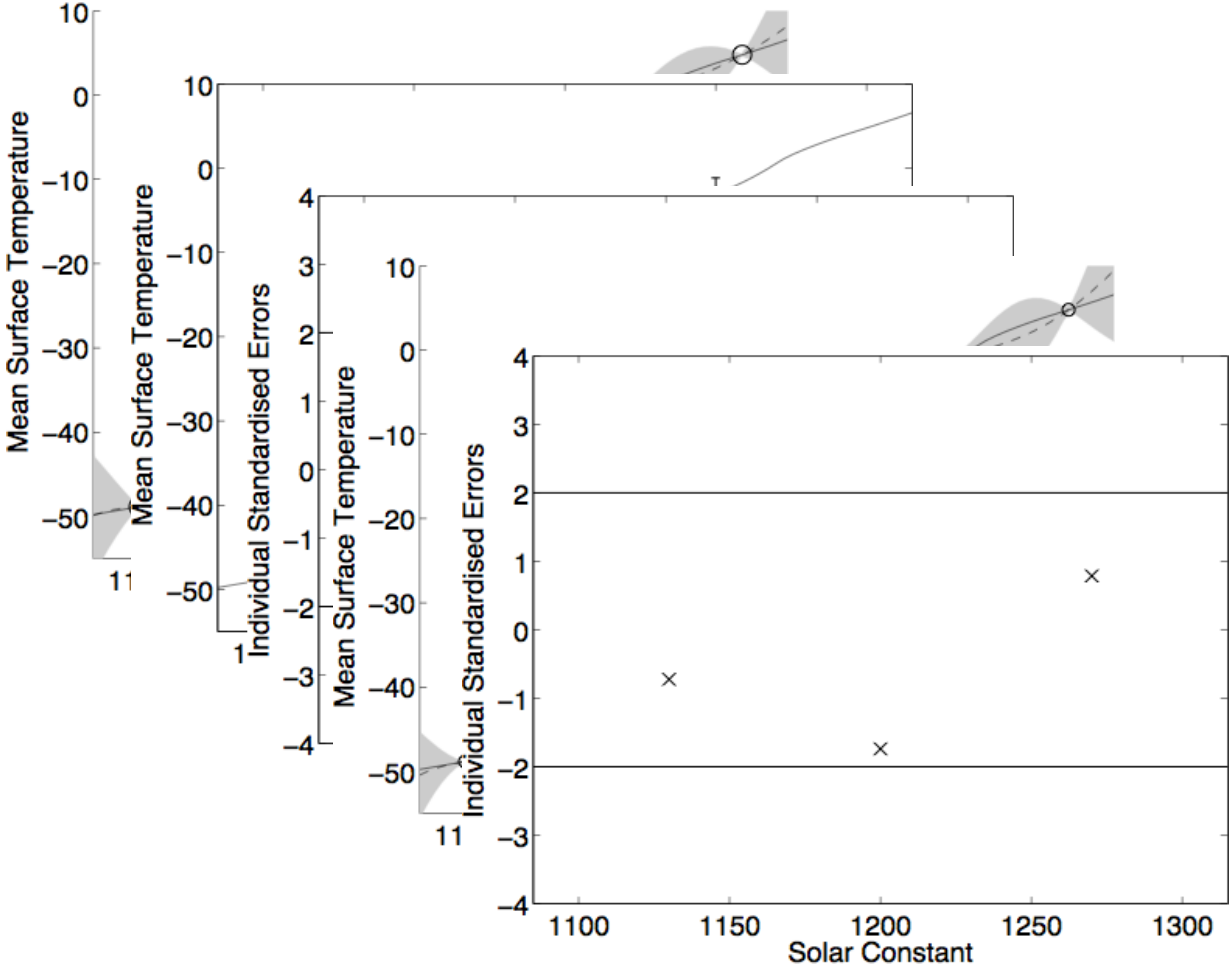

Mean Surface Temperature
Solar Constant



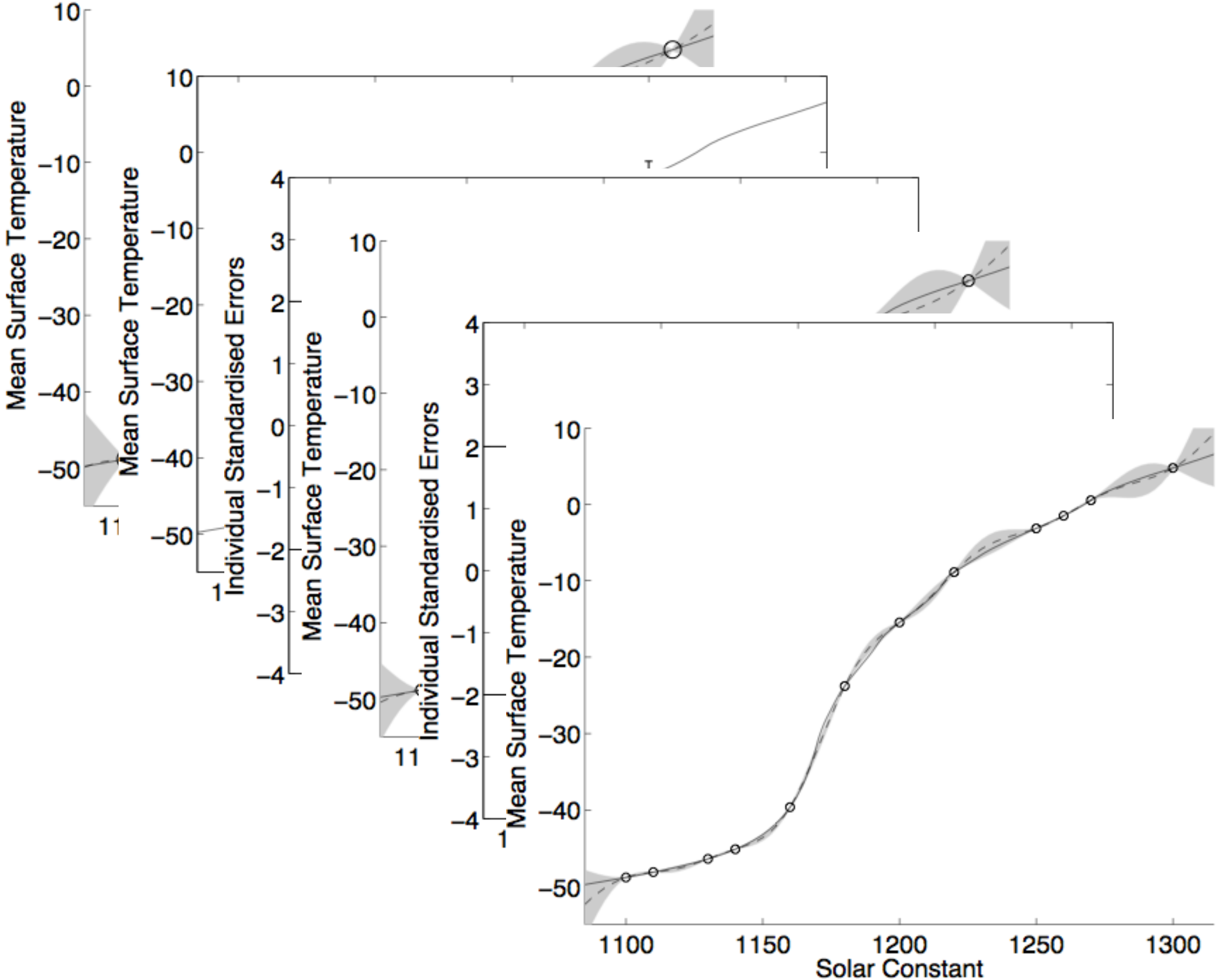
Mean Surface Temperature
Mean Surface Temperature
Individual Standardised Errors
Solar Constant



Mean Surface Temperature
Mean Surface Temperature
Individual Standardised Errors
Mean Surface Temperature
Solar Constant
1100
1150
1200
1250
1300
10
0
−10
−20
−30
−40
−50
4
3
2
1
0
−1
−2
−3
−4

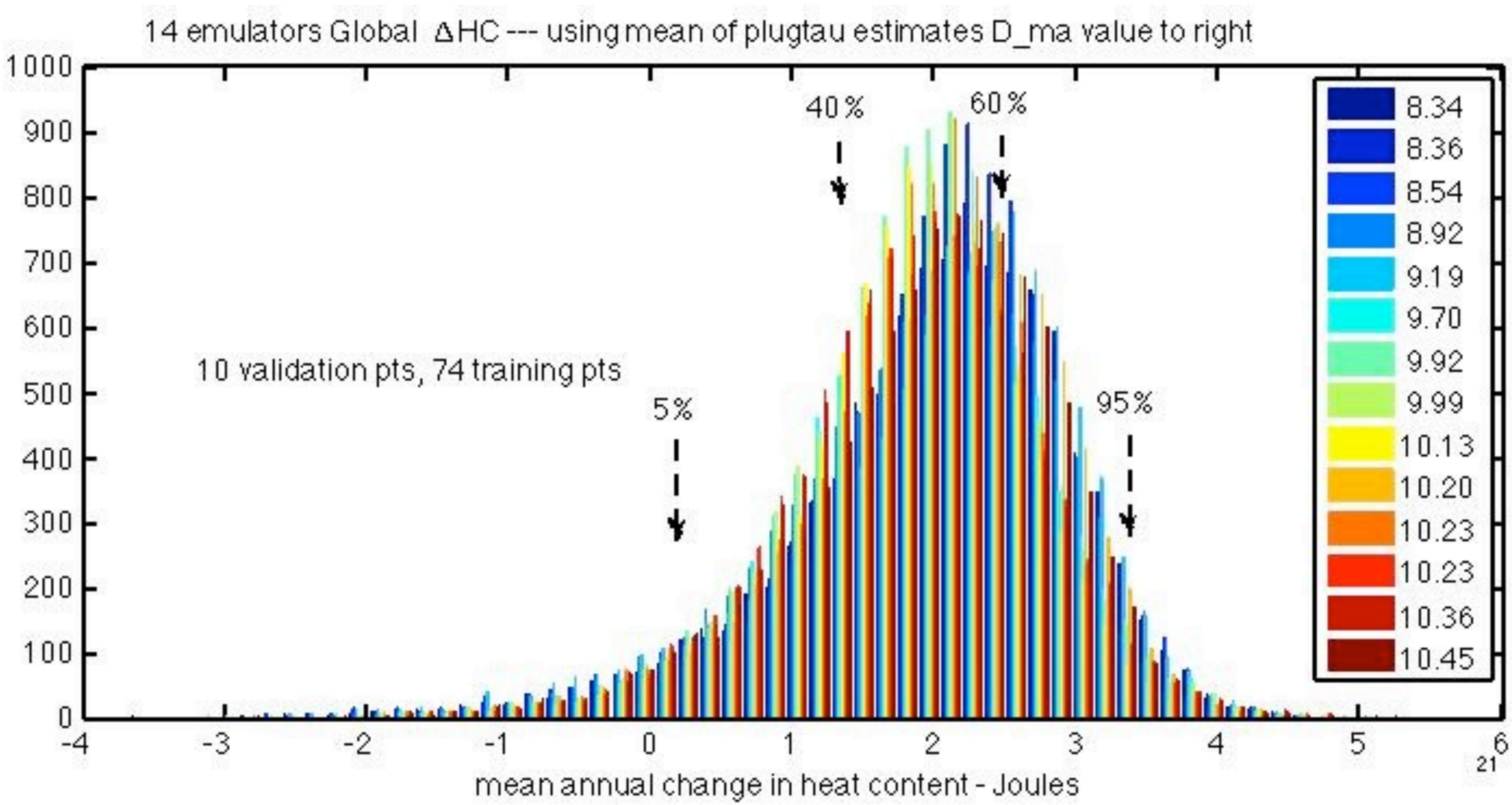


Mean Surface Temperature
Individual Standardised Errors
Solar Constant



Mean Surface Temperature
Individual Standardised Errors
10
0
−10
−20
−30
−40
−50
4
3
2
1
0
−1
−2
−3
−4
1100
1150
1200
1250
1300
Solar Constant

Not Just Toy Models



Courtesy Robin Tokmakian NPS

Not Just Toy Models

MUCM Toolkit

Managing Uncertainty in Complex Models

- Toolkit Home
- Tutorial
- Toolkit Structure
- Threads
- Case Studies
- Page List
- Notation
- Comments

Procedure: Uncertainty Analysis using a GP emulator

Description and Background

One of the simpler tasks that is required by users of simulators is uncertainty analysis (UA), which studies the uncertainty in model outputs that is induced by uncertainty in the inputs. Although relatively simple in concept, UA is both important and demanding. It is important because it is the primary way to quantify the uncertainty in predictions made by simulators. It is demanding because in principle it requires us to evaulate the output at all possible values of the uncertain inputs. The MUCM approach of first building an emulator is a powerful way of making UA feasible for complex and computer-intensive simulators.

This procedure describes how to compute some of the UA measures discussed in the definition page of Uncertainty Analysis (DefUncertaintyAnalysis). In particular, we consider the uncertainty mean and variance:

$$\mathrm{E}[f(X)] = \int_{\mathcal{X}} f(x)\omega(x)\mathrm{d}x$$

$$\mathrm{Var}[f(X)] = \int_{\mathcal{X}} (f(x) - \mathrm{E}[f(x)])^2\omega(x)\mathrm{d}x$$

Notice that it is necessary to specify the uncertainty about the inputs through a full probability distribution for the inputs. This clearly demands a good understanding of probability and its use to express personal degrees of belief. However, this specification of uncertainty often also requires interaction with relevant experts whose knowledge is being used to specify values for individual inputs. There is a considerable literature on the elicitation of expert knowledge and uncertainty in probabilistic form, and some references are given at the end of this page.

In practice, we cannot evaluate either $\mathrm{E}[f(X)]$ or $\mathrm{Var}[f(X)]$ directly from the simulator because the integrals require us to know $f(x)$ at every x. Even evaluating numerically by a Monte Carlo integration approach would require a very large number of runs of the simulator, so this is one task for which emulation is very powerful. We build an emulator from a limited training sample of simulator runs and then use the emulator to evaluate these quantities. We still cannot evaluate them exactly because of uncertainty in the emulator. We therefore present procedures here for calculating the emulator (i.e. posterior) mean of each quantity as an estimate; while the emulator variance provides a measure of accuracy of that estimate. We use E^* and Var^* to denote emulator mean and variance.

We assume here that a Gaussian process (GP) emulator has been built in the form described in the procedure page for building a GP emulator (ProcBuildCoreGP), and that we are only emulating a single output. Note that ProcBuildCoreGP gives procedures for deriving emulators in a number of different forms, and we consider here only the "linear mean and weak prior" case where the GP has a linear mean function, weak prior information is specified on the hyperparameters β and σ^2 and the emulator is derived with a single point estimate for the hyperparameters δ.

Inputs

- An emulator as defined in ProcBuildCoreGP
- A distribution $\omega(.)$ for the uncertain inputs

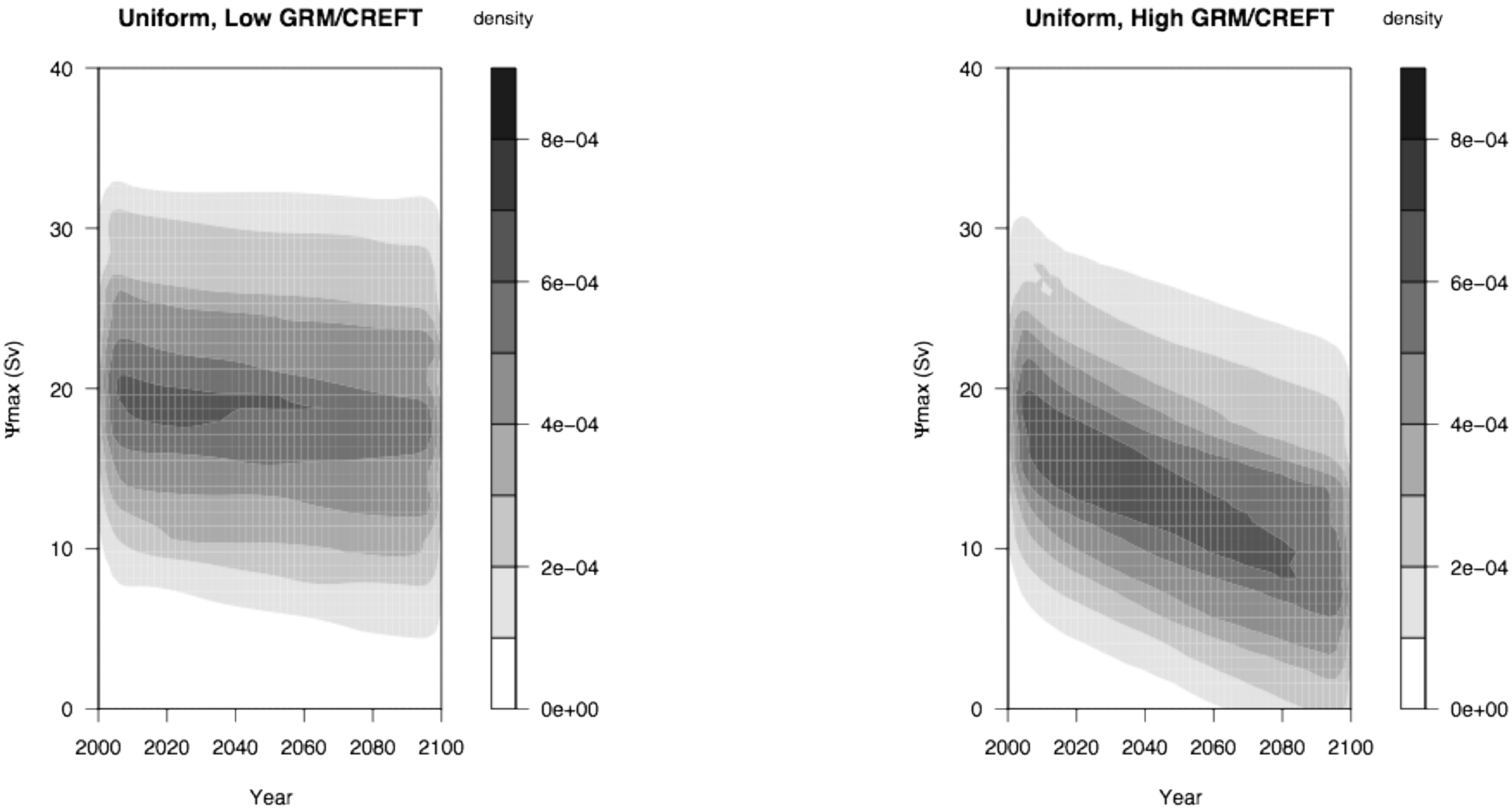
Outputs

- The expected value $\mathrm{E}^*[\mathrm{E}[f(X)]]$ and variance $\mathrm{Var}^*[\mathrm{E}[f(X)]]$ of the uncertainty distribution mean
- The expected value $\mathrm{E}^*[\mathrm{Var}[f(X)]]$ of the uncertainty distribution variance

Beyond the Core

- The core problem is the simplest method.
- More complex methods are built on this.
  - ThreadVariantMultipleOutputs
  - ThreadVariantDynamic
  - ThreadVariantTwoLevelEmulation
  - ThreadVariantWithDerivatives
  - ThreadVariantModelDiscrepancy
  - ThreadGenericMultipleEmulators
  - ThreadGenericEmulateDerivatives
  - ThreadGenericHistoryMatching

Multiple Outputs



MUCM Toolkit

Managing Uncertainty in Complex Models

Thread: Analysis of a simulator with multiple outputs using Gaussian Process methods

The multivariate emulator

The principal user entry points to the MUCM toolkit are the various *threads*, as explained in the Toolkit structure page (MetaToolkitStructure). The main threads give detailed instructions for building and using emulators in various contexts.

This thread takes the user through the analysis of a variant of the most basic kind of problem, using the fully Bayesian approach based on a Gaussian process (GP) emulator. We characterise the basic multi-output model as follows:

- We are only concerned with one simulator.
- The output is deterministic.
- We do not have observations of the real world process against which to compare the simulator.
- We do not wish to make statements about the real world process.
- We cannot directly observe derivatives of the simulator.

Each of these requirements is also a part of the core problem, and is discussed further in DiscCore. However, the core problem further assumes that the simulator only produces one output, or that we are only interested in one output. We relax that assumption here. The core thread ThreadCoreGP deals with the analysis of the core problem using a GP emulator. This variant thread extends the core analysis to the case of a simulator with more than one output.

The fully Bayesian approach has a further restriction:

- We are prepared to represent the simulator as a Gaussian process.

There is discussion of this requirement in DiscGaussianAssumption.

Alternative approaches to emulation

There are various approaches to tackling the problems raised by having multiple outputs, which are discussed in the alternatives page on emulating multiple outputs (AltMultipleOutputsApproach). Some approaches reduce or transform the multi-output model so that it can be analysed by the methods in ThreadCoreGP. However, others employ a multivariate GP emulator that is described in detail in the remainder of this thread.

The GP model

The first stage in building the emulator is to model the mean and covariance structures of the Gaussian process that is to represent the simulator. As explained in the definition of a multivariate Gaussian process, a GP is characterised by a mean function and a covariance function. We model these functions to represent *prior* beliefs that we have about the simulator, i.e. beliefs about the simulator prior to incorporating information from the training sample.

Alternative choices of the emulator prior mean function are considered in AltMeanFunction, with specific discussion on the multivariate case in AltMeanFunctionMultivariate. In general, the choice will lead to the mean function depending on a set of hyperparameters that we will denote by β. We will generally write the mean function as $m(\cdot)$ where the dependence on β is implicit. Note that if we have r outputs, then $m(\cdot)$ is a vector of $1 \times r$ elements comprising the mean functions of the various outputs.

Calibration and History Matching

Calibration and History Matching

- Simulator users often want to tune the simulator using observations of the real system

Calibration and History Matching

- Simulator users often want to tune the simulator using observations of the real system
- Calibration finds estimates and uncertainties for the ‘best’ inputs

Calibration and History Matching

- Simulator users often want to tune the simulator using observations of the real system
- Calibration finds estimates and uncertainties for the ‘best’ inputs
- History Matching finds regions of ‘not implausible inputs’

Calibration and History Matching

- Simulator users often want to tune the simulator using observations of the real system
- Calibration finds estimates and uncertainties for the 'best' inputs
- History Matching finds regions of 'not implausible inputs'
- Two very important points

1. Calibration will reduce uncertainty about x but will not eliminate it
2. It is necessary to understand how the simulator relates to reality
    - Model discrepancy

Model discrepancy

Model discrepancy

- Simulator output $y = f(x)$ will not equal the real system value z
  - Even with best/correct inputs x

Model discrepancy

- Simulator output $y = f(x)$ will not equal the real system value z
  - Even with best/correct inputs x
- Model discrepancy is the difference $z - f(x)$

Model discrepancy

- Simulator output $y = f(x)$ will not equal the real system value z
    - Even with best/correct inputs x
- Model discrepancy is the difference $z - f(x)$
- Can be expressed a mean $(z - f(x))$ or a variance $(z - f(x))^2$

Model discrepancy

- Simulator output $y = f(x)$ will not equal the real system value z
    - Even with best/correct inputs x
- Model discrepancy is the difference $z - f(x)$
- Can be expressed a mean $(z - f(x))$ or a variance $(z - f(x))^2$
- Model discrepancy is due to
    - Wrong or incomplete science
    - Programming errors, rounding errors
    - Inaccuracy in numerically solving systems of equations

Model discrepancy

- Simulator output $y = f(x)$ will not equal the real system value z
  - Even with best/correct inputs x
- Model discrepancy is the difference $z - f(x)$
- Can be expressed a mean $(z - f(x))$ or a variance $(z - f(x))^2$
- Model discrepancy is due to
  - Wrong or incomplete science
  - Programming errors, rounding errors
  - Inaccuracy in numerically solving systems of equations
- Ignoring model discrepancy leads to poor calibration
  - Over-fitting of parameter estimates
  - Over-confidence in the fitted values

History Matching

Implausibility

- Define a measure of implausibility (I_{mp})

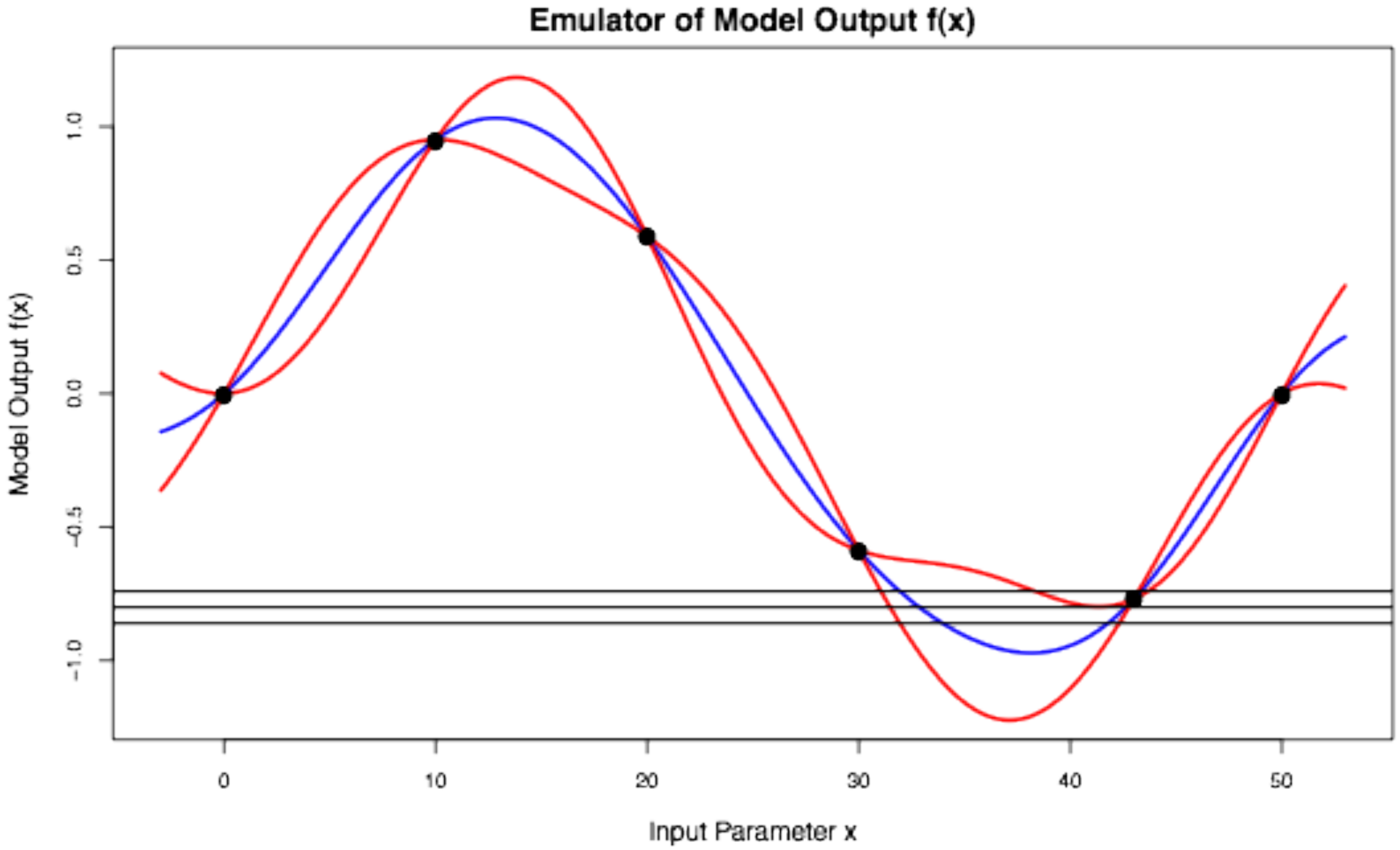
$$I_{mp}^2 = \frac{(x_{obs} - x_{emul})^2}{\sigma_{emul}^2 + \sigma_{obs}^2 + \sigma_{discrep}^2}$$

- If the implausibility is greater then ±3 those values of the inputs are deemed implausible
- Because this is a function of the emulator not the original simulator runs we calculate it everywhere in input space

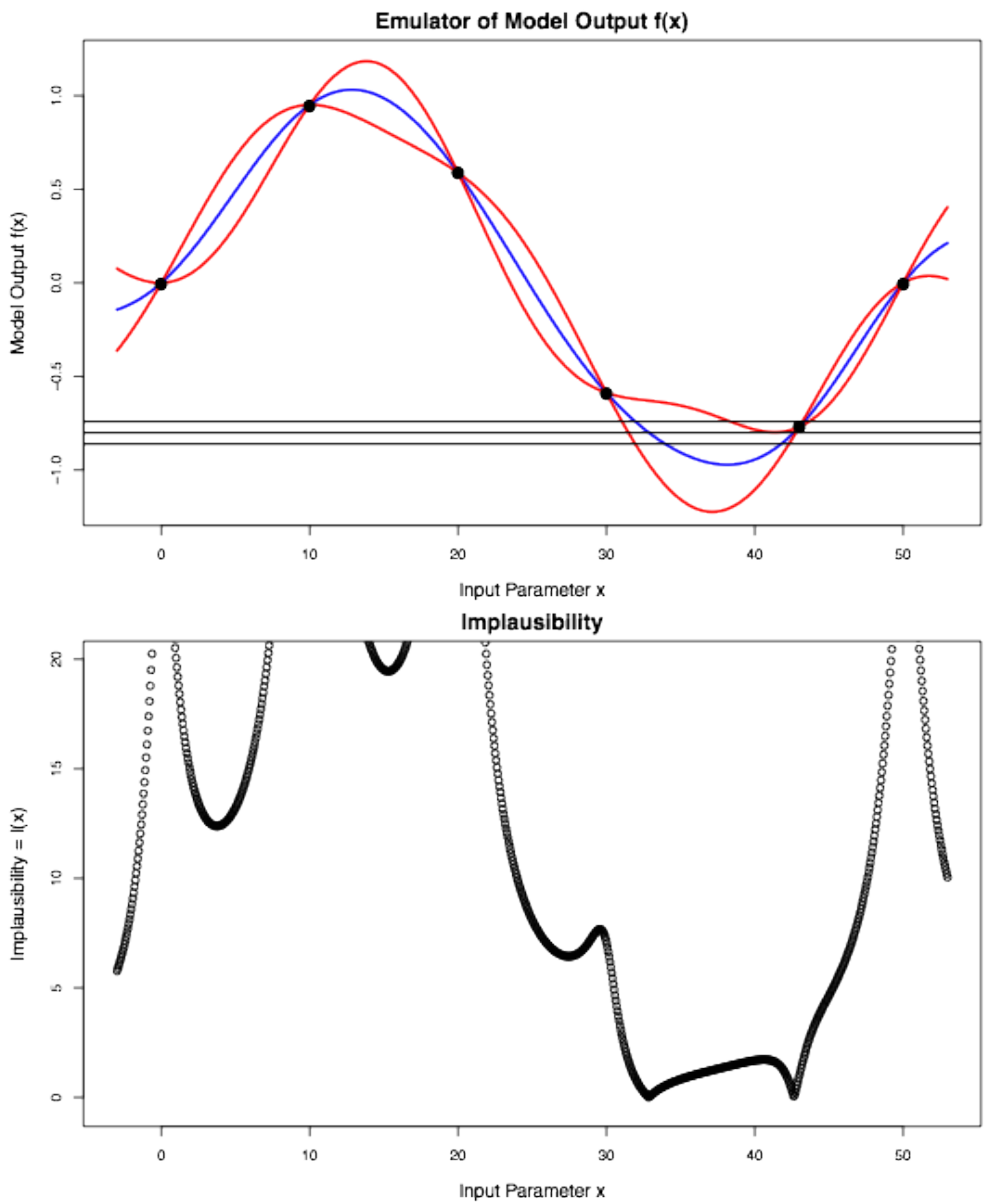
Waves of Implausibility

- Wave 1: Apply the implausibility measure. Mark part of input space as implausible
- Wave 2: Add extra points in the not implausible region and rebuild the emulator. Repeat the implausibility measure
- Wave 3+: Repeat until the implausible region ceases to grow

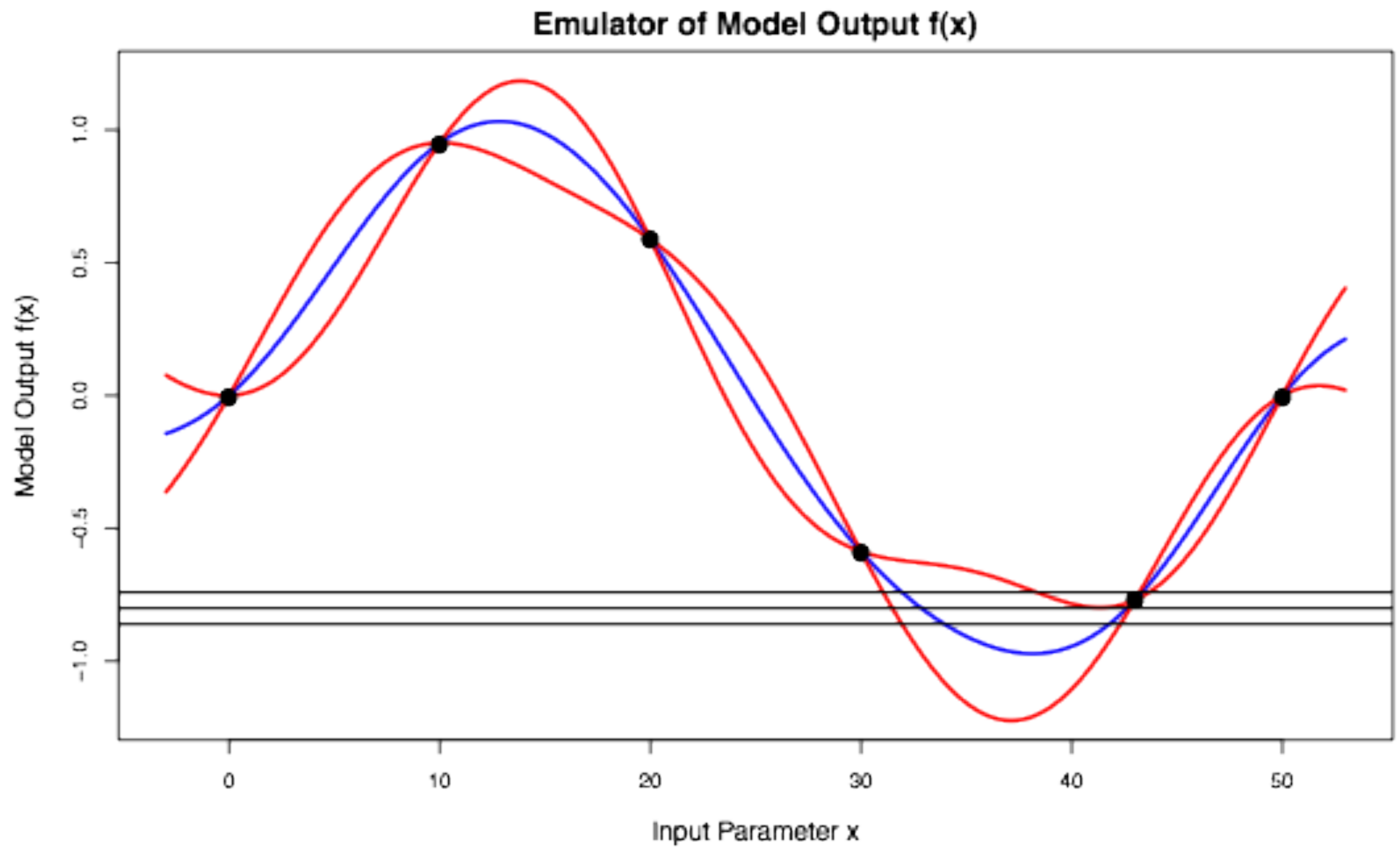
A 1-d example

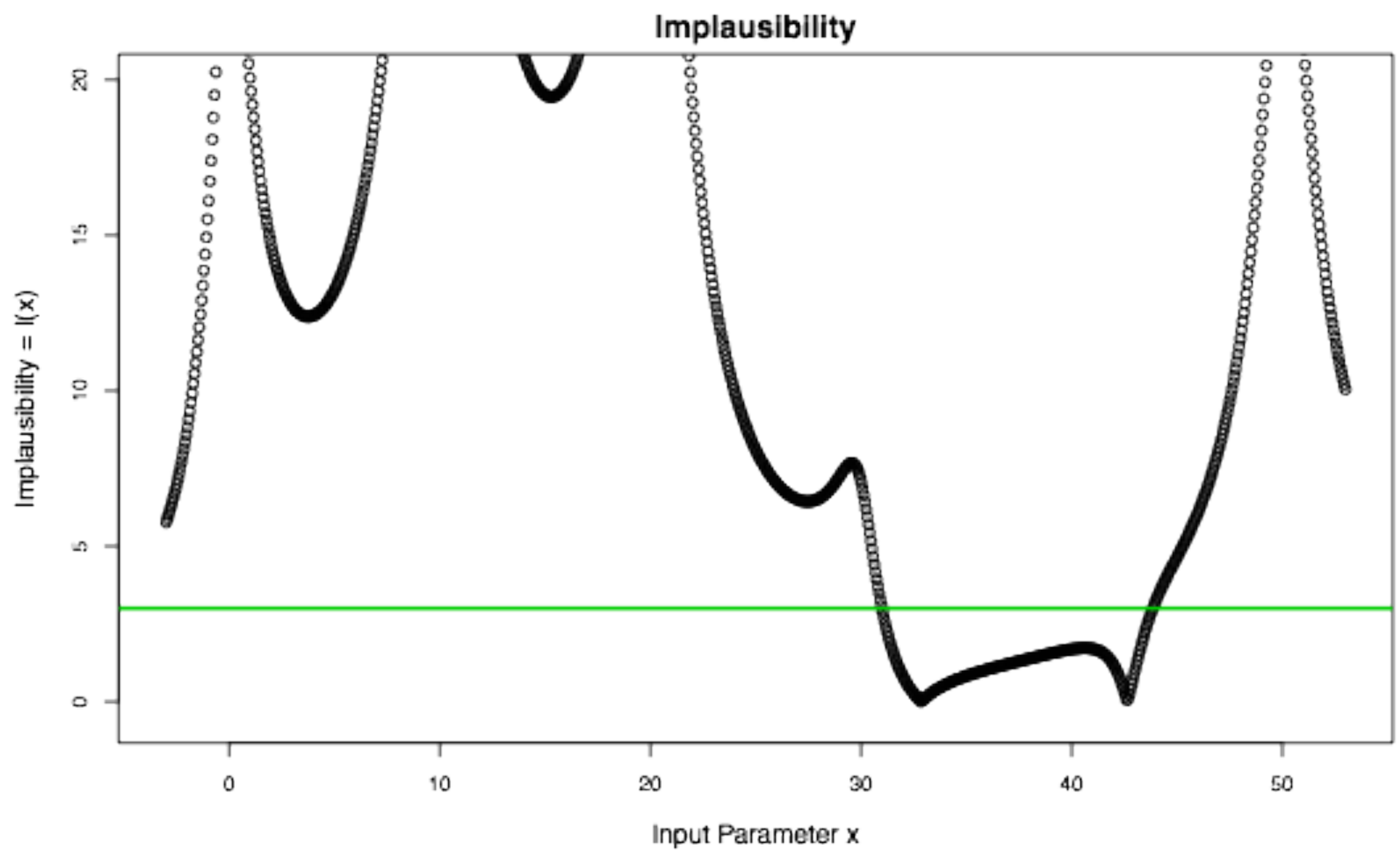


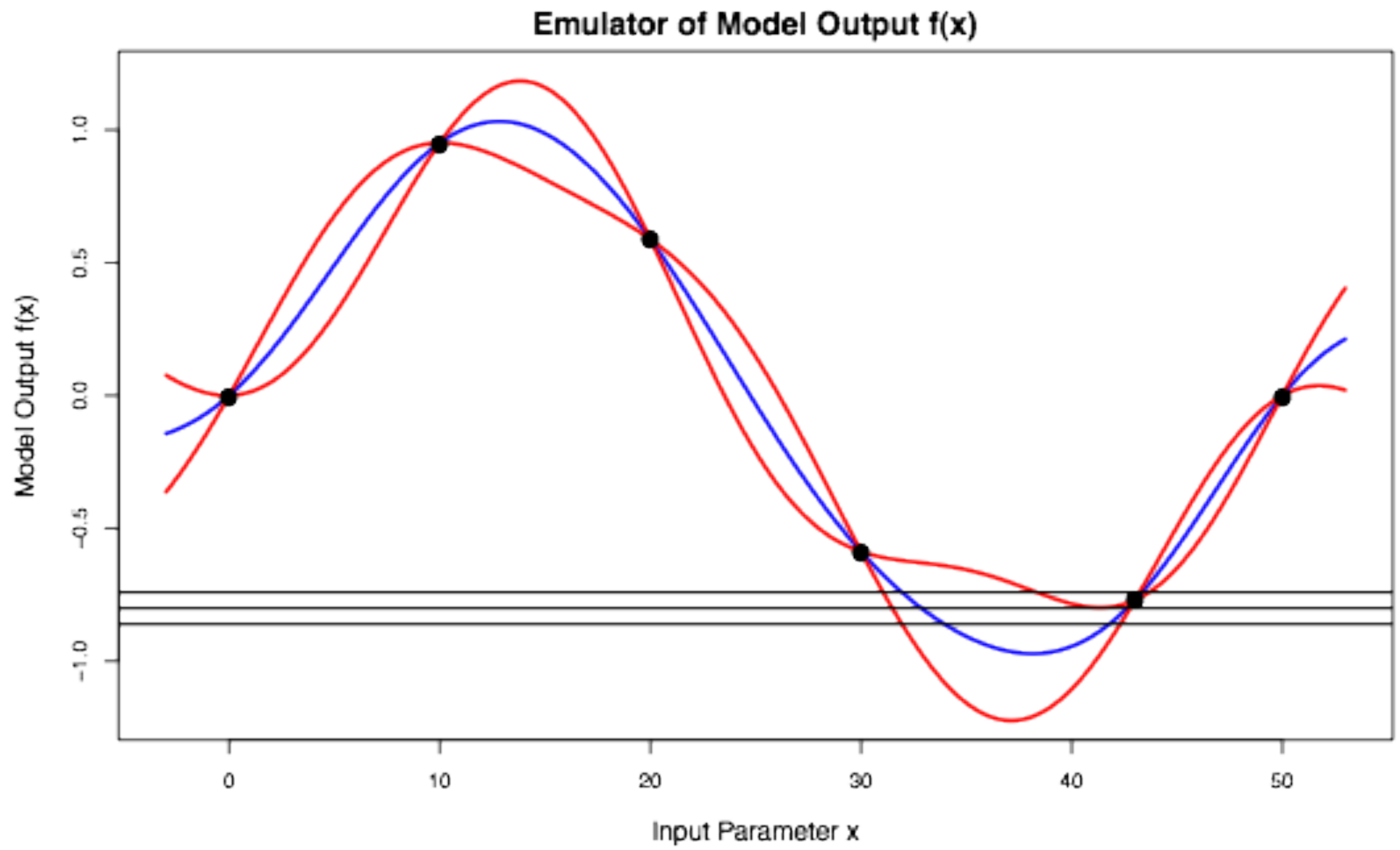
Thanks to Ian Vernon U. Durham



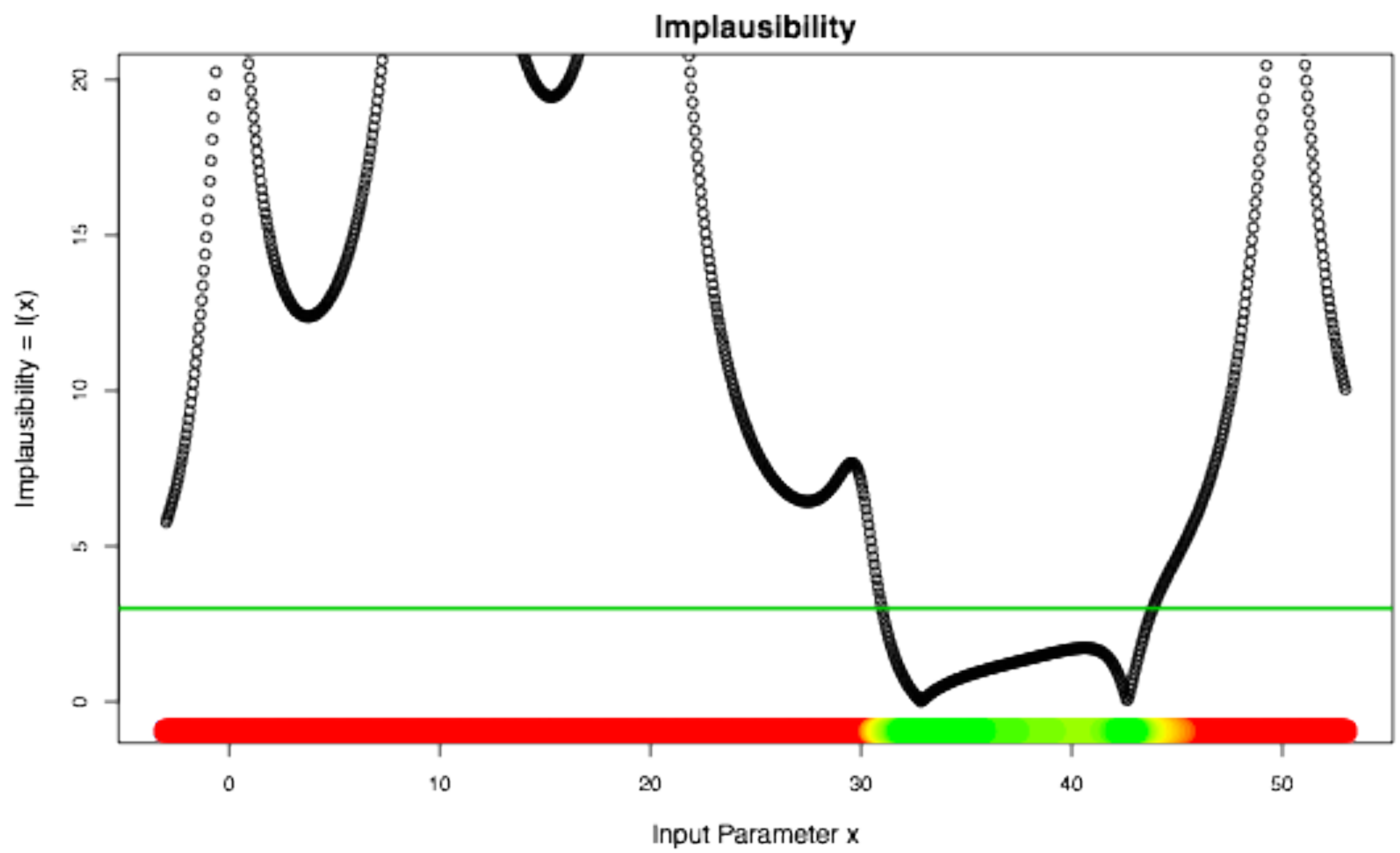
Emulator of Model Output f(x)
Model Output f(x)
1.0
0.5
0.0
−0.5
−1.0
0
10
20
30
40
50
Input Parameter x
Implausibility
Implausibility = I(x)
20
15
10
5
0
0
10
20
30
40
50
Input Parameter x


Emulator of Model Output f(x)
Model Output f(x)
1.0
0.5
0.0
−0.5
−1.0
0
10
20
30
40
50
Input Parameter x

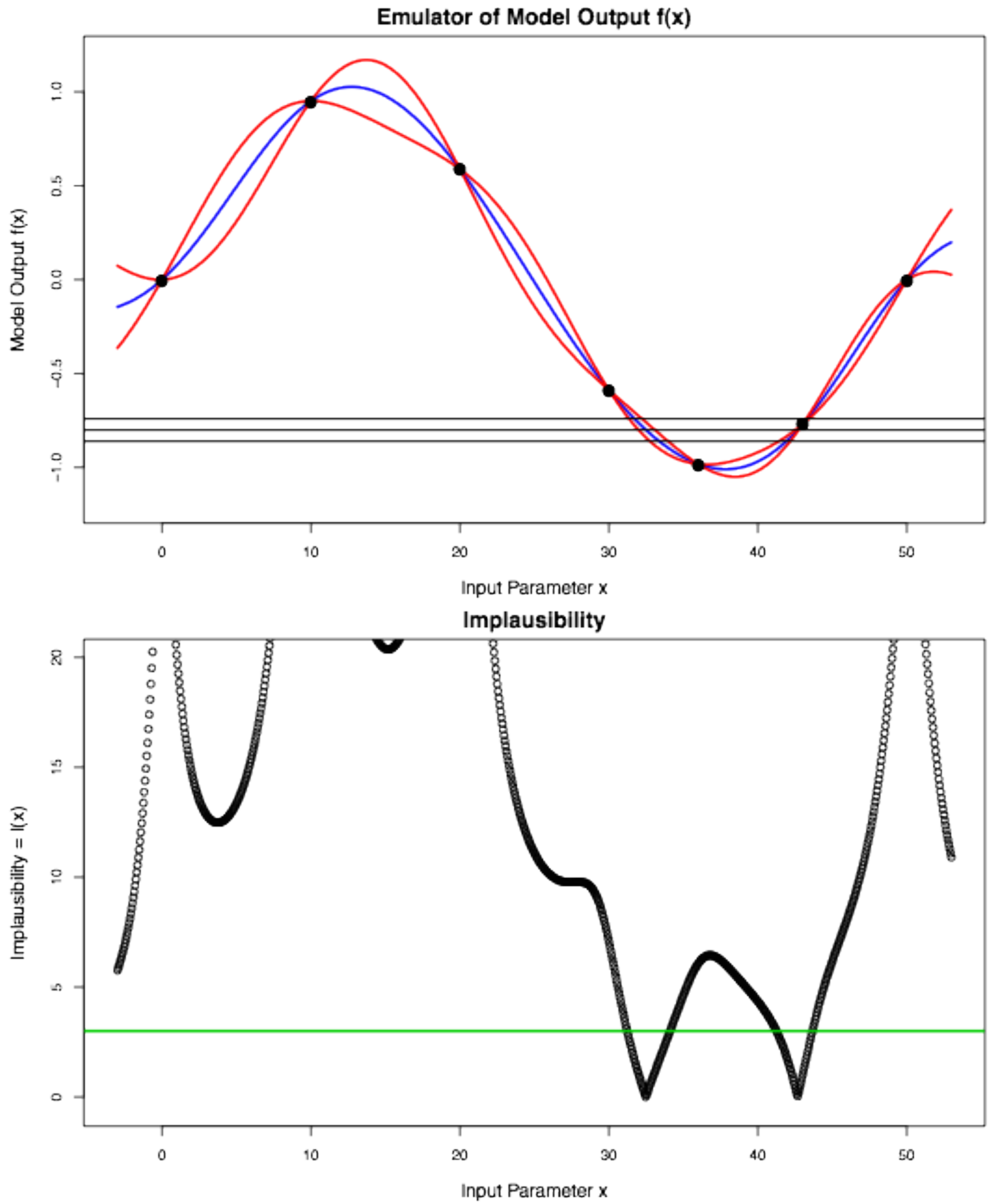

Implausibility
Implausibility = I(x)
20
15
10
5
0
0
10
20
30
40
50
Input Parameter x



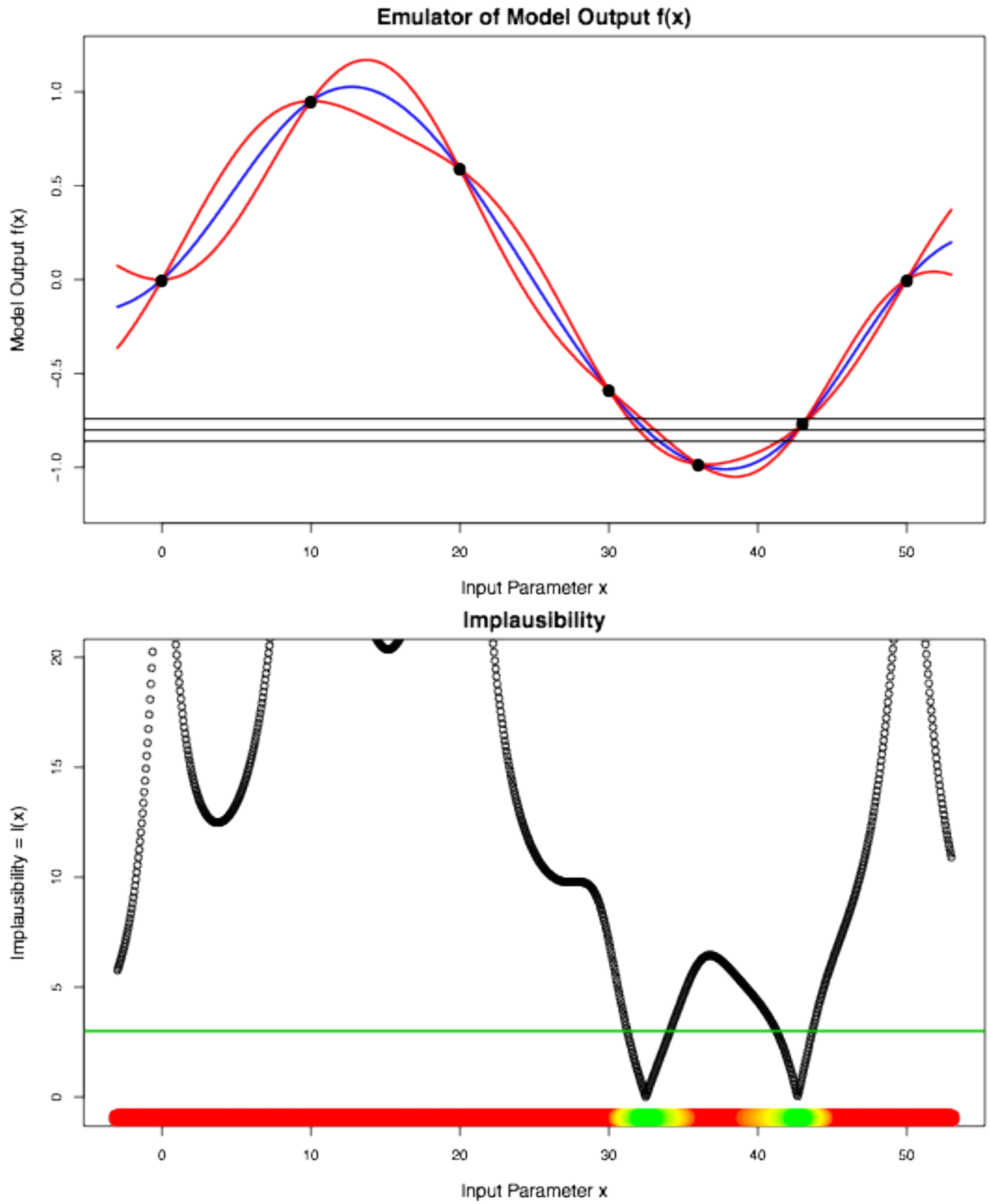
Emulator of Model Output f(x)
Model Output f(x)
1.0
0.5
0.0
-0.5
-1.0
0
10
20
30
40
50
Input Parameter x

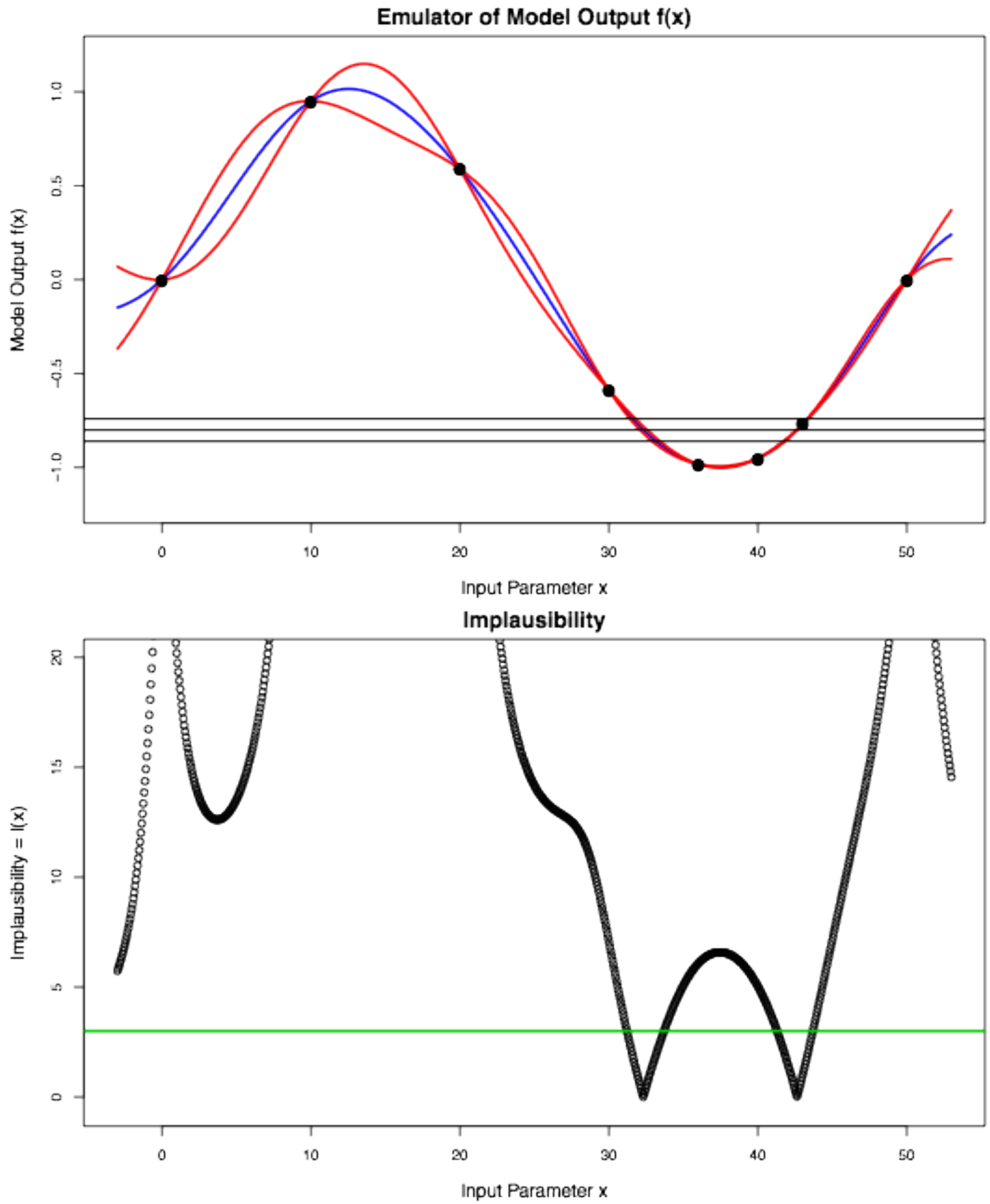


Implausibility
Implausibility = I(x)
20
15
10
5
0
0
10
20
30
40
50
Input Parameter x

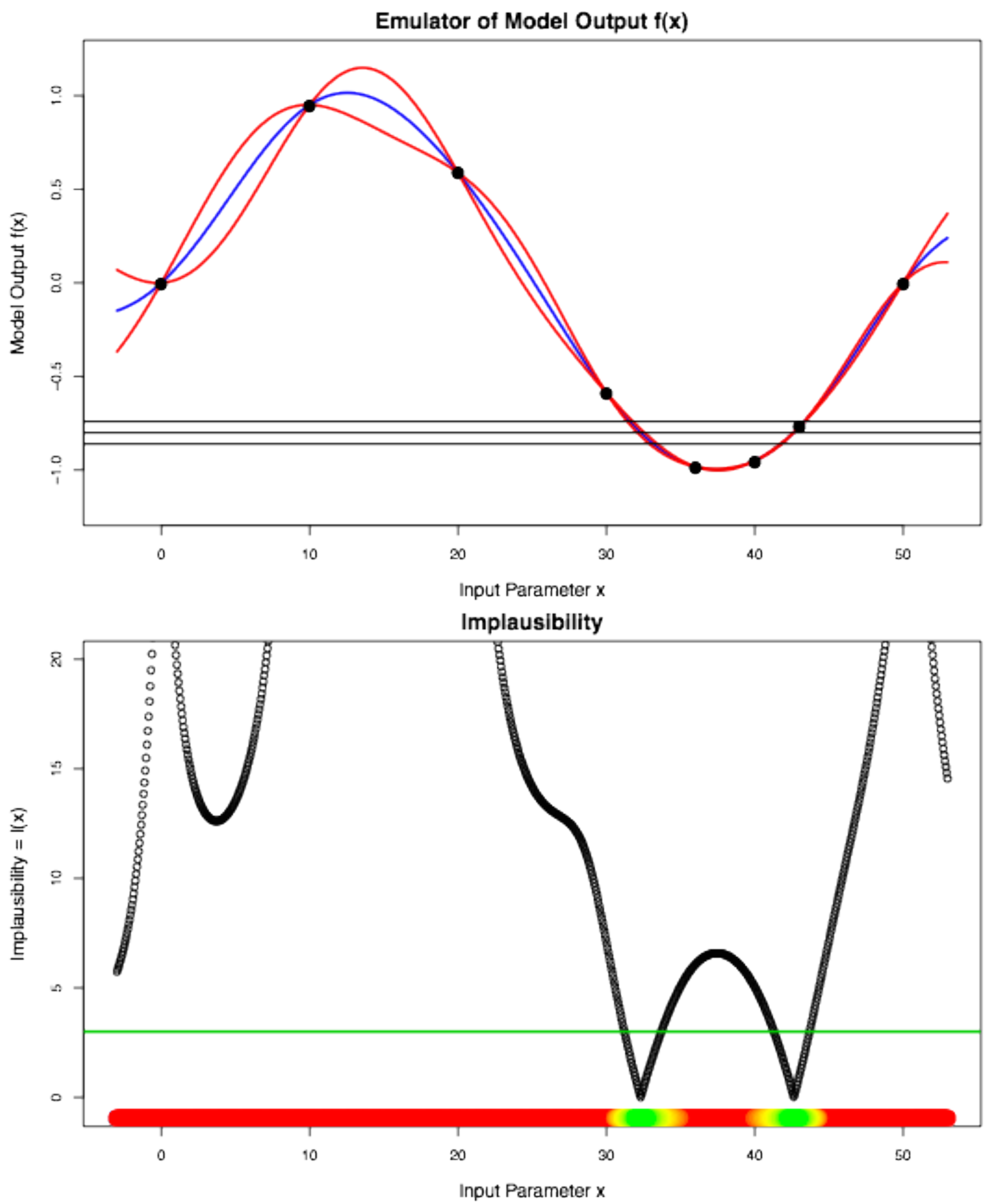


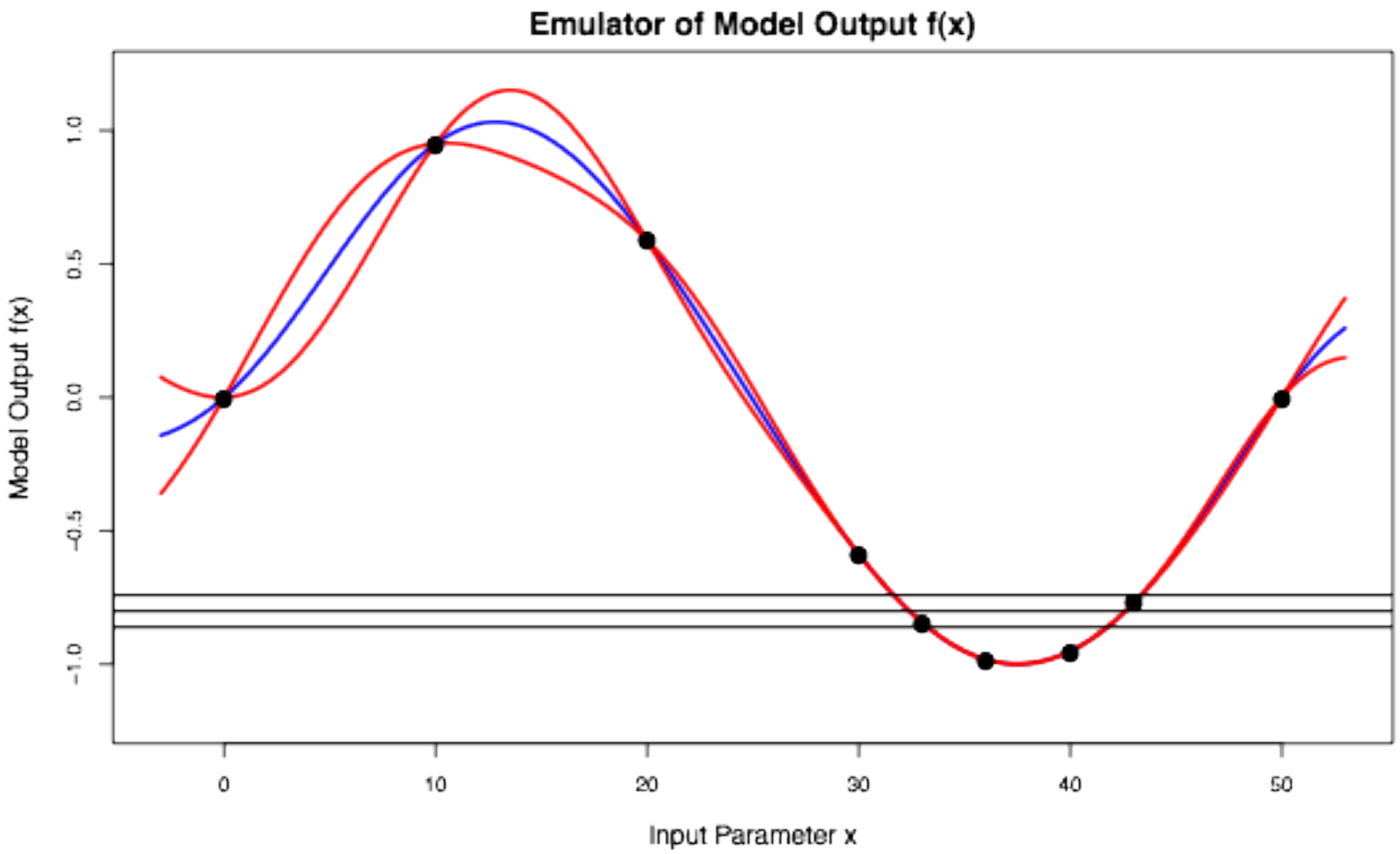
Emulator of Model Output f(x)
Model Output f(x)
Input Parameter x
Implausibility
Implausibility = I(x)
Input Parameter x


Emulator of Model Output f(x)
Model Output f(x)
1.0
0.5
0.0
-0.5
-1.0
0
10
20
30
40
50
Input Parameter x
Implausibility
Implausibility = I(x)
20
15
10
5
0
0
10
20
30
40
50
Input Parameter x

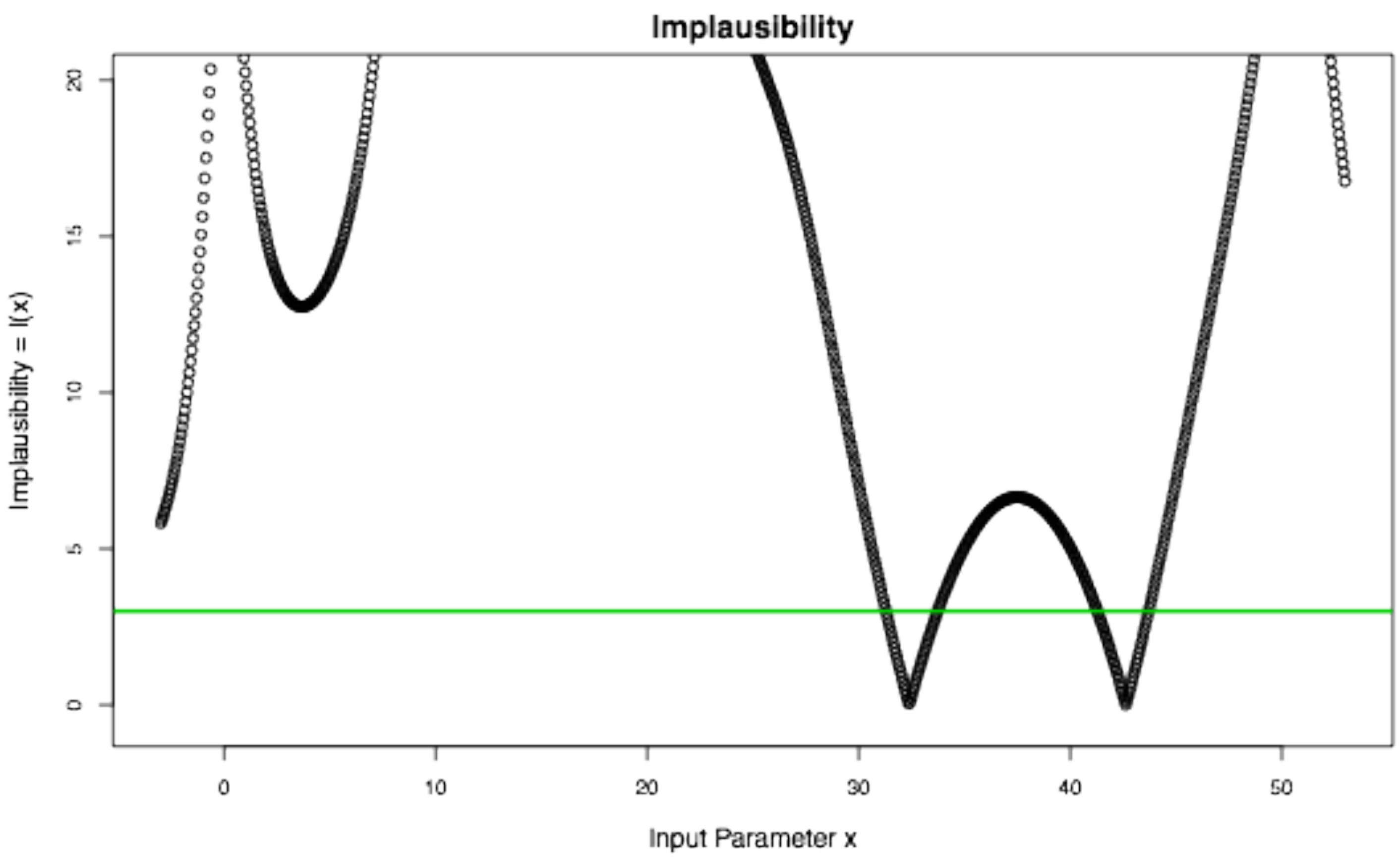


Emulator of Model Output f(x)
Model Output f(x)
1.0
0.5
0.0
−0.5
−1.0
0
10
20
30
40
50
Input Parameter x
Implausibility
Implausibility = I(x)
20
15
10
5
0
0
10
20
30
40
50
Input Parameter x

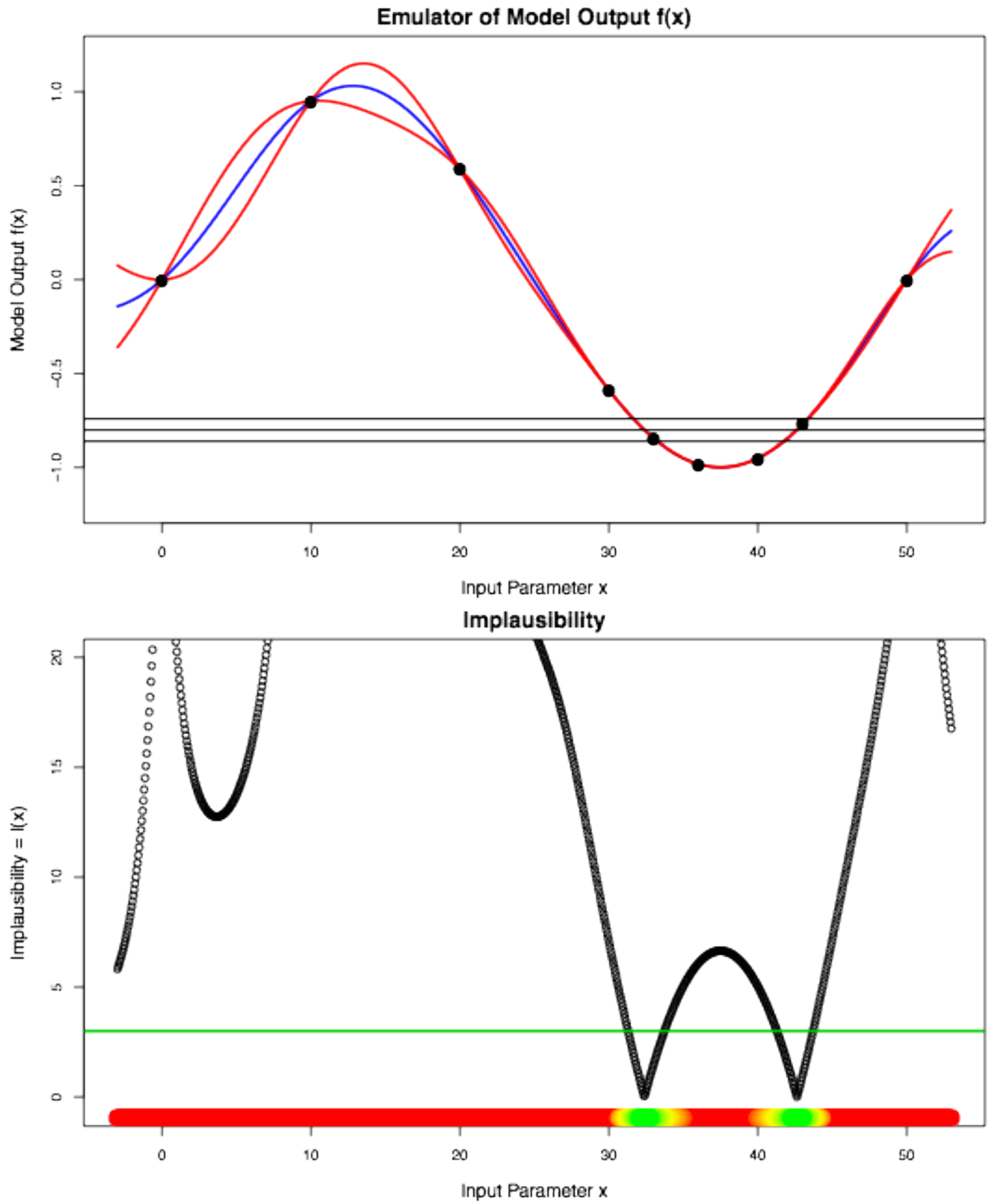

Emulator of Model Output f(x)
Model Output f(x)
Input Parameter x
Implausibility
Implausibility = I(x)
Input Parameter x



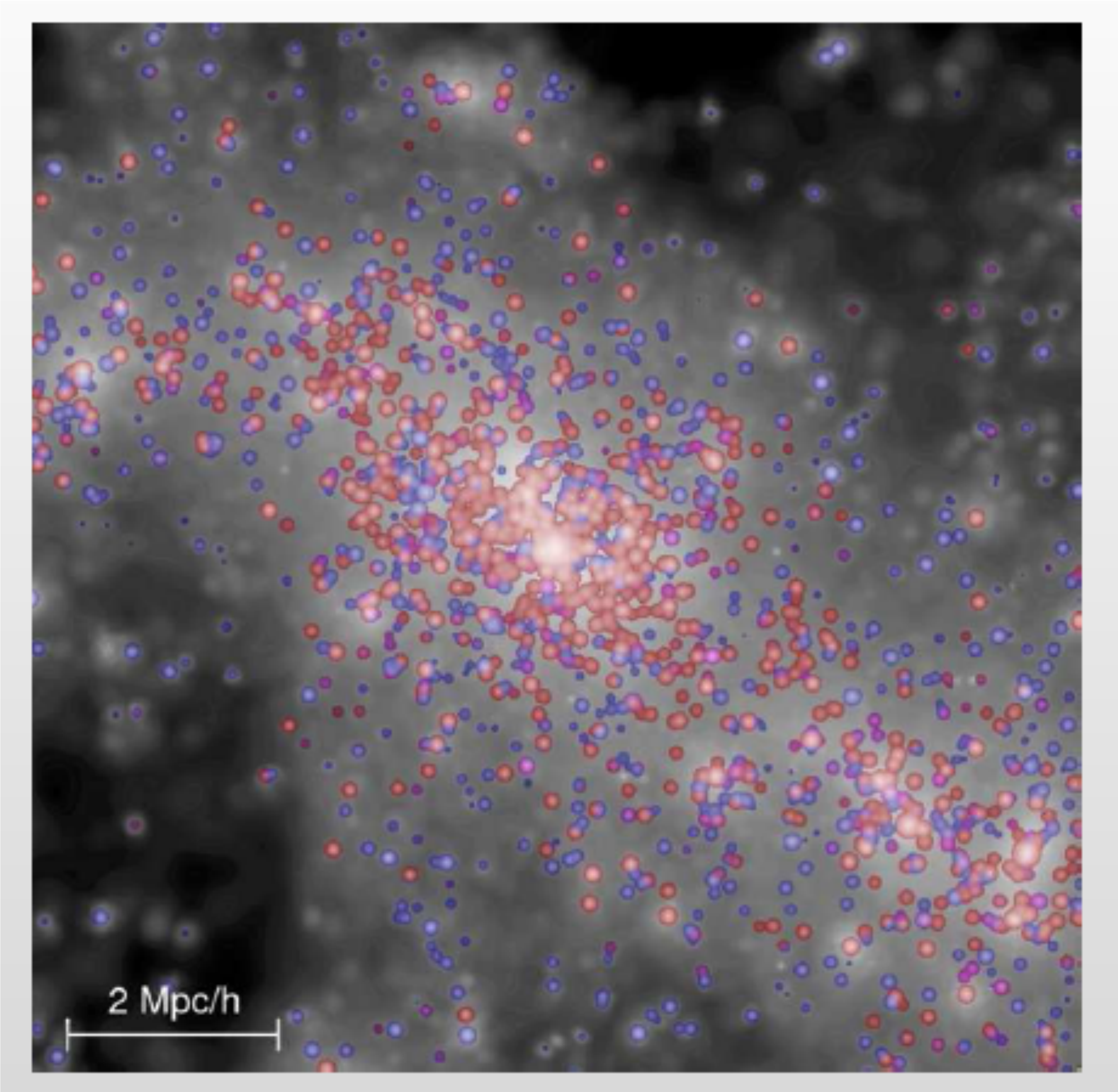
Emulator of Model Output f(x)
Model Output f(x)
Input Parameter x



Implausibility
Implausibility = I(x)
Input Parameter x


Emulator of Model Output f(x)
Model Output f(x)
Input Parameter x
Implausibility
Implausibility = I(x)
Input Parameter x

Example -Galform



Example - Galform

- Galform is a simulator of Galaxy formation
- It has 17 inputs
- The amount of not implausible space in each wave is

Wave 1	14.9%
Wave 2	5.9%
Wave 3	1.6%
Wave 4	0.26%
Wave 5	0.036%

- None of the original 1000 member LHC was an acceptable fit to the data

MUCM Toolkit

Managing Uncertainty in Complex Models

Thread: History Matching

Overview

The principal user entry points to the MUCM toolkit are the various threads, as explained in MetaToolkitStructure. This thread takes the user through a technique known as history matching, which is used to learn about the inputs x to a model $f(x)$ using observations of the real system z. As the history matching process typically involves the use of expectations and variances of emulators, we assume that the user has successfully emulated the model using the Bayes Linear strategy as detailed in ThreadCoreBL. An associated technique corresponding to a fully probabilistic emulator, as described in ThreadCoreGP, will be discussed in a future release. Here we use the term model synonymously with the term simulator.

The description of the link between the model and the real system is vital in the history matching process, therefore several of the concepts discussed in ThreadVariantModelDiscrepancy will be used here.

Calibration

Calibration

- We can also calibrate the simulator(possibly after history matching)
- Include a model discrepancy term
- Reality = simulator + discrepancy

Calibration

- We can also calibrate the simulator(possibly after history matching)
- Include a model discrepancy term
- Reality = simulator + discrepancy

$$y(x_{con}) = f(x_{con}, x_{cal}) + d(x_{con})$$

Calibration

- We can also calibrate the simulator(possibly after history matching)
- Include a model discrepancy term
- Reality = simulator + discrepancy

$$y(x_{con}) = f(x_{con}, x_{cal}) + d(x_{con})$$

- Split the inputs in control inputs (x_{con}) and calibration inputs (x_{cal})
- Simultaneously fit a GP to both the simulator output and the discrepancy
- Obtain posteriors for the the best inputs, the observational error, the emulator and the discrepancy function

Calibration

Thread: Calibration

Overview

Calibration is the process of learning from observations of a real process about how to use a simulator of that process to best approximate and predict reality. Calibration brings together a number of different elements that are dealt with individually elsewhere in the Toolkit.

- The simulator itself, which we will usually need to approximate using an emulator
- The best input values of the calibration parameters needed to tune the simulator to reality
- A representation of the relationship between the simulator (using best inputs) and reality through a model discrepancy function
- A formulation of how the observations of the real process relate to the variables represented by simulator outputs.

In principle, the observations of reality enable us to learn about all four of these elements, because in general there is uncertainty regarding all four.

- We will learn about the best input values of calibration parameters, favouring values which bring the simulator outputs close to the observations.
- We will learn about model discrepancy through the fact that even with best input values the simulator will not predict the observations perfectly.
- We will learn about the observation process, for instance the variance of observation error, through the residual noise after accounting for (smooth) model discrepancy.
- We will learn about the simulator, favouring values within the range of uncertainty of the emulator that allow the simulator to be tuned well without using a priori implausible values for the calibration parameters.

All of this learning will be heavily interlinked, and since the number of observations is often very limited the learning about any one element may be minimal. In this thread, in addition to the full calibration which puts all of the uncertain elements into the analysis, we consider methods which compromise by making simplifying assumptions and so bring out some learning more strongly.

This thread adopts the Bayesian perspective (as opposed to the Bayes linear one; see AltGPorBLEmulator).

Notation and terminology

In accordance with toolkit notation, in this page we use the following symbols:

- x - inputs to the simulator
- $f(x)$ - output(s) of the simulator
- y - actual system value(s)
- z - observations of reality y
- d - model discrepancy

Using the additional notation introduced in the model discrepancy thread page ThreadVariantModelDiscrepancy, model discrepancy links the simulator to reality via the model discrepancy equation, which is usually written

$$y(x_{con}) = f(x_{con}, x_{cal}^{+}) + d(x_{con}),$$

where y, f and d are all vectors of r elements corresponding to the r simulator outputs and we have divided the inputs x into control inputs x_{con} and calibration

Stochastic Simulators

- Some simulators are not deterministic
- Current work looks at emulating means and variances in a linked way
- Design
- Extremes
- Not in the toolkit - yet

Conclusions

- Emulators can be used to quantify uncertainty in complex systems
- The MUCM toolkit is a useful resource
    - Updated every 3 months
    - Comments/feedback needed
    - Next update imminent!
- Still work to do - we are not finished yet