Emulating computer simulators with high dimensional input and output

David Crevillen Andrew Cliffe Marco Iglesias
Henry Power[2] **Richard Wilkinson**

School of Mathematical Sciences

[2]Faculty of Engineering
University of Nottingham

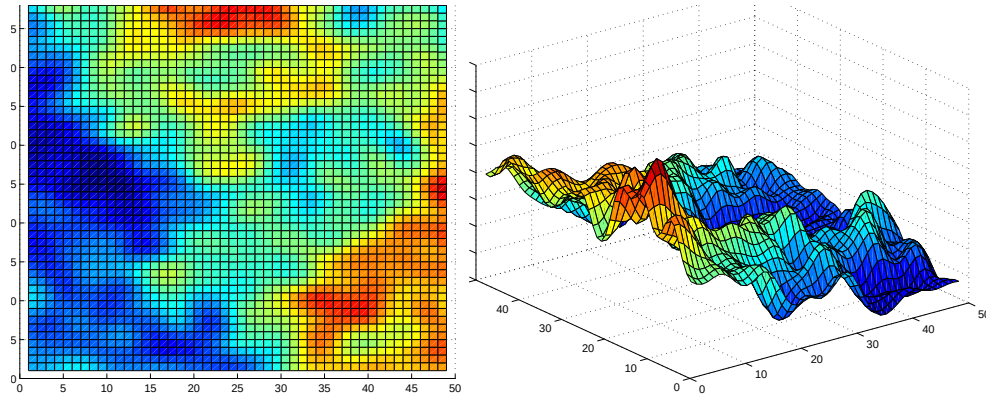
PANACEA Trondheim 2014

Introduction

Knowledge of the physical problem is encoded in a simulator f

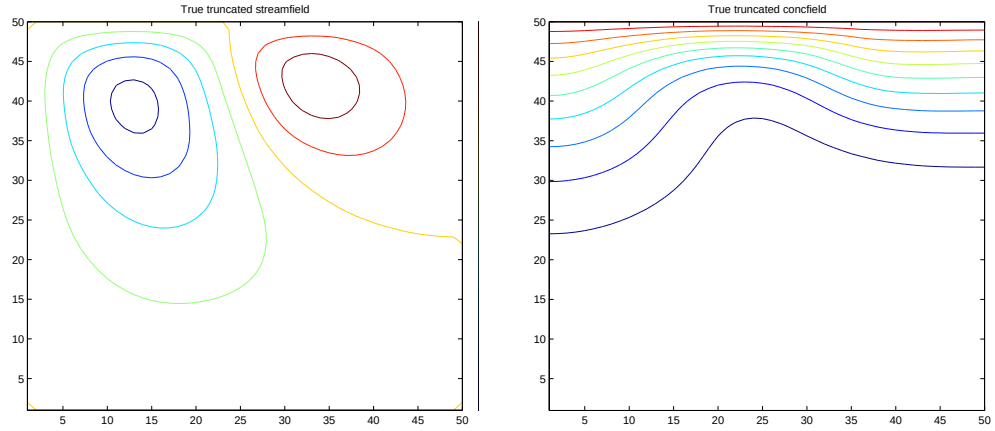
Inputs:

Permeability field, K
(2d field)



$\downarrow f(K)$

$f(K)$



Outputs:

Stream func. (2d field),
concentration (2d field),
surface flux (1d scalar),
$\vdots$

Surface Flux$= 6.43, \ldots$

Uncertainty quantification (UQ) for CCS

The simulator maps from permeability field K to outputs such as the surface flux $\mathcal{S}$. Let $f(K)$ denote this mapping

$$f : K \rightarrow \mathcal{S}$$

For most problems the permeability K is unknown.

Uncertainty quantification (UQ) for CCS

The simulator maps from permeability field K to outputs such as the surface flux $\mathcal{S}$. Let $f(K)$ denote this mapping

$$f : K \rightarrow \mathcal{S}$$

For most problems the permeability K is unknown.

If we assume a distribution for $K \sim \pi(K)$, we can quantify our uncertainty about $\mathcal{S} = f(K)$.

- e.g., by finding the cumulative distribution function (CDF) of $\mathcal{S}$:

$$F(s) = \mathbb{P}(f(K) \leq s)$$

Uncertainty quantification (UQ) for CCS

The simulator maps from permeability field K to outputs such as the surface flux $\mathcal{S}$. Let $f(K)$ denote this mapping

$$f : K \to \mathcal{S}$$

For most problems the permeability K is unknown.

If we assume a distribution for $K \sim \pi(K)$, we can quantify our uncertainty about $\mathcal{S} = f(K)$.

- e.g., by finding the cumulative distribution function (CDF) of $\mathcal{S}$:

$$F(s) = \mathbb{P}(f(K) \leq s)$$

We use a log-Gaussian process model for K

$$\log K(\cdot) \sim GP(m(\cdot), c(\cdot, \cdot))$$

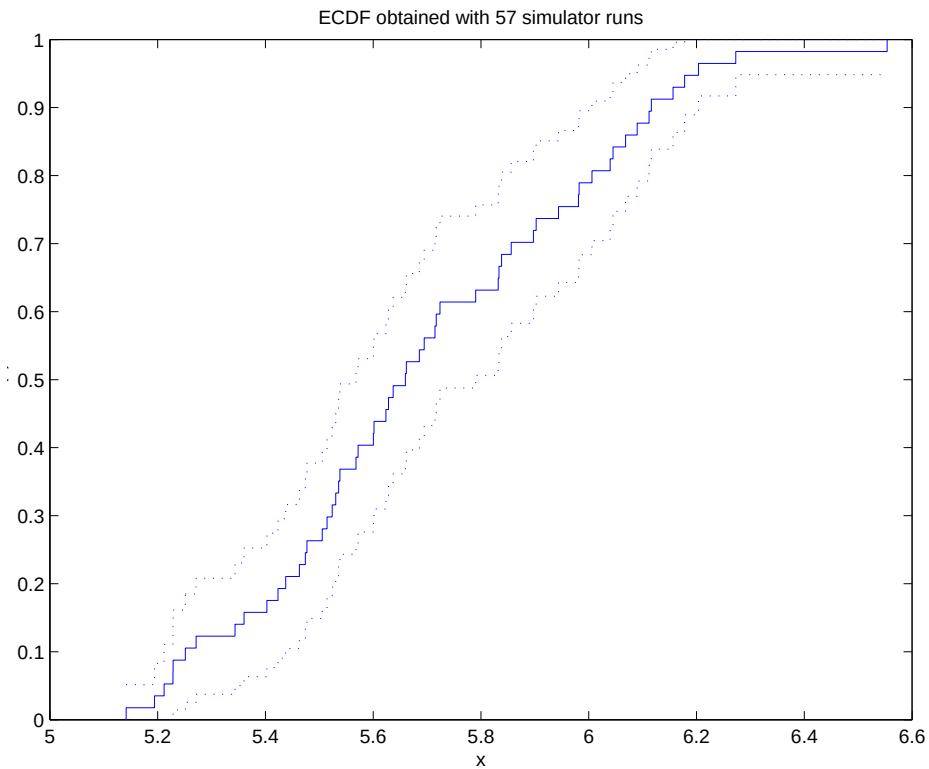
where $K(x)$ is the permeability at location x, and $m(\cdot)$ and $c(\cdot, \cdot)$ are the mean and covariance function of the GP (c is an exponential covariance function in the examples here).

UQ for complex computer models

Gold standard approach: Monte Carlo simulation

- Draw $K_1, \ldots, K_N \sim \pi(K)$, and evaluate the simulator at each giving fluxes
  $s_1 = f(K_1), \ldots, s_N = f(K_N)$
- Estimate the empirical CDF

$$\widehat{F}(s) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}_{s_i \leq s}$$

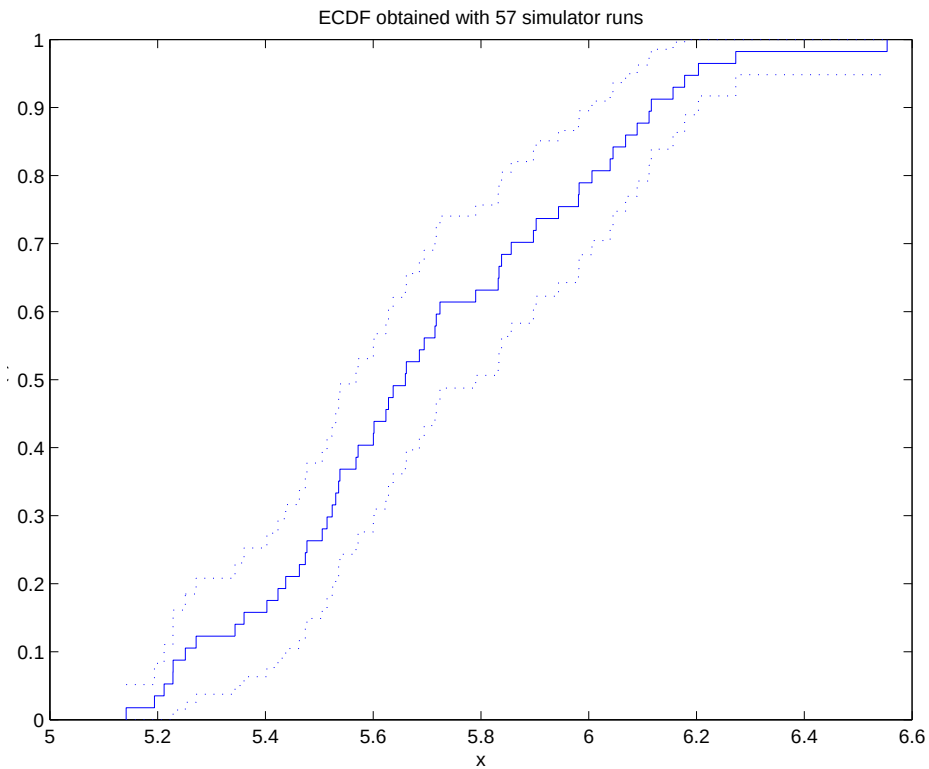


UQ for complex computer models

Gold standard approach: Monte Carlo simulation

- Draw $K_1, \ldots, K_N \sim \pi(K)$, and evaluate the simulator at each giving fluxes
  $s_1 = f(K_1), \ldots, s_N = f(K_N)$
- Estimate the empirical CDF

$$\widehat{F}(s) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}_{s_i \leq s}$$



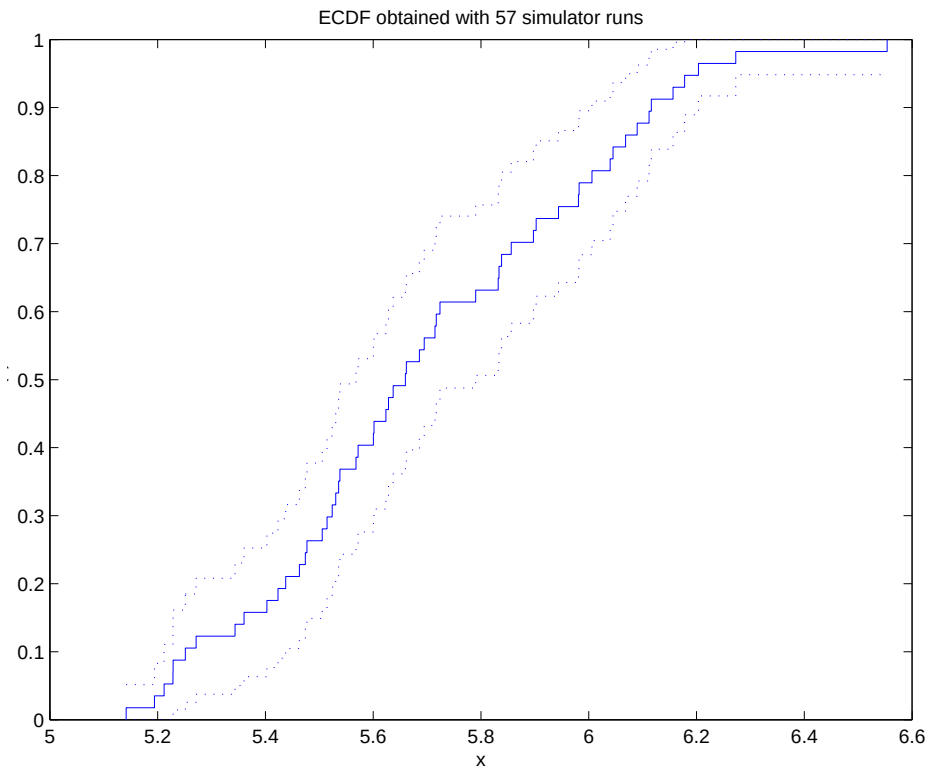
Note that $N = 10^3$ is not large if we want quantiles in the tail of the distribution

UQ for complex computer models

Gold standard approach: Monte Carlo simulation

- Draw $K_1, \ldots, K_N \sim \pi(K)$, and evaluate the simulator at each giving fluxes
  $s_1 = f(K_1), \ldots, s_N = f(K_N)$
- Estimate the empirical CDF

$$\widehat{F}(s) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}_{s_i \leq s}$$



Note that $N = 10^3$ is not large if we want quantiles in the tail of the distribution

What can we do if f is expensive to evaluate?

Gaussian Process emulation

Consider a 1d problem $y = f(x)$ and suppose we can only afford to evaluate the simulator a small number of times

$$\mathcal{D} = \{x_i, y_i = f(x_i)\}$$

We must make any inference about the simulator using $\mathcal{D}$ only.

Gaussian Process emulation

Consider a 1d problem $y = f(x)$ and suppose we can only afford to evaluate the simulator a small number of times

$$\mathcal{D} = \{x_i, y_i = f(x_i)\}$$

We must make any inference about the simulator using $\mathcal{D}$ only.

Build a meta-model/surrogate/*emulator*/reduced-order model for f.

- Try to find $\eta(x)$ such that

$$\eta(x) \approx f(x) \quad \forall \quad x \in I \subset \mathbb{R}$$

Gaussian Process emulation

Consider a 1d problem $y = f(x)$ and suppose we can only afford to evaluate the simulator a small number of times

$$\mathcal{D} = \{x_i, y_i = f(x_i)\}$$

We must make any inference about the simulator using $\mathcal{D}$ only.

Build a meta-model/surrogate/*emulator*/reduced-order model for f.

- Try to find $\eta(x)$ such that

$$\eta(x) \approx f(x) \quad \forall \quad x \in I \subset \mathbb{R}$$

We can use Gaussian processes (GP) to model $f(\cdot)$.

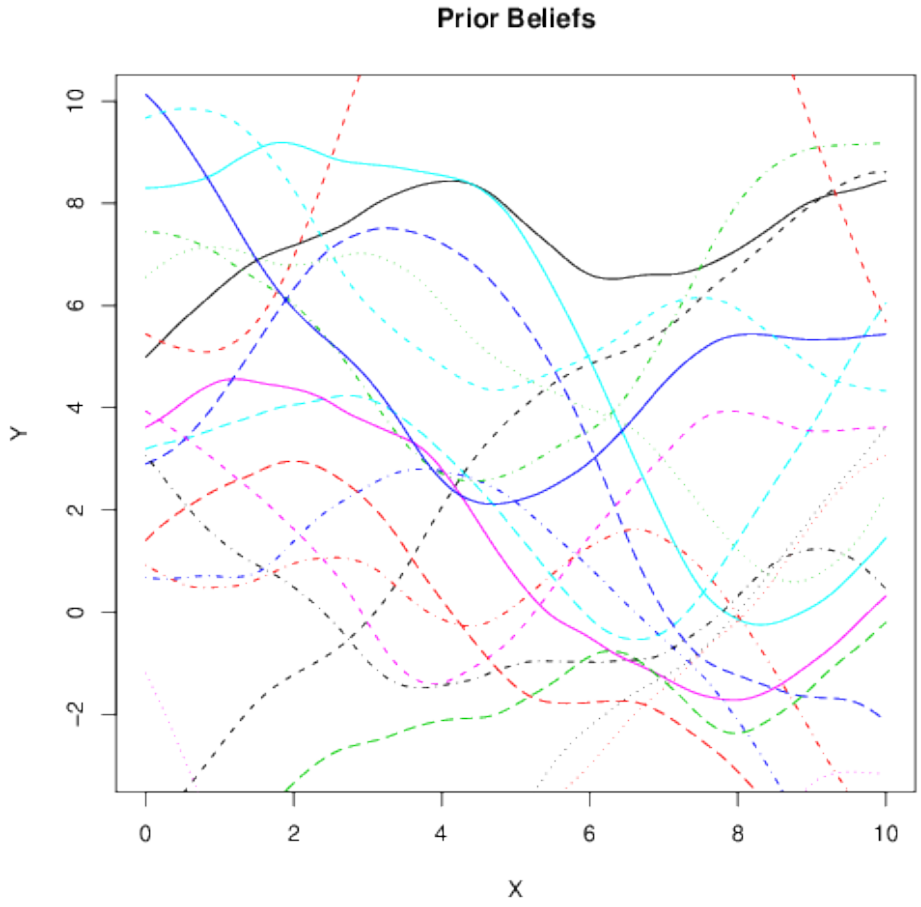
- $\eta(\cdot) \sim GP(m(\cdot), c(\cdot, \cdot))$
- We update our beliefs about η in light of the data $\mathcal{D}$,

$$\eta(\cdot)|\mathcal{D} \sim GP(m^*(\cdot), c^*(\cdot, \cdot))$$

- Note that $\eta(x)$ is a random value.

Gaussian Process prior for unknown functions

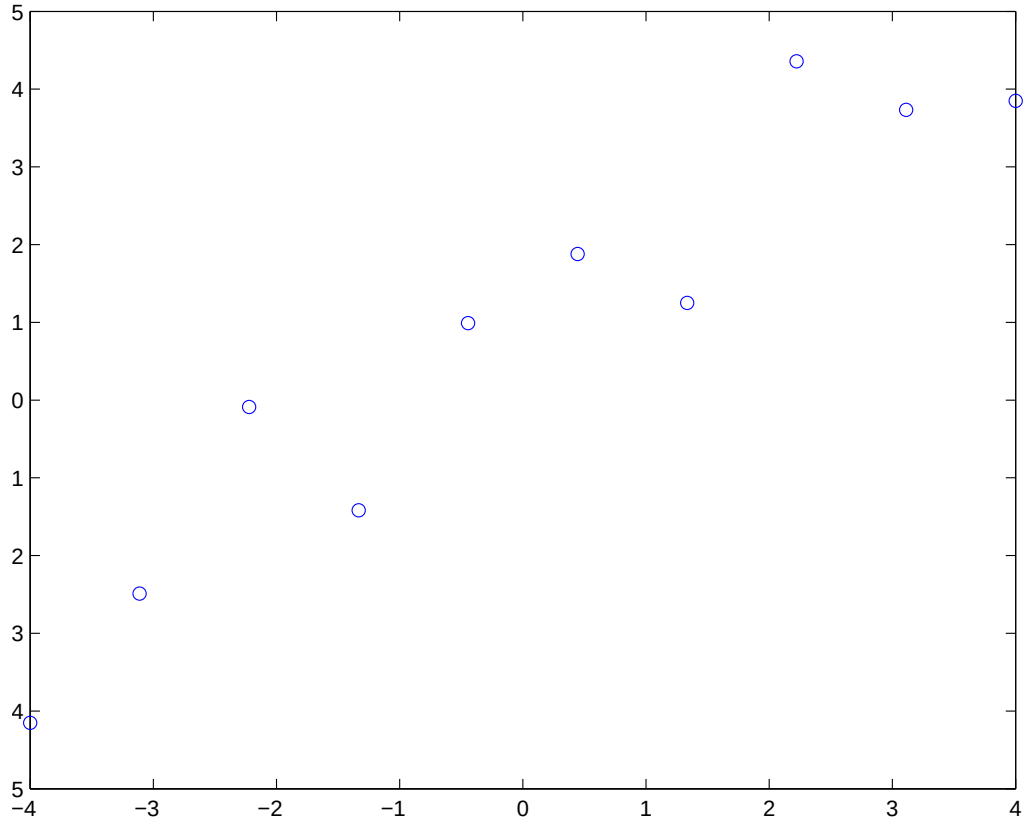
Prior belief about f



GPs can be understood as prior distributions over functions. Their properties, such as the smoothness and differentiability are controlled by the choice of mean and covariance functions, and the hyper-parameters.

Gaussian Process prior for unknown functions

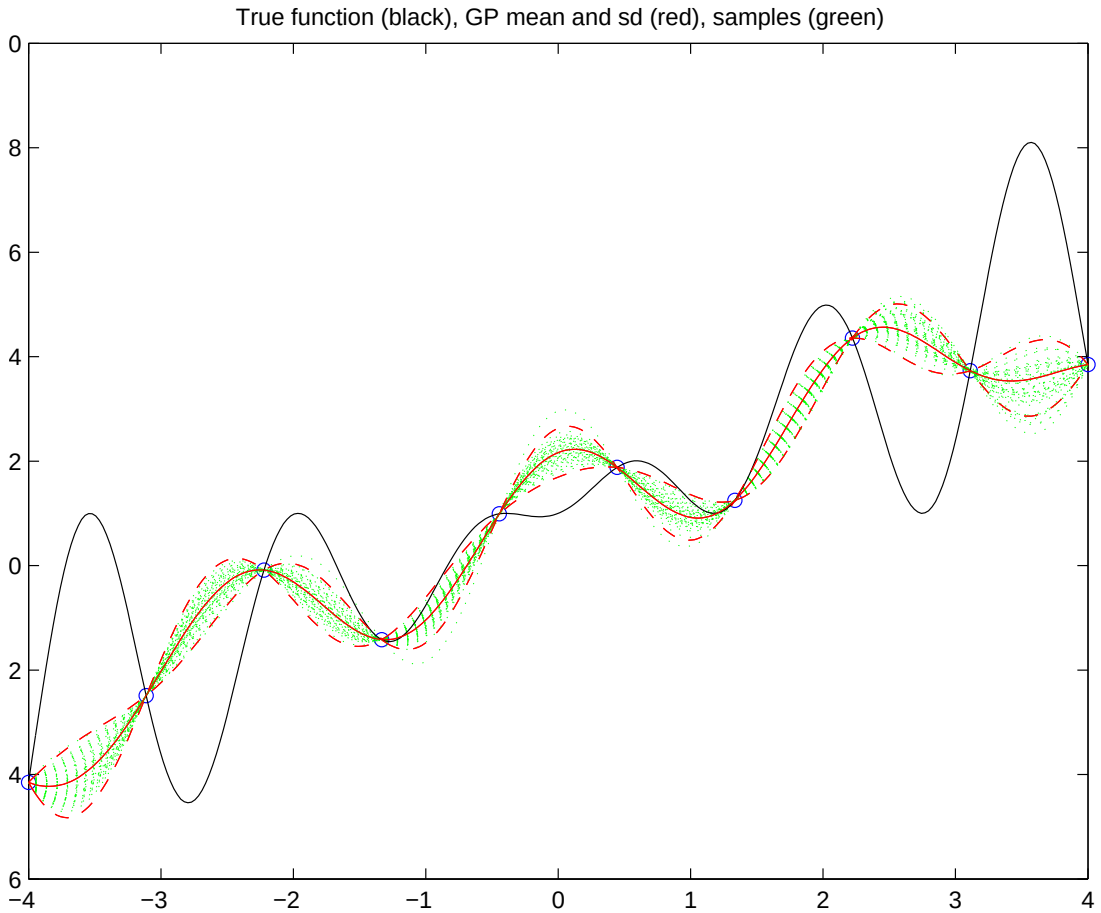
$y = f(x) = 1 + x + x\sin(4x)$ - 10 data points



Once we observe the data $D = \{(x_i, y_i)\}$, we can update our prior belief about the unknown function $f(x)$

Gaussian Process emulation - posterior beliefs about $f(\cdot)$

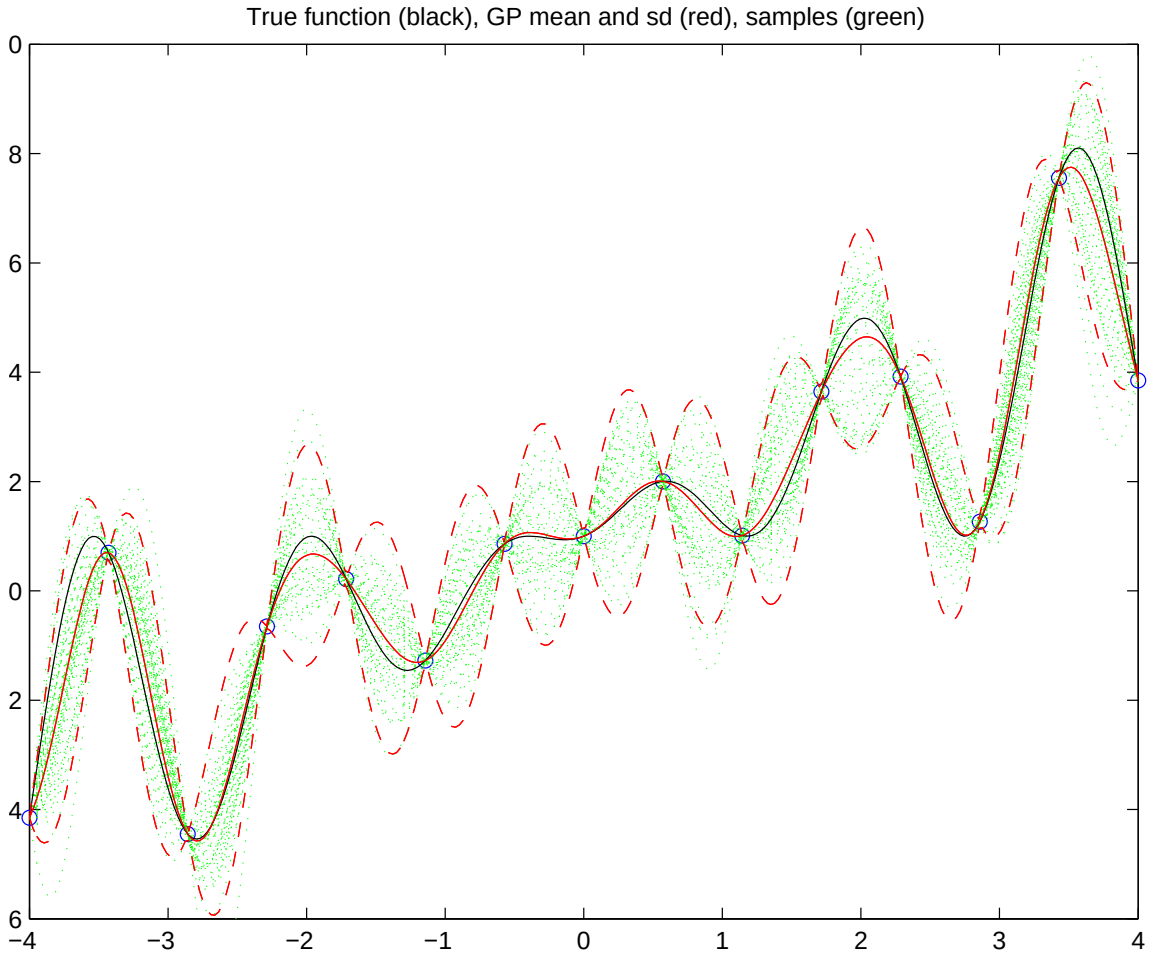
$y = 1 + x + x\sin(4x)$ - 10 data points



Perverse example: we can spot errors using cross-validation $\rightarrow$ More data required.

Gaussian Process emulation

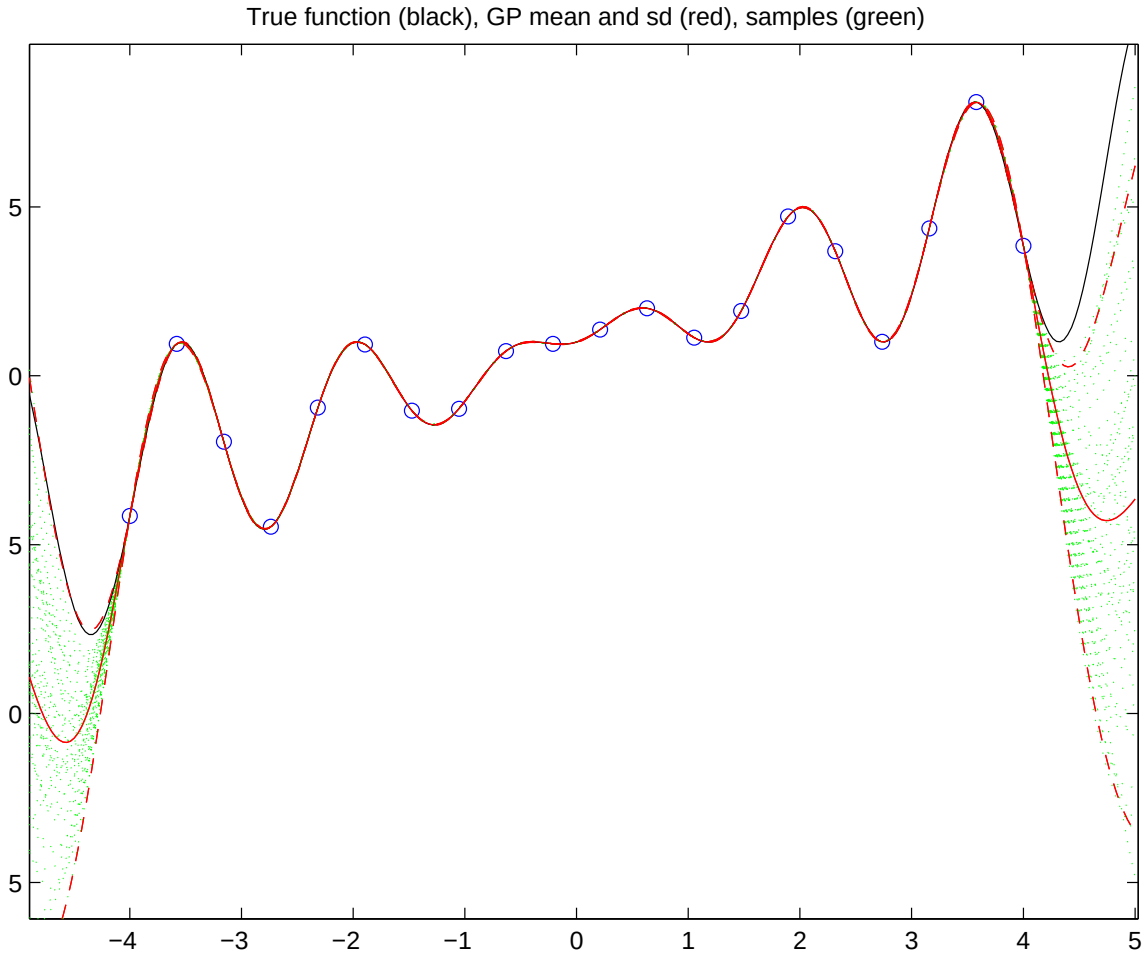
$y = 1 + x + x\sin(4x)$ - 15 data points



The covariance function is key. There are a small number of common choices, e.g., squared exponential (RBF/Gaussian), Matern, neural-net

Gaussian Process emulation

$y = 1 + x + x\sin(4x)$ - 20 data points



We can add, multiply and transform any covariance function to obtain a new valid covariance function.

Emulating simulators with high dimensional input

For the CCS simulator, the input is a permeability field which only needs to be known at a finite but large number of locations,

- e.g. if we use a 100×100 grid in the solver, K contains 10^4 entries
- Impossible to directly model $f : \mathbb{R}^{10,000} \to \mathbb{R}$

Emulating simulators with high dimensional input

For the CCS simulator, the input is a permeability field which only needs to be known at a finite but large number of locations,

- e.g. if we use a 100×100 grid in the solver, K contains 10^4 entries
- Impossible to directly model $f : \mathbb{R}^{10,000} \to \mathbb{R}$

Instead, we can use the Karhunen-Loève (KL) expansion of K to reduce the dimension:

- $K = \exp(Z)$ where $Z \sim GP(m, C)$
- Z can be represented as

$$Z(\cdot) = \sum_{i=1}^{\infty} \lambda_i \xi_i \phi_i(\cdot)$$

where λ_i and ϕ_i are the eigenvalues and eigenfunctions of the covariance function of Z and $\xi_i \sim N(0, 1)$.

Emulating simulators with high dimensional input

By truncating

$$K(x) \approx \exp\left(\sum_{i=1}^{n} \lambda_i \xi_i \phi_i(x)\right)$$

we reduce the modelling problem to one of modelling

$$f : \mathbb{R}^n \to \mathbb{R}$$

- This involves some loss of information
  - Add a nugget term to the GP to represent the missing information

Emulating simulators with high dimensional input

By truncating

$$K(x) \approx \exp\left(\sum_{i=1}^{n} \lambda_i \xi_i \phi_i(x)\right)$$

we reduce the modelling problem to one of modelling

$$f : \mathbb{R}^n \to \mathbb{R}$$

- This involves some loss of information
  - Add a nugget term to the GP to represent the missing information

Build a GP emulator from $\mathbf{x} = (\xi_1, \ldots, \xi_n)^\top$ to the surface flux (SF)

- We need a training set $(\mathbf{x}_i, SF_i)_{i=1}^N$ of simulator runs to build the emulator

Emulating simulators with high dimensional input

By truncating

$$K(x) \approx \exp\left(\sum_{i=1}^{n} \lambda_i \xi_i \phi_i(x)\right)$$

we reduce the modelling problem to one of modelling

$$f : \mathbb{R}^n \to \mathbb{R}$$

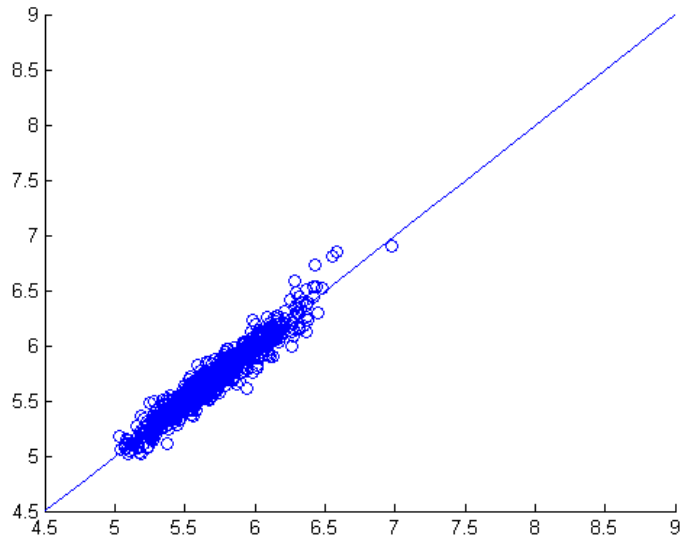
- This involves some loss of information
  - Add a nugget term to the GP to represent the missing information

Build a GP emulator from $\mathbf{x} = (\xi_1, \ldots, \xi_n)^\top$ to the surface flux (SF)

- We need a training set $(\mathbf{x}_i, SF_i)_{i=1}^N$ of simulator runs to build the emulator
- The design (choice of $\mathbf{x}$ locations) is key. Generally space-filling designs are recommended.
  - We use a Sobol sequence to find a space-filling design of N points on $[0,1]^n$
  - Spread the points by pushing them through the inverse CDF of a $N(0,1)$ distribution to get a design on $\mathbb{R}^n$ that can be used for $N(0,1)$ inputs.

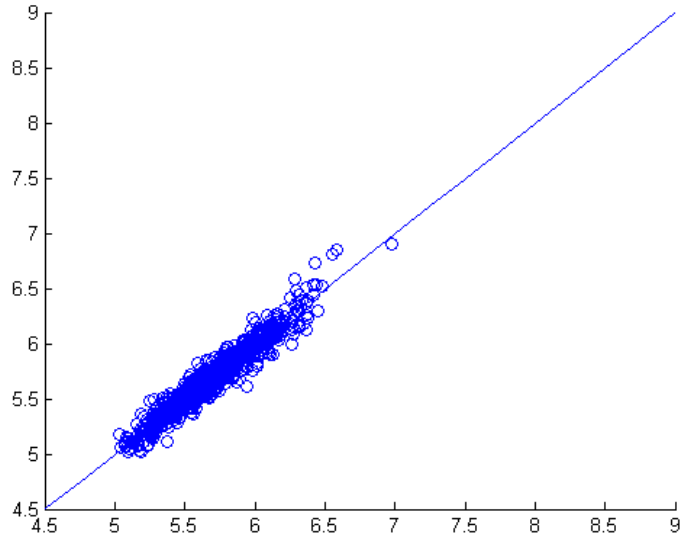
Predictive performance vs n = no. of KL components

We can assess the accuracy of the emulator by examining the prediction error on a held-out test set. Plotting predicted vs true value indicates the accuracy the GP emulator.

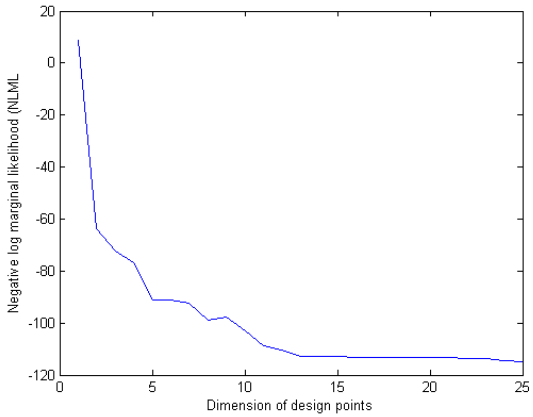


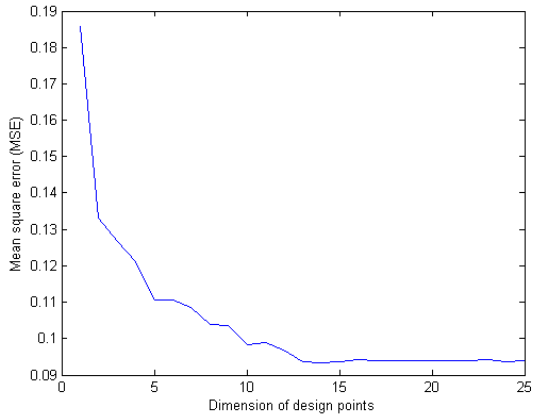
Predictive performance vs n = no. of KL components

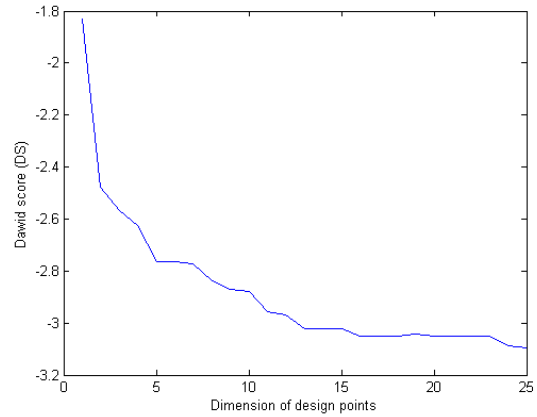
We can assess the accuracy of the emulator by examining the prediction error on a held-out test set. Plotting predicted vs true value indicates the accuracy the GP emulator.



We can also choose the number of KL components to retain using numerical scores





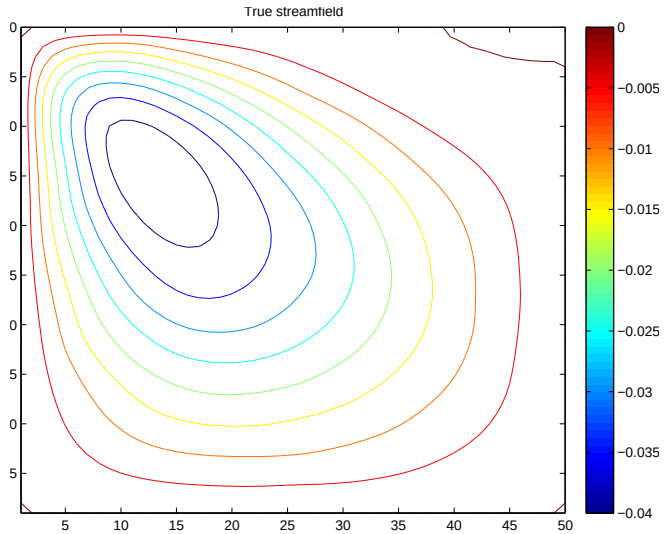


Emulating from fields to fields

W. 2011, Holden, Edwards, Garthwaite and W. in prep.

Now consider emulating the stream function and concentration fields (100×100 matrices).

We can use a similar trick, and use the singular value decomposition to reduce the dimension.



- Let $\mathbf{y}_1, \ldots, \mathbf{y}_N \in \mathbb{R}^d$ be the N fields obtained and let Y be the $d \times N$ matrix with column i being $\mathbf{y}_i$.
- Let $\tilde{Y}$ be the row centred version of Y,
- Form the SVD of $\tilde{Y}$: $\quad \tilde{Y} = LDR^T$

- We can form a reduced rank approximation to $\tilde{Y}$ by ignoring all but the first k eigenvectors:

$$L_* = (l_1, \ldots, l_k), \qquad R_* = (r_1, \ldots, r_k)$$

so that

$$\tilde{Y} \approx L_* D_* R_*^T$$

- If $R_*^T = (t_1, \ldots, t_N)$, where each t_i is a vector of length k, then

$$L_* D_* t_1 \approx \mathbf{y}_1$$

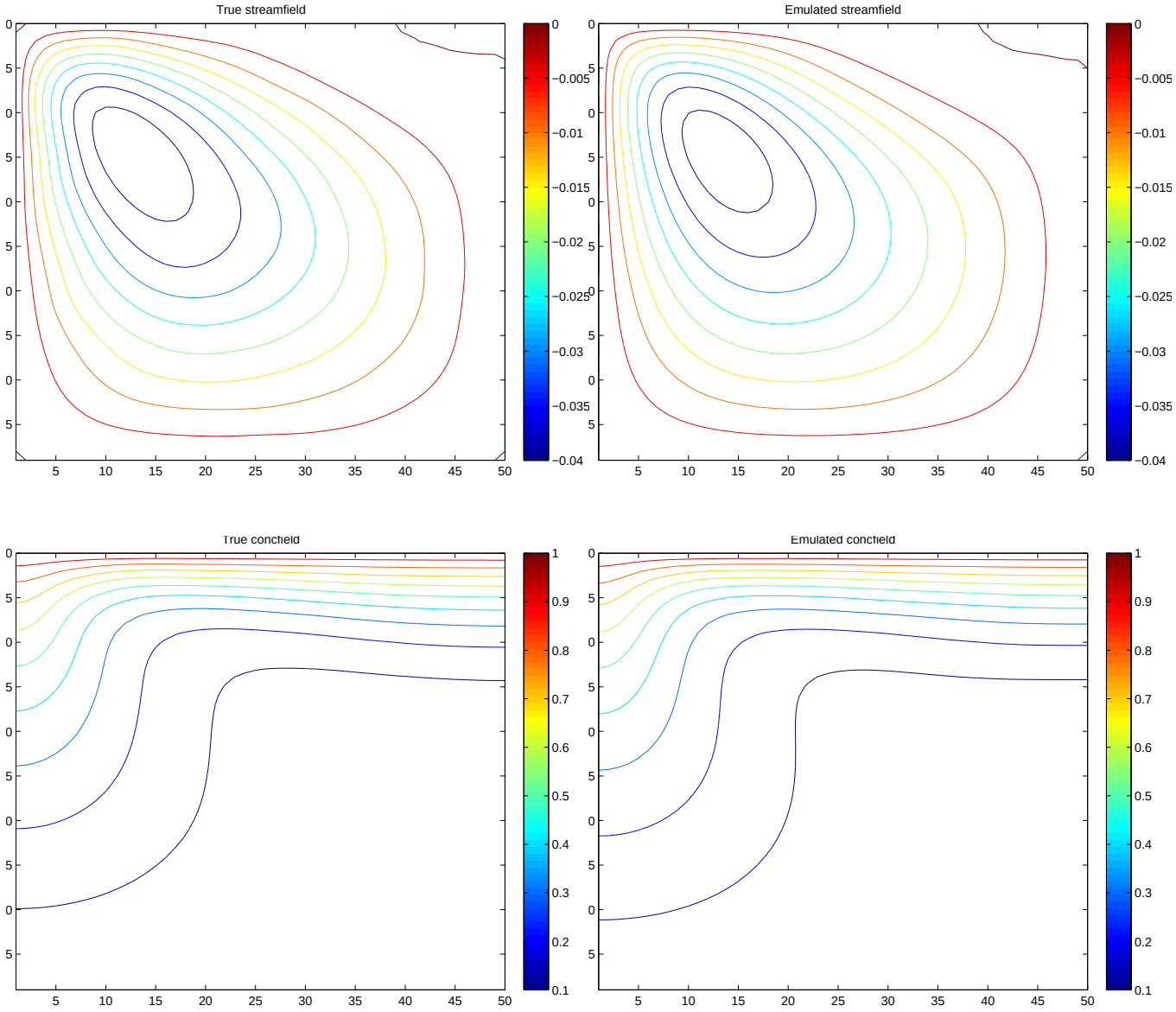
e.g., the centred concentration field for the 1st simulation.

To build an emulator from $\mathbf{x}$ to $\mathbf{y}$, we can build an emulator from $\mathbf{x}$ to the rows of R_* = columns of R_*^T.

To do this, we can build k separate emulators from $\mathbf{x}$ to each element in the vector t.

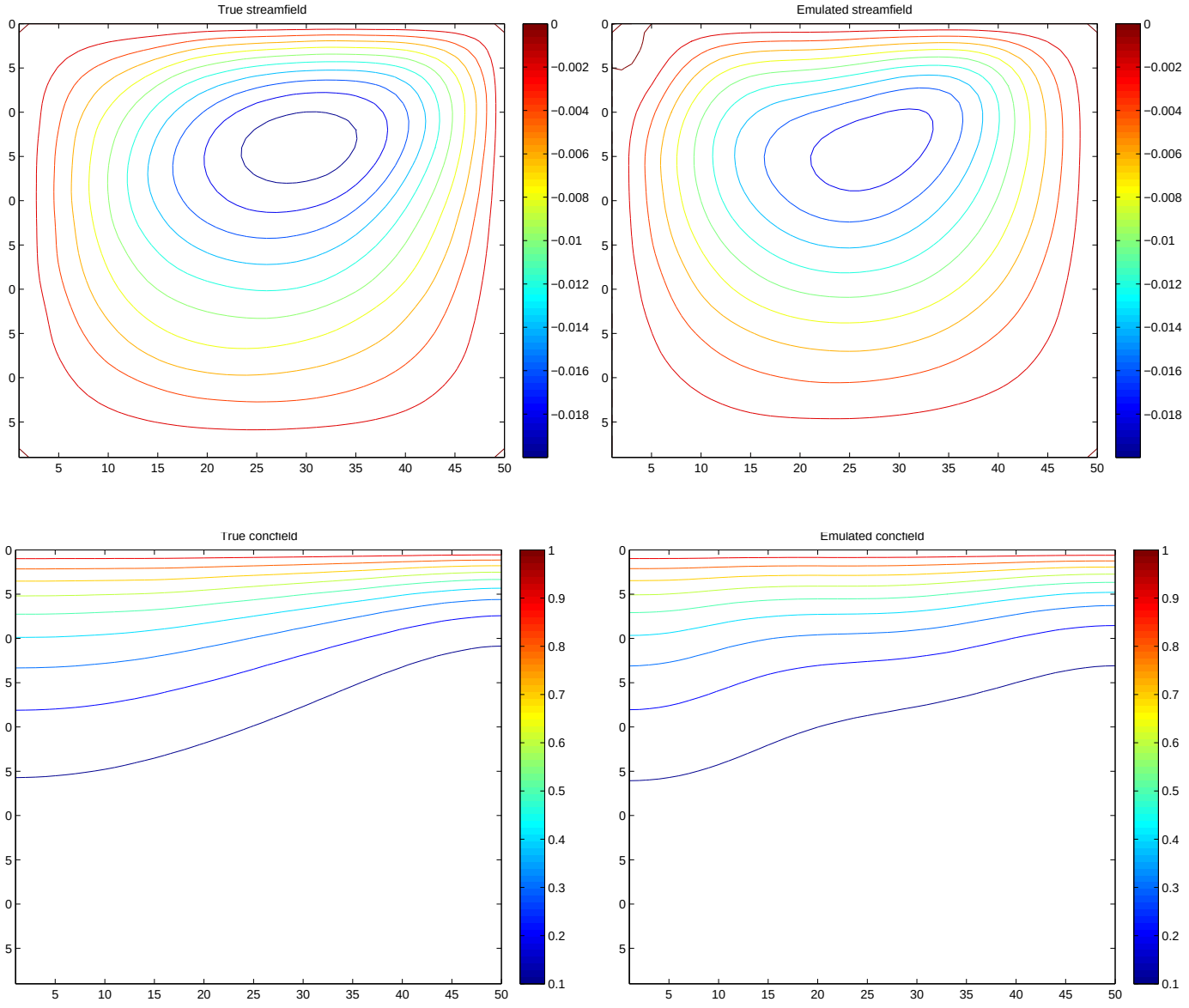
Emulating the stream function and concentration fields

Left=true, right = emulated, 118 training runs, held out test set.



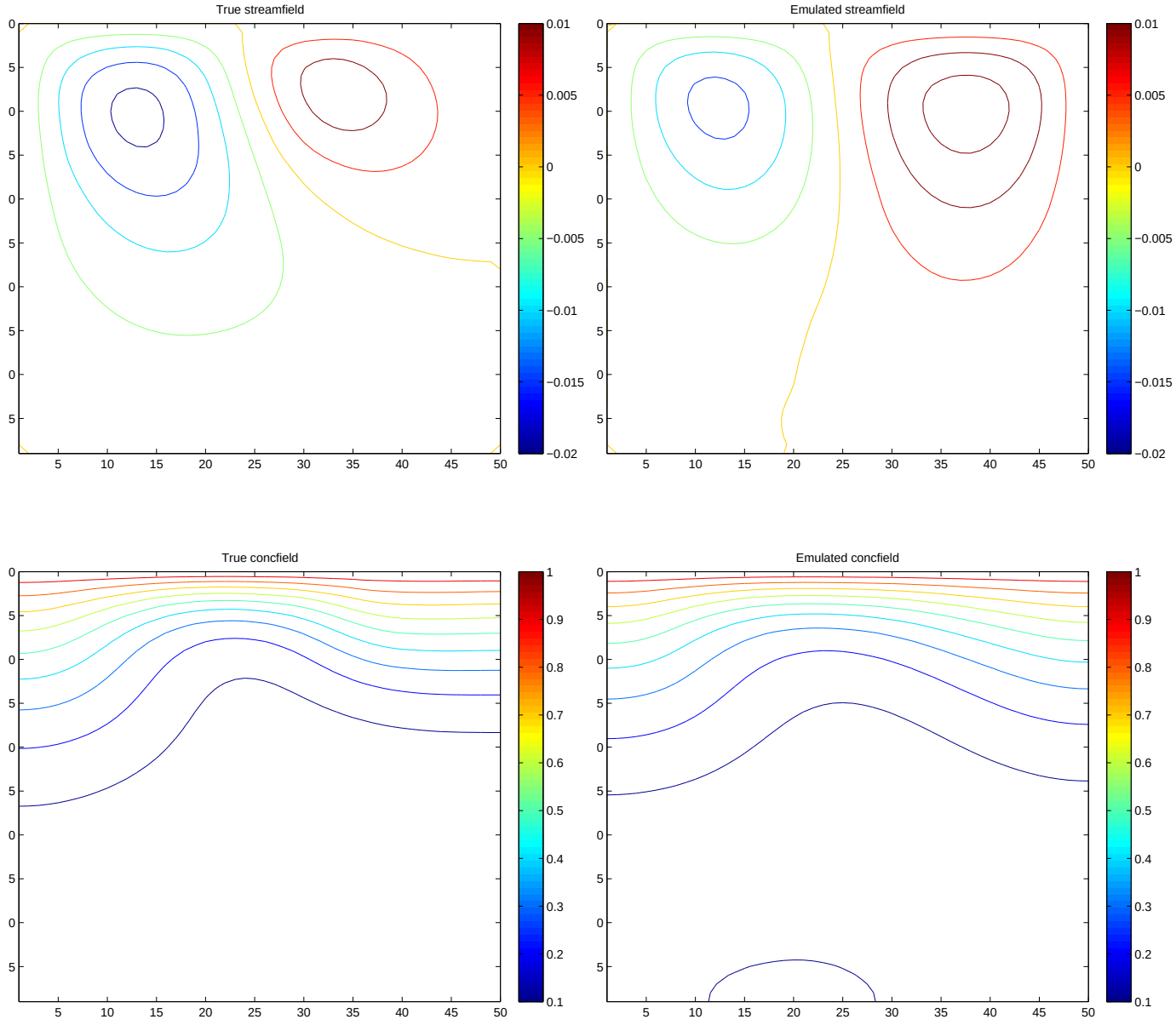
Emulating the stream function and concentration fields

Left=true, right = emulated, 118 training runs, held out test set.



Emulating the stream function and concentration fields

Left=true, right = emulated, 118 training runs, held out test set.



Finding CDFs

$\eta(x)$ is a random function approximating the simulator $f(x)$. Hence, any summary will be a random variable.

In particular, the CDF $F_\eta(s) = \mathbb{P}_X(\eta(X) \leq s)$ is a random variable.

Finding CDFs

$\eta(x)$ is a random function approximating the simulator $f(x)$. Hence, any summary will be a random variable.

In particular, the CDF $F_\eta(s) = \mathbb{P}_X(\eta(X) \leq s)$ is a random variable.

We can use Monte Carlo to evaluate the distribution of $F_\eta(s)$: for $j = 1, \ldots, M$:

- Draw $\eta_j(\cdot) \sim GP(m^*(\cdot), c^*(\cdot, \cdot))$
- Evaluate

$$\begin{aligned} \hat{F}_j(s) &= \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}_{\eta_j(x_i) \leq s} \\ &\approx \mathbb{P}(\eta_j(X) \leq s) \end{aligned}$$

where x_i are i.i.d. samples of X.

Finding CDFs

$\eta(x)$ is a random function approximating the simulator $f(x)$. Hence, any summary will be a random variable.

In particular, the CDF $F_\eta(s) = \mathbb{P}_X(\eta(X) \leq s)$ is a random variable.

We can use Monte Carlo to evaluate the distribution of $F_\eta(s)$: for $j = 1, \ldots, M$:

- Draw $\eta_j(\cdot) \sim GP(m^*(\cdot), c^*(\cdot, \cdot))$
- Evaluate

$$\begin{aligned}\hat{F}_j(s) &= \frac{1}{m}\sum_{i=1}^{m} \mathbb{I}_{\eta_j(x_i)\leq s} \\ &\approx \mathbb{P}(\eta_j(X) \leq s)\end{aligned}$$

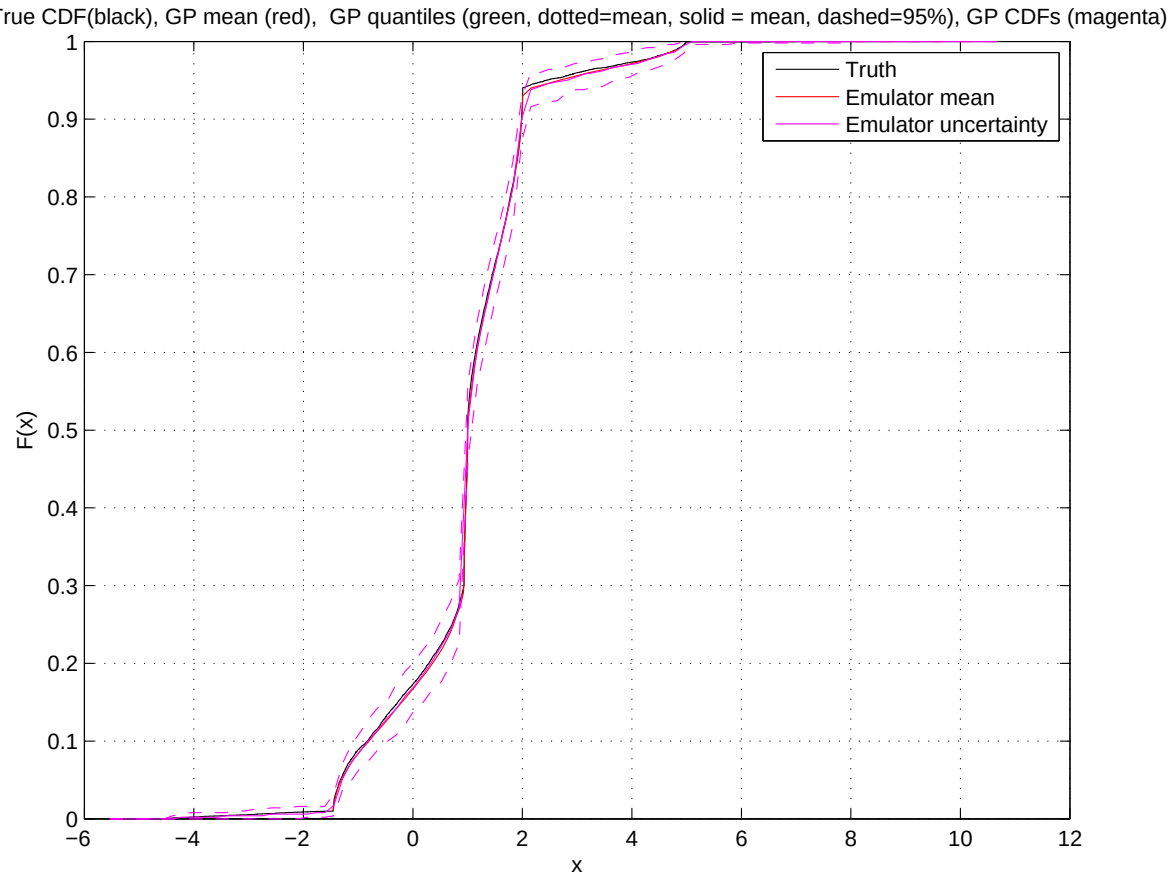
  where x_i are i.i.d. samples of X.

This gives a Monte Carlo sample of distribution functions

$$\hat{F}_1(\cdot), \ldots, \hat{F}_M(\cdot)$$

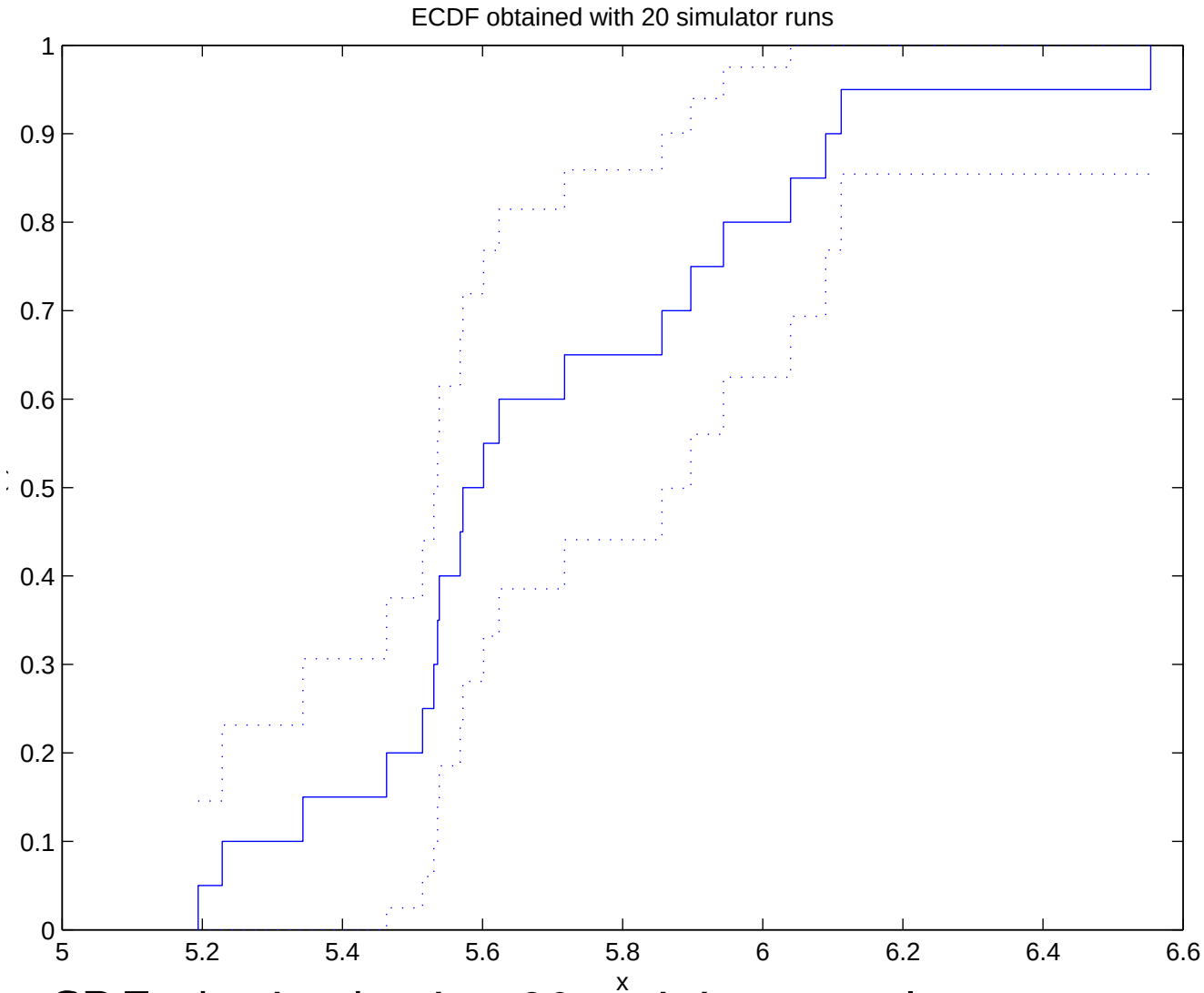
From these, we can estimate the median CDF and any confidence intervals we require.

1d example, 20 data points



We can give the median (mean estimate is skewed in the tails because $0 \leq F \leq 1$), and a 95% confidence interval for the unknown CDF.

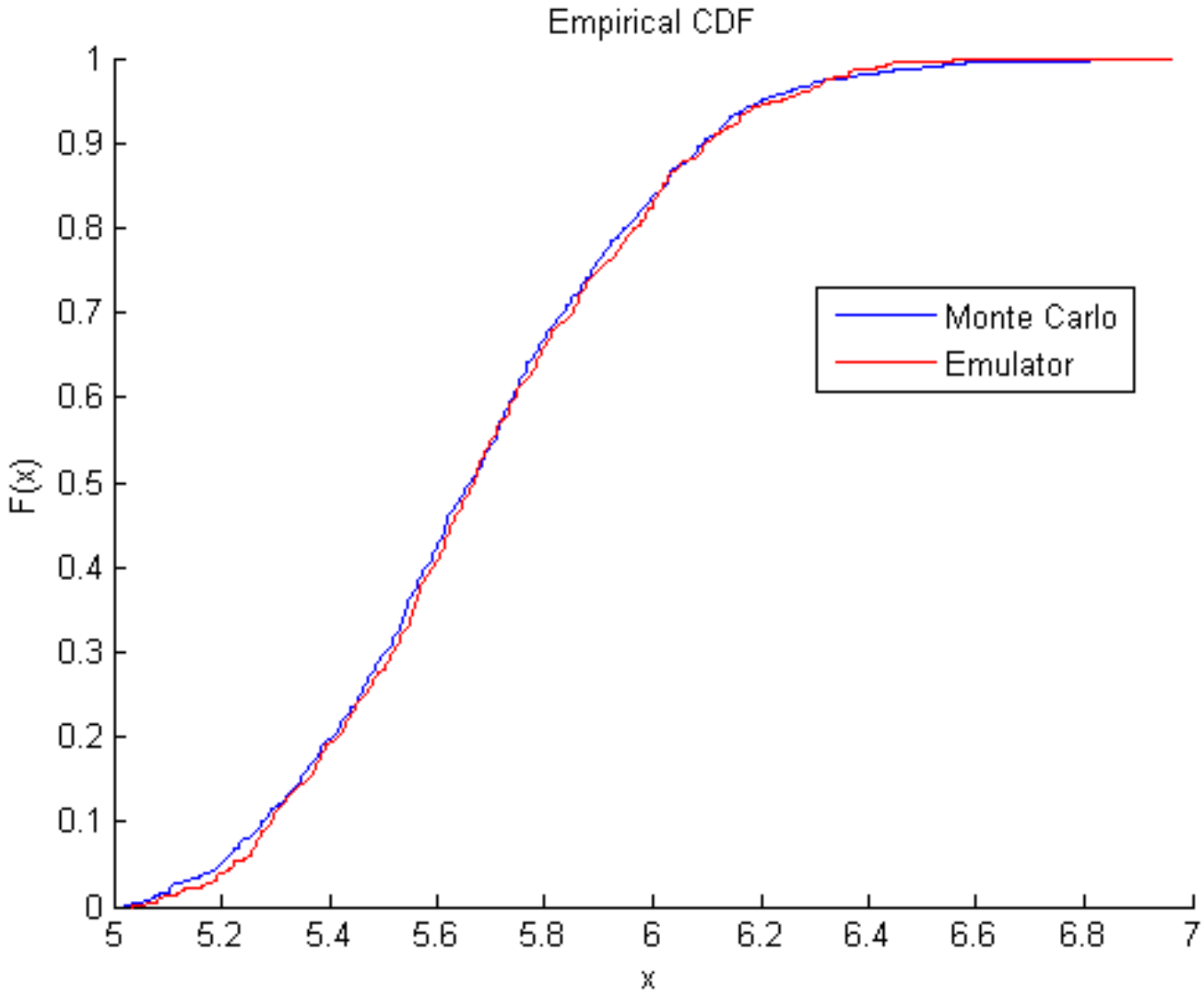
CCS simulator results - 20 simulator training runs



Blue line = CDF obtained using 20 training samples

dotted line = 95% confidence interval

CCS simulator results - 20 simulator training runs



Blue line = CDF from using 10^3 Monte Carlo samples from the simulator
Red line = CDF obtained using emulator (trained with 20 simulator runs, rational quadratic covariance function)

Limitations & future work

This describes an approach for doing uncertainty quantification on the simulator output rather than the physical system.

Limitations & future work

This describes an approach for doing uncertainty quantification on the simulator output rather than the physical system.

Bifurcations cause non-continuous behaviour in the simulator response.

- If this happens, the simulator can't easily be modelled.
- No easy way to assess if this has happened in any given simulation.

Limitations & future work

This describes an approach for doing uncertainty quantification on the simulator output rather than the physical system.

Bifurcations cause non-continuous behaviour in the simulator response.

- If this happens, the simulator can't easily be modelled.
- No easy way to assess if this has happened in any given simulation.

Future work:

- Uncertainty on the emulator prediction
- Uncertainty in the hyper-parameters
- Multi-level Monte Carlo
- Two-stage emulation
- Polynomial Chaos.

Limitations & future work

This describes an approach for doing uncertainty quantification on the simulator output rather than the physical system.

Bifurcations cause non-continuous behaviour in the simulator response.

- If this happens, the simulator can't easily be modelled.
- No easy way to assess if this has happened in any given simulation.

Future work:

- Uncertainty on the emulator prediction
- Uncertainty in the hyper-parameters
- Multi-level Monte Carlo
- Two-stage emulation
- Polynomial Chaos.

Thank you for listening!