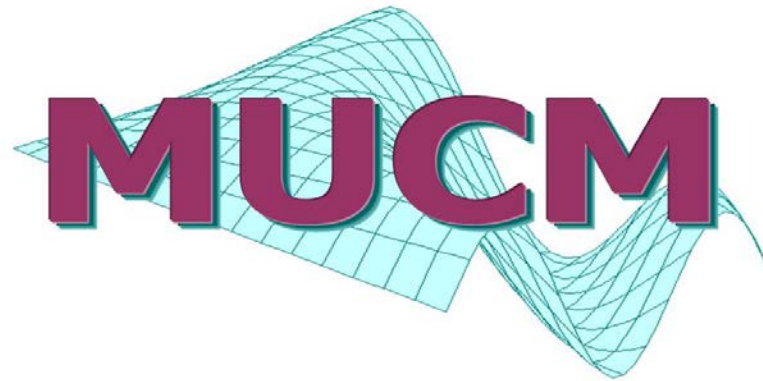


Emulators and MUCM



Outline

- ▲ Background
 - ▲ Simulators
- ▲ Uncertainty in model inputs
 - ▲ Uncertainty analysis
 - ▲ Case study – dynamic vegetation simulator
- ▲ Emulators
 - ▲ Computation
 - ▲ MUCM

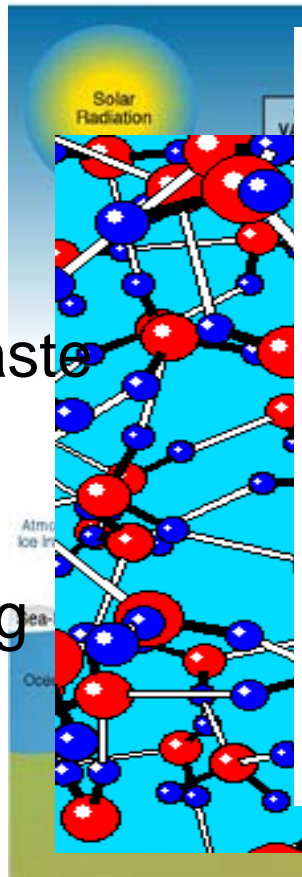
Background

Simulators

- ▲ In almost all fields of science, technology, industry and policy making, people use mechanistic models
 - ▲ For understanding, prediction, control
- ▲ A model simulates a real-world, usually complex, phenomenon as a set of mathematical equations
- ▲ Models are usually implemented as computer programs
 - ▲ We will refer to a computer implementation of a model as a *simulator*

Examples

- ▲ Climate prediction
- ▲ Molecular dynamics
- ▲ Nuclear waste disposal
- ▲ Oil fields
- ▲ Engineering design
- ▲ Hydrology



Hydrologic Cycle in Catchment



The simulator as a function

- ▲ Using computer language, a simulator takes a number of inputs and produces a number of outputs
- ▲ We can represent any output y as a function
$$y = f(x)$$
of a vector x of inputs

Why worry about uncertainty?

- ▲ How accurate are model predictions?
- ▲ There is increasing concern about uncertainty in model outputs
 - ▲ Particularly where simulator predictions are used to inform scientific debate or environmental policy
 - ▲ Are their predictions robust enough for high stakes decision-making?

For instance ...

- ▲ Models for climate change produce different predictions for the extent of global warming or other consequences
 - ▲ Which ones should we believe?
 - ▲ What error bounds should we put around these?
 - ▲ Are simulator differences consistent with the error bounds?
- ▲ Until we can answer such questions convincingly, why should anyone have faith in the science?

Where is the uncertainty?

- ▲ How might the simulator output $y = f(x)$ differ from the true real-world value z that the simulator is supposed to predict?
 - ▲ Error in inputs x
 - ▲ Initial values
 - ▲ Forcing inputs
 - ▲ Model parameters
 - ▲ Error in model structure or solution
 - ▲ Wrong, inaccurate or incomplete science
 - ▲ Bugs, solution errors

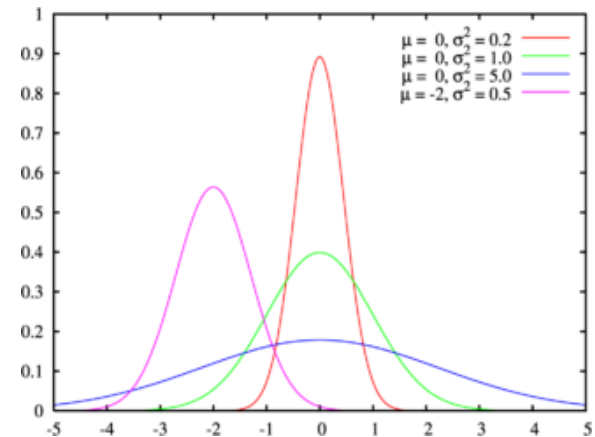
Quantifying uncertainty

- ▲ The ideal is to provide a probability distribution $p(z)$ for the true real-world value

- ▲ The centre of the distribution is a best estimate
- ▲ Its spread shows how much uncertainty about z is induced by uncertainties on the previous slide

- ▲ How do we get this?

- ▲ Input uncertainty: characterise $p(x)$, propagate through to $p(y)$
- ▲ Structural uncertainty: characterise $p(z-y)$



Uncertainty in model inputs

Inputs

- ▲ We will interpret “inputs” widely
 - ▲ Initial conditions
 - ▲ Other data defining the particular context being simulated
 - ▲ Exogenous (“forcing”) data
 - ▲ Parameters in model equations
 - ▲ Often hard-wired (which is a problem!)

The CENTURY model

For example, 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

- ▲ 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

- ▲ 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

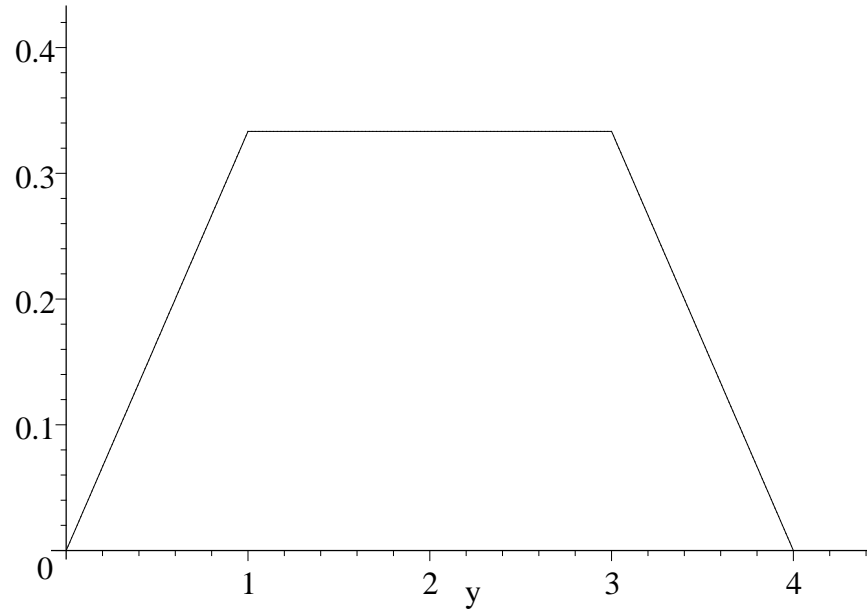
A trivial model

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

$$y = x_1 + 3x_2$$

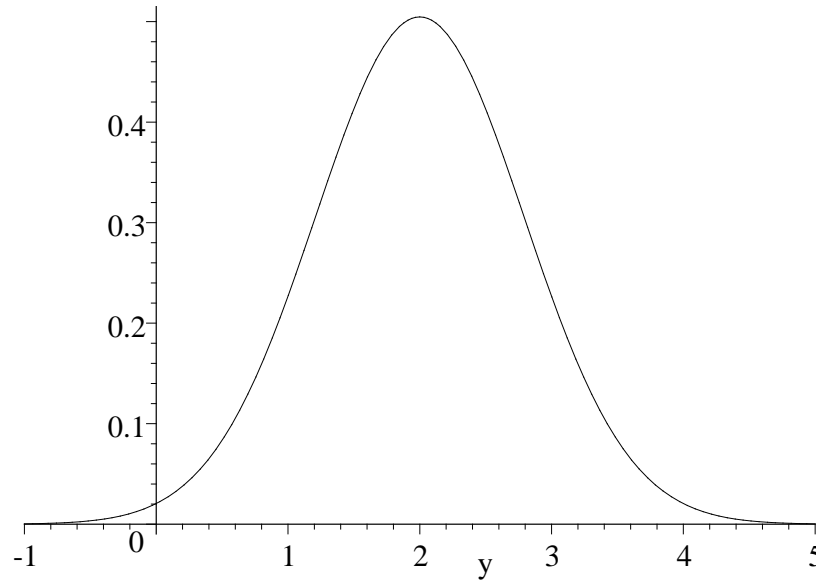
- ▲ Suppose that x_1 and x_2 have independent uniform distributions over $[0, 1]$
 - ▲ i.e. they define a point that is equally likely to be anywhere in the unit square
- ▲ Then we can determine the distribution of y exactly

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

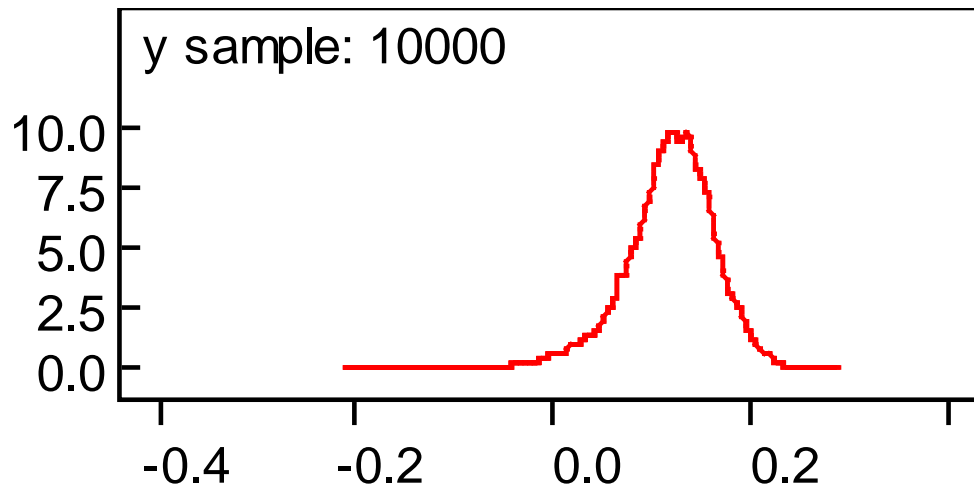
A slightly less trivial model

- ▲ 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

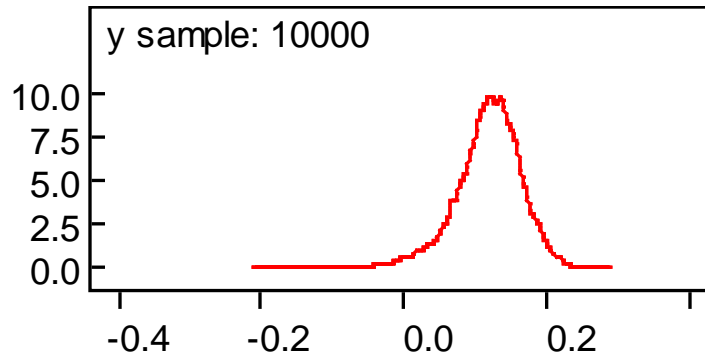


- ▲ This is for the normal inputs
- ▲ 10,000 random normal pairs were generated and y calculated for each pair

Uncertainty analysis (UA)

- ▲ 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 of slightly nonlinear model



- ▲ Mean = 0.117, median = 0.122
- ▲ Std. dev. = 0.048
- ▲ 50% range (quartiles) = [0.092, 0.148]
- ▲ 95% range = [0.004, 0.200]

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

Summary

Why UA?

- ▲ Proper quantification of output uncertainty
 - ▲ Need proper probabilistic expression of input uncertainty
- ▲ Improved central estimate of output
 - ▲ Better than the usual plug-in approach

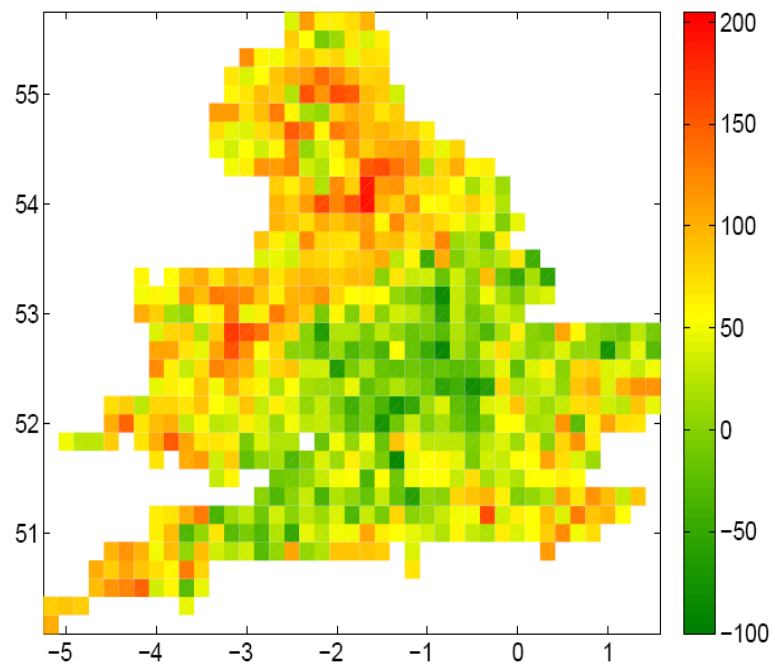
A real case study

Example: UK carbon flux in 2000

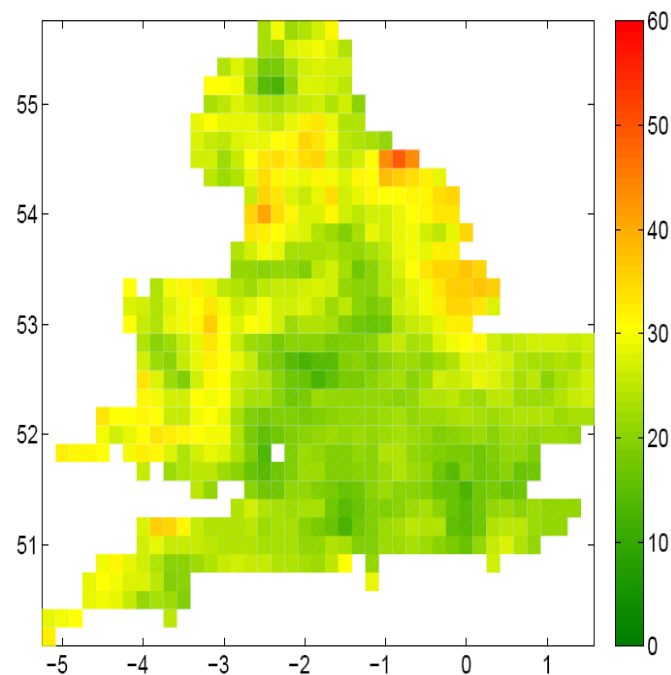
- ▲ Vegetation model predicts carbon exchange from each of 700 pixels over England & Wales in 2000
 - ▲ Principal output is Net Biosphere Production
- ▲ Accounting for uncertainty in inputs
 - ▲ Soil properties
 - ▲ Properties of different types of vegetation
 - ▲ Land usage
 - ▲ (Not structural uncertainty)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

- ▲ 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

Emulators

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

- ▲ 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

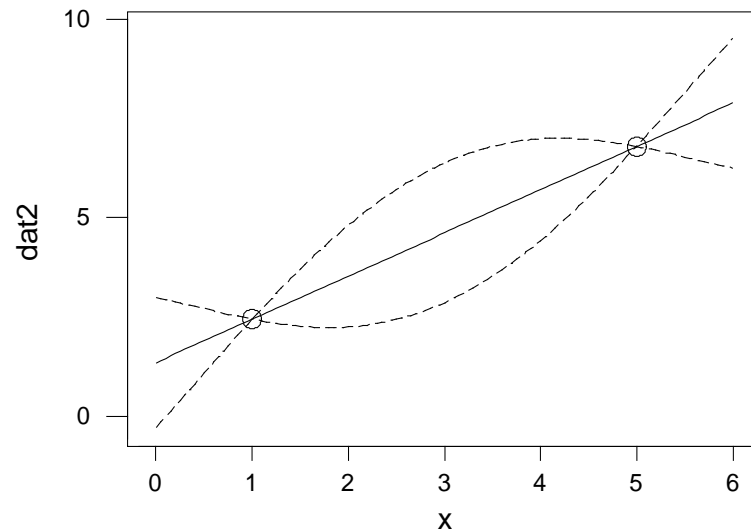
- ▲ More efficient approach
 - ▲ First work in early 1980s (DACE)
- ▲ Represent the code as an unknown function
 - ▲ $f(\cdot)$ becomes a random process
 - ▲ We generally represent it as a Gaussian process (GP)
 - ▲ Or its second-order moment version
- ▲ Training runs
 - ▲ Run simulator for sample of x values
 - ▲ Condition GP on observed data
 - ▲ Typically requires many fewer runs than Monte Carlo
 - ▲ And 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 \mathbf{x} (prediction)
 - ▲ With uncertainty about that prediction given by posterior variance
 - ▲ Correctly reproduces training data

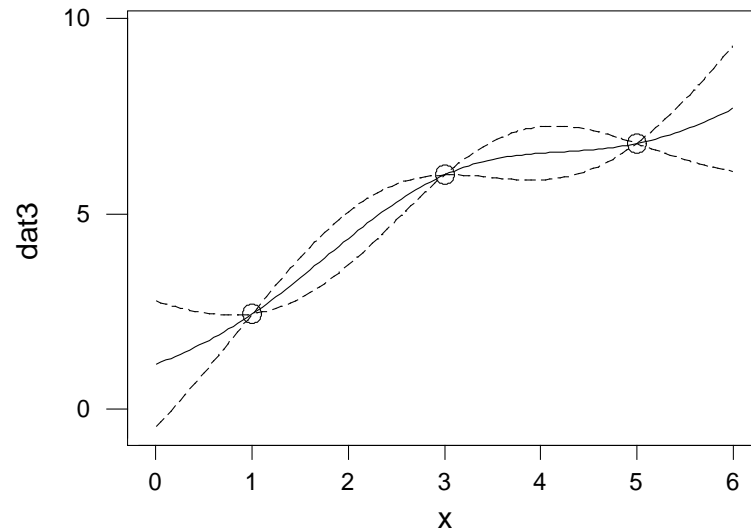
2 code runs

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



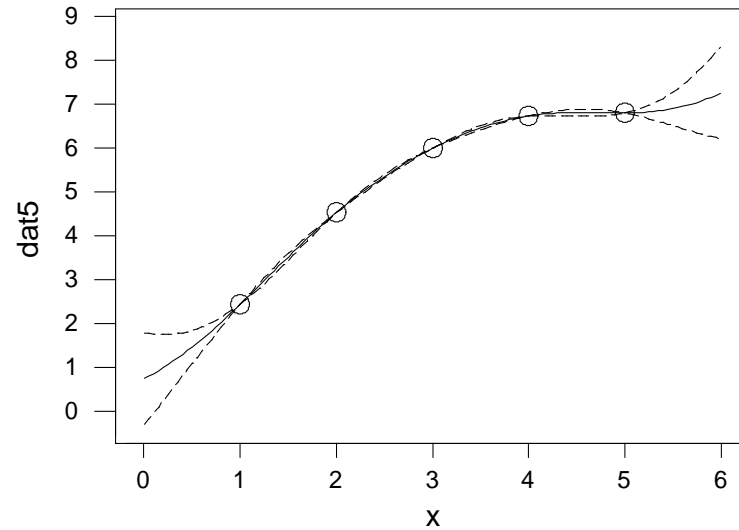
3 code runs

- ▲ Adding another point changes estimate and reduces uncertainty



5 code runs

▲ And so on



Then what?

- ▲ 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

BACCO

- ▲ This has led to a wide ranging body of tools for all kinds of tasks involving uncertainties in computer models
 - ▲ Uncertainty analysis, sensitivity analysis, calibration, data assimilation, validation, optimisation etc.
 - ▲ All in a single coherent framework
- ▲ All based on building an *emulator* of the simulator from a set of training runs
- ▲ This area is sometimes known as BACCO
 - ▲ Bayesian Analysis of Computer Code Output

MUCM

- ▲ Managing Uncertainty in Complex Models
 - ▲ 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

- ▲ 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

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