



Simulators and Emulators

Tony O'Hagan
University of Sheffield

Overview

- ▶ **Part 1 – Introduction**
 - ▶ Simulators
 - ▶ Emulators
 - ▶ Examples
 - ▶ Resources

- ▶ **Part 2 – Validation and dynamic emulation**
 - ▶ Validating emulators and simulators
 - ▶ Emulating dynamic simulators

Part One - Introduction

Simulators

- ▶ In almost all fields of science, technology, industry and policy making, people use mechanistic models to simulate complex real-world processes
 - ▶ For understanding, prediction, control
- ▶ Usually implemented in computer codes
 - ▶ Often very computationally intensive
 - ▶ We'll call them simulators
- ▶ There is a growing realisation of the importance of uncertainty in simulator predictions
 - ▶ Can we trust them?
 - ▶ Without any quantification of output uncertainty, it's easy to dismiss them

Examples

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

Hydrologic Cycle in Catchment



Sources of uncertainty

- ▶ A simulator takes inputs x and produces outputs $y = f(x)$
- ▶ How might y 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 last slide
- ▶ How do we get this?
 - ▶ Input uncertainty: characterise $p(x)$, propagate through to $p(y)$
 - ▶ For example, use Monte Carlo sampling
 - ▶ Generate random sample of x values from $p(x)$, run the model for each to get a random sample from $p(y)$
 - ▶ Structural uncertainty: characterise $p(z-y)$

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

- ▶ 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

- ▶ Key tasks 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
 - ▶ Learn about uncertain inputs from observations of the real process
 - ▶ Traditionally this is done by searching the x space for good fits to the data
- ▶ These tasks are impractical if the simulator takes more than a few seconds to run
 - ▶ We need a more efficient technique

Gaussian process representation

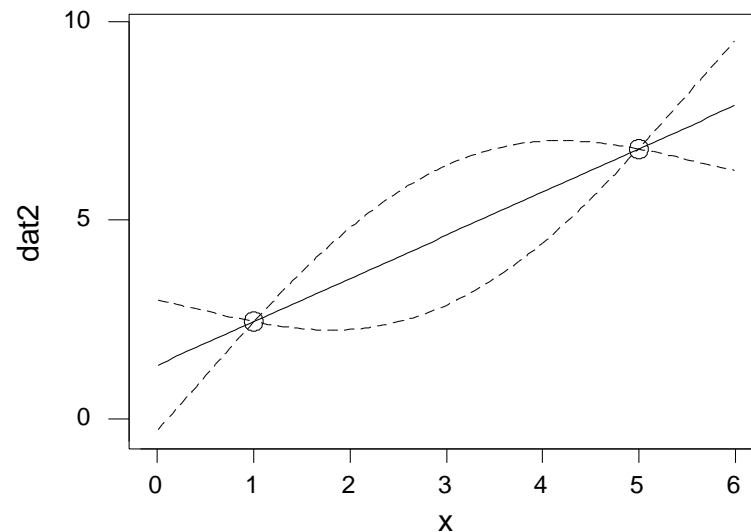
- ▶ More efficient approach
 - ▶ First work in early 1980s
- ▶ Consider the simulator as an unknown function
 - ▶ $f(\cdot)$ becomes a random process
 - ▶ We represent it as a Gaussian process (GP)
 - ▶ Conditional on hyperparameters
 - ▶ Or its Bayes linear analogue
- ▶ Training runs
 - ▶ Run simulator for sample of x values
 - ▶ Condition GP on observed data
 - ▶ Typically requires many fewer runs than MC
 - ▶ 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 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

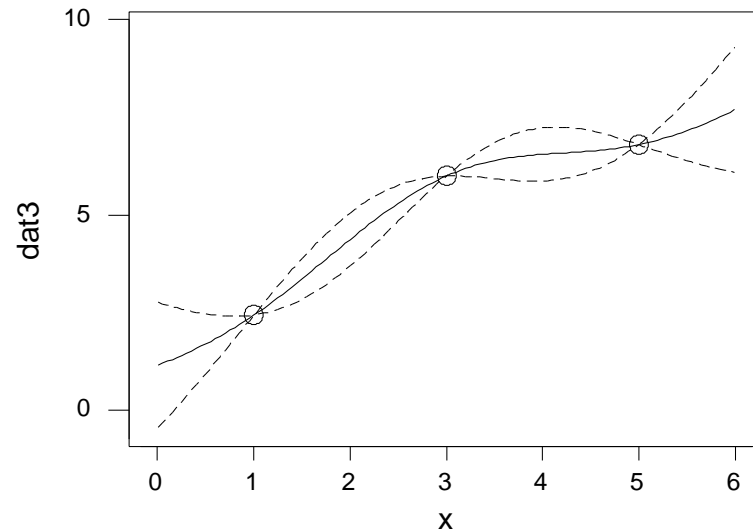
2 code runs

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



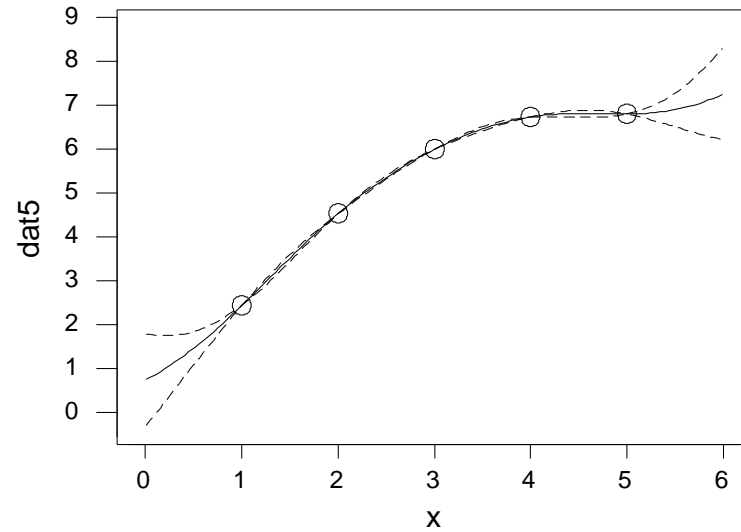
3 code runs

- ▶ Adding another point changes estimate and reduces uncertainty



5 code runs

- ▶ And so on

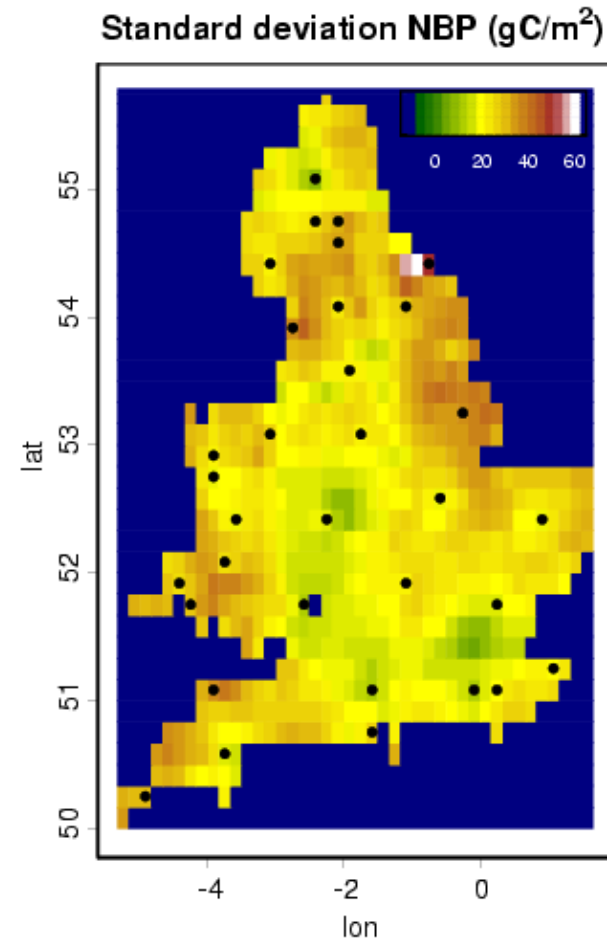
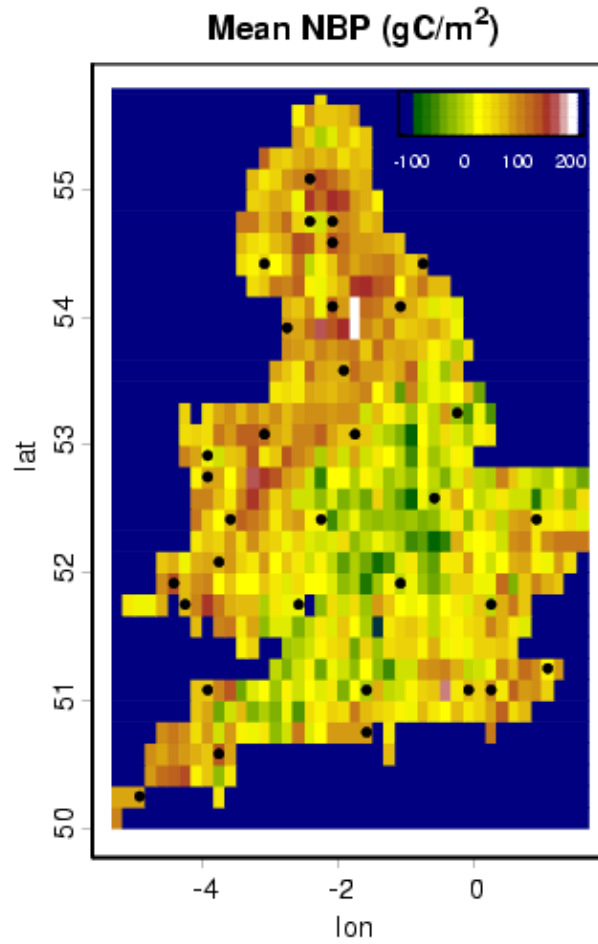


Then what?

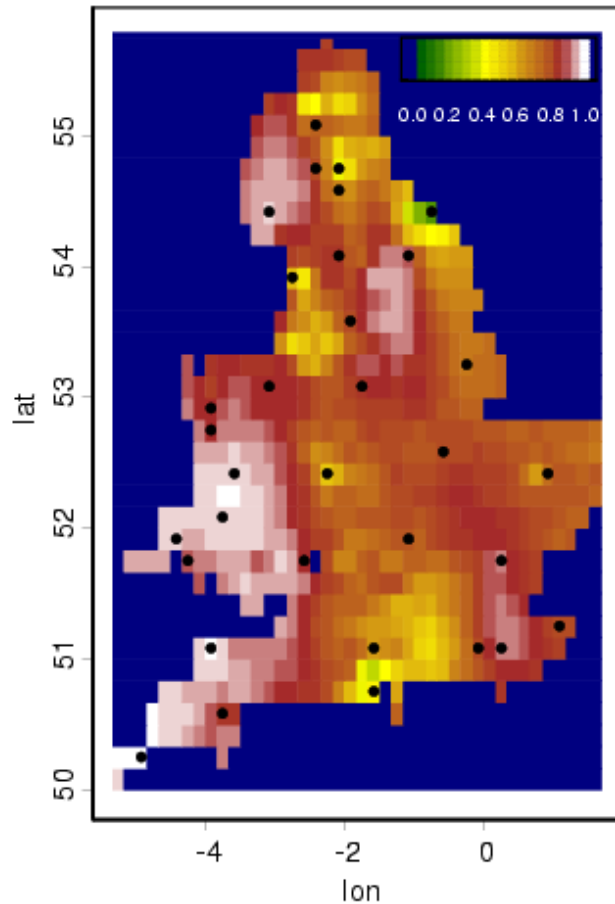
- ▶ Given enough training data points we can emulate any simulator accurately
 - ▶ So that posterior variance is small “everywhere”
 - ▶ Typically, this can be done with orders of magnitude fewer simulator runs than traditional methods
- ▶ Use the emulator to make inference about other things of interest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

Example: UK carbon flux in 2000

- ▶ Vegetation simulator predicts carbon exchange from each of 700 pixels over England & Wales
 - ▶ Principal output is Net Biosphere Production
 - ▶ Sheffield Dynamic Global Vegetation Model (SDGVM)
- ▶ Accounting for uncertainty in inputs
 - ▶ Soil properties
 - ▶ Properties of different types of vegetation
 - ▶ Propagate input uncertainty through the model
- ▶ Aggregated to England & Wales total
 - ▶ Allowing for correlations
 - ▶ Estimate 7.61 Mt C
 - ▶ Std deviation 0.61 Mt C



Sensitivity analysis



- ▶ Map shows proportion of overall uncertainty in each pixel that is due to uncertainty in the vegetation parameters
 - ▶ As opposed to soil parameters
- ▶ Contribution of vegetation uncertainty is largest in grasslands/moorlands

England & Wales aggregate

PFT	Plug-in estimate (Mt C)	Mean (Mt C)	Variance (Mt C ²)
Grass	5.28	4.65	0.323
Crop	0.85	0.50	0.038
Deciduous	2.13	1.69	0.009
Evergreen	0.80	0.78	0.001
Covariances			0.001
Total	9.06	7.61	0.372

Role of emulation

- ▶ Gaussian process emulation was crucial to the feasibility of this exercise
 - ▶ Almost 3000 simulator runs for a single set of inputs
 - ▶ Imagine this repeated hundreds or thousands of times for Monte Carlo
 - ▶ And all that repeated to evaluate the sensitivity to each input group
- ▶ We emulated each PFT at a sample of 33 sites
 - ▶ Typically 200 simulator runs for each
 - ▶ Kriging to interpolate between sites
 - ▶ Also equivalent to Gaussian process emulation

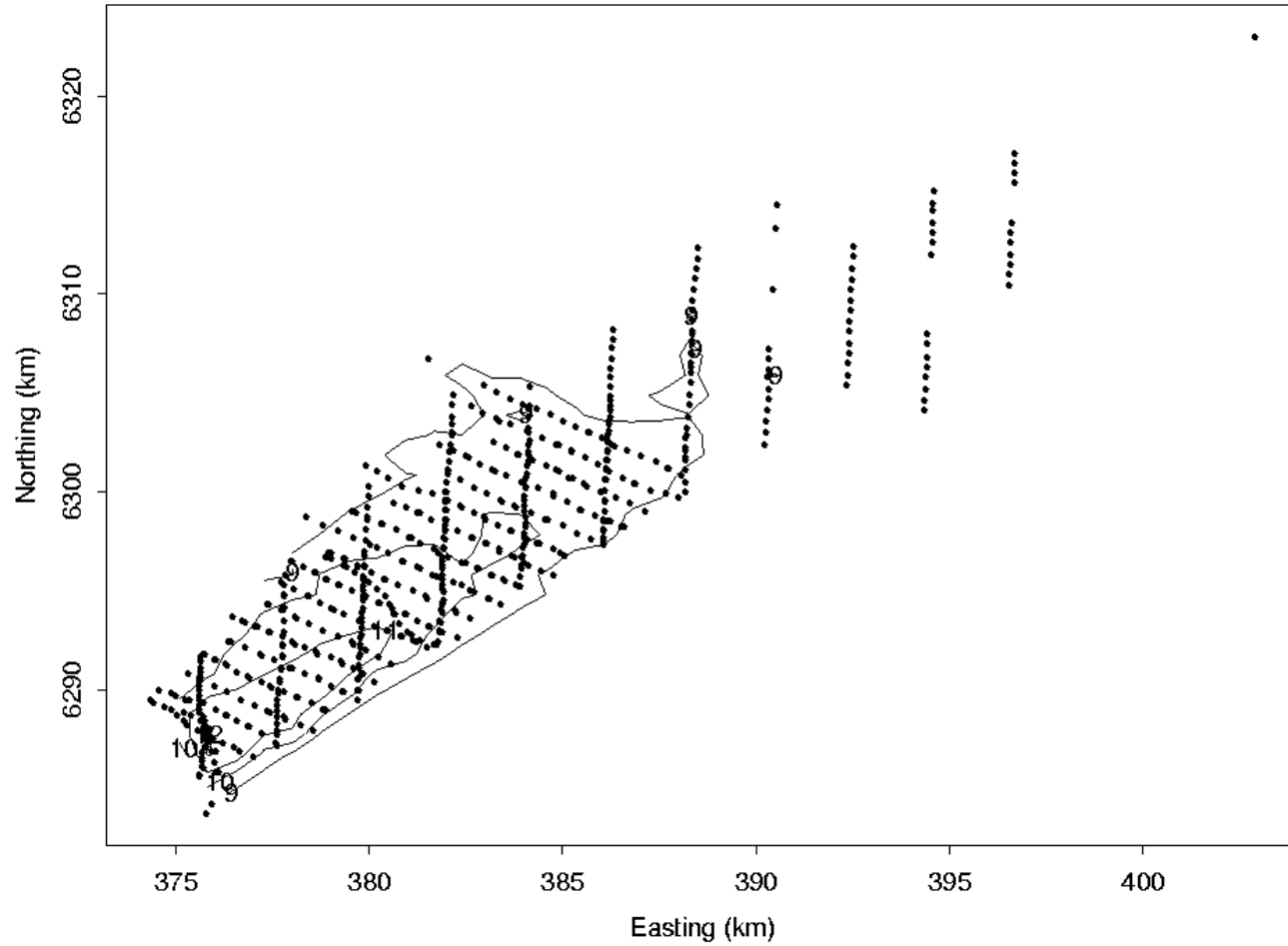
Kennedy, M. C. et al (2008). Quantifying uncertainty in the biospheric carbon flux for England and Wales. *Journal of the Royal Statistical Society A* 171, 109-135.

Example: Nuclear accident

- ▶ Radiation was released after an accident at the Tomsk-7 chemical plant in 1993
- ▶ Data comprise measurements of the deposition of ruthenium 106 at 695 locations obtained by aerial survey after the release
- ▶ The simulator is a simple Gaussian plume model for atmospheric dispersion
- ▶ Two calibration parameters
 - ▶ Total release of ^{106}Ru (source term)
 - ▶ Deposition velocity



Data



Calibration

- ▶ A small sample (N=10 to 25) of the 695 data points was used to calibrate the model
- ▶ Then the remaining observations were predicted and RMS prediction error of log-deposition computed

<i>Sample size N</i>	10	15	20	25
Best fit calibration	0.82	0.79	0.76	0.66
Bayesian calibration	0.49	0.41	0.37	0.38

- ▶ On a log scale, error of 0.7 corresponds to a factor of 2

Role of emulation

- ▶ The simulator in this case was fast
 - ▶ No need to emulate it
- ▶ But emulation is effectively used for the model discrepancy
 - ▶ Gaussian process representation
 - ▶ Trained on $N = 10, 15, 20, 25$ data points

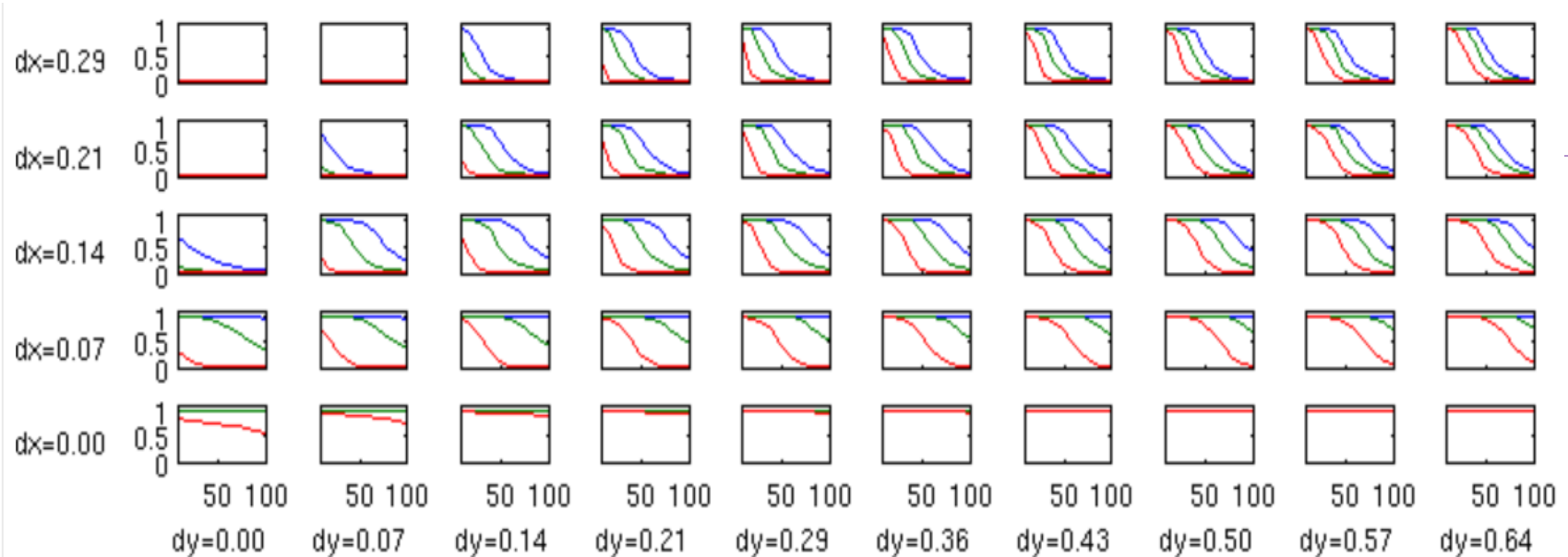
Kennedy, M. C. and O'Hagan, A. (2001). Bayesian calibration of computer models (with discussion). *Journal of the Royal Statistical Society B* 63, 425-464.

Example: Global warming

- ▶ How much CO₂ can we add to the atmosphere without increasing global mean temperature more than 2°C?
 - ▶ Future scenarios for atmospheric CO₂ concentrations are governed by 3 parameters, t_1 , dx and dy

$$C = \begin{cases} C_0(1 + dx)^t & \text{for } t < t_1 \\ C_0(1 + dx)^t(1 - dy)^{(t-t_1)} & \text{for } t \geq t_1 \end{cases}$$

- ▶ For what values of these parameters is there a realistic chance of staying below 2 degrees warming?
- ▶ Analysis based on C-Goldstein simulator
 - ▶ Although only of medium complexity, emulation was still crucial



- ▶ Red lines are for 2 degrees warming
 - ▶ Green for 4 degrees and blue for 6
- ▶ Each frame shows probability as a function of t_1
 - ▶ Chance of staying under 2 degrees decreases the later we act
 - ▶ And the faster we increase CO_2 before acting
 - ▶ And the slower we decrease thereafter
- ▶ That's as expected of course, but now we have quantitative assessments of the chance

Role of emulation

▶ Key steps in this exercise

1. Elicit expert knowledge about C-Goldstein parameter values and model discrepancy
2. Build an emulator for decadal average global mean temperature outputs
 - ▶ 17 inputs
3. Calibrate to observed global mean temperature anomalies
4. Build a second emulator for max temperature rise up to year 2010
 - ▶ 21 inputs including 3 parameters of future CO₂ scenario
5. Compute probabilities of staying under 2 degrees
 - ▶ Allowing for all uncertainties

Resources

- ▶ **MUCM project**

- ▶ Managing Uncertainty in Complex Models

- ▶ <http://mucm.ac.uk>

- ▶ **MUCM toolkit**

- ▶ Large set of web pages on building and using emulators

- ▶ Background, theory, discussion, advice, procedures, examples

- ▶ Case studies, including the global warming example

What about regional climate models?

- ▶ Applying this kind of methodology to regional climate models is very challenging
 - ▶ Very large numbers of inputs
 - ▶ Enormous run times for serious models
 - ▶ Scientists always want to run the heaviest models
- ▶ But if we can't credibly quantify uncertainty in the predictions of the heaviest models, what use are they?

- ▶ Discussion!

Part Two – Validation and Dynamic Emulation

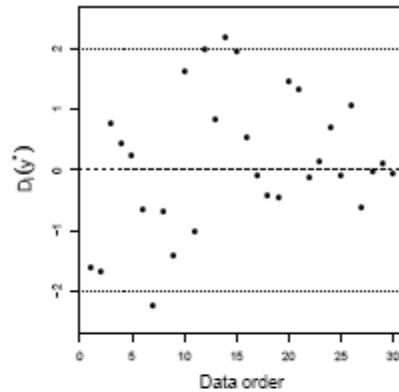
Validation

- ▶ **What does it mean to validate a simulation model?**
 - ▶ Compare simulator predictions with reality
 - ▶ But the model is always wrong
 - ▶ How can something which is always wrong ever be called valid?
- ▶ **Conventionally, a simulator is said to be valid if its predictions are close enough to reality**
 - ▶ How close is close enough?
 - ▶ Depends on purpose
 - ▶ Conventional approaches to validation confuse the absolute (valid) with the relative (fit for this purpose)
- ▶ **Let's look at an analogous validation problem**

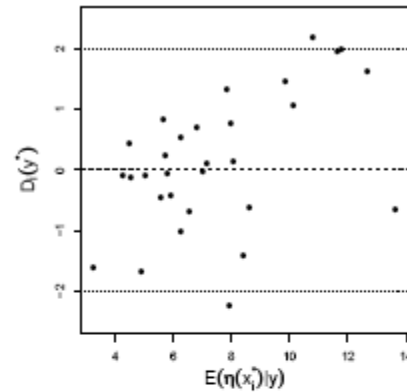
Validating an emulator

- ▶ What does it mean to validate an emulator?
 - ▶ Compare the emulator's predictions with the reality of simulator output
 - ▶ Make a validation sample of runs at new input configurations
 - ▶ The emulator mean is the best prediction and is always wrong
 - ▶ But the emulator predicts uncertainty around that mean
- ▶ The emulator is valid if its expressions of uncertainty are correct
 - ▶ Actual outputs should fall in 95% intervals 95% of the time
 - ▶ No less *and* no more than 95% of the time
 - ▶ Standardised residuals should have zero mean and unit variance

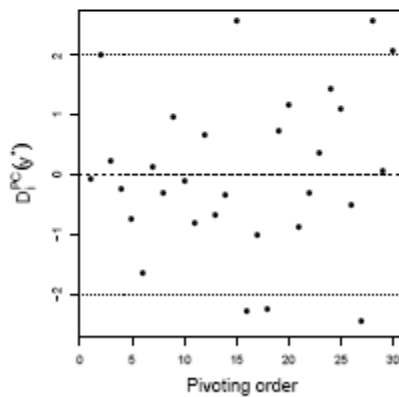
Validation diagnostics



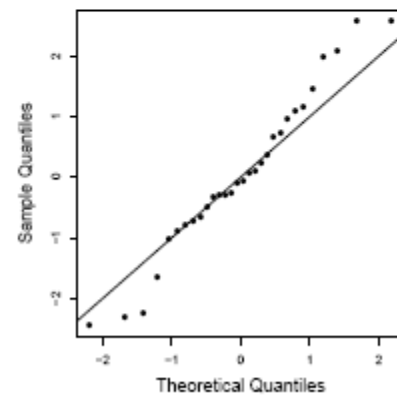
(a)



(b)



(c)



(d)

Validating the simulator

- ▶ Let's accept that there is uncertainty around simulator predictions
- ▶ We need to be able to make statistical predictions
 - ▶ Then if we compare with observations we can see whether reality falls within the prediction bounds correctly
- ▶ The difference between simulator output and reality is called *model discrepancy*
 - ▶ It's also a function of the inputs
 - ▶ Like the simulator output, it's typically a smooth function
 - ▶ Like the simulator output, we can emulate this function
 - ▶ We can validate this

Model discrepancy

- ▶ Model discrepancy was first introduced within the MUCM framework in the context of calibration
 - ▶ Ignoring discrepancy leads to over-fitting and over-confidence in the calibrated parameters
 - ▶ Understanding that it is a smooth error term rather than just noise is also crucial
- ▶ To learn about discrepancy we need a training sample of observations of the real process
- ▶ Then we can validate our emulation of reality using further observations

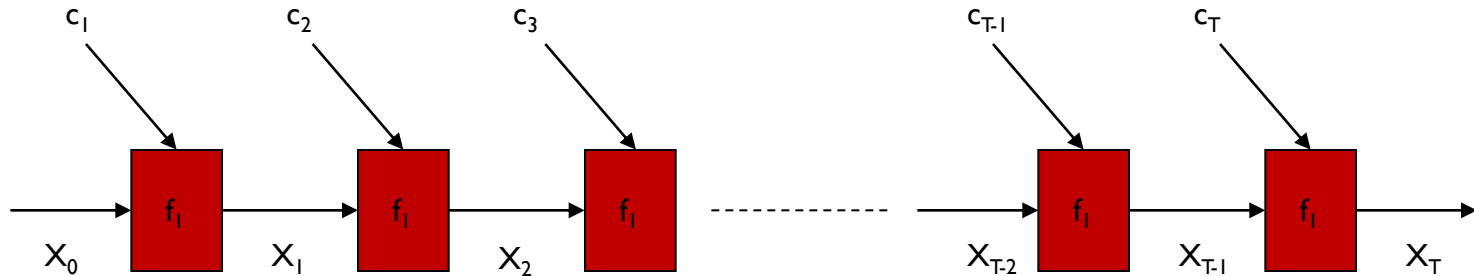
Beyond validation

- ▶ An emulator (of a model or of reality) can be valid and yet useless in practice
 - ▶ Given a sample of real-process observations, we can predict the output at any input to be the sample mean plus or minus two sample standard deviations
 - ▶ This will validate OK
 - ▶ Assuming the sample is representative
 - ▶ But it ignores the model and makes poor use of the sample!
- ▶ Two valid emulators can be compared on the basis of the variance of their predictions
- ▶ And declared fit for purpose if the variance is small enough

Summary: My view of validation

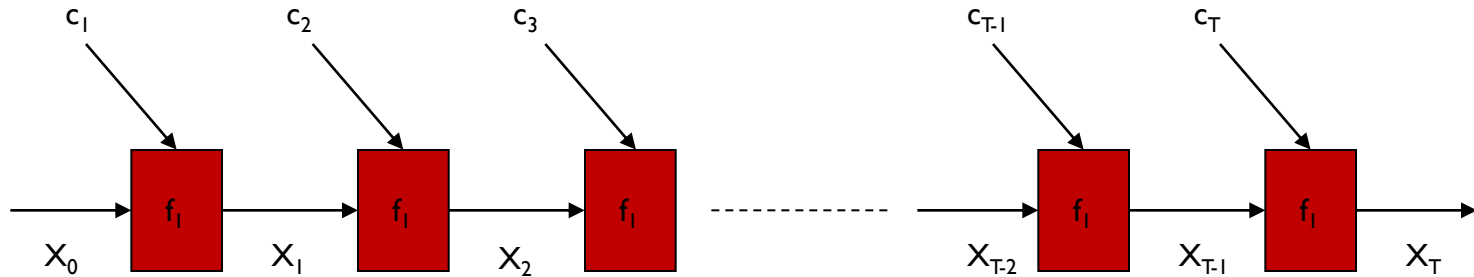
- ▶ I think it is useful to separate the absolute property of validity from the relative property of fitness for purpose
- ▶ Simulator predictions alone are useless without some idea of how accurate they are
- ▶ Quantifying uncertainty in the predictions by building an emulator allows us to talk about validity
- ▶ Only valid statistical predictions of reality should be accepted
 - ▶ Simulator predictions with a *false* measure of their accuracy are also useless!
- ▶ We can choose between valid predictions on the basis of how accurate they are
 - ▶ And ask if they are sufficiently accurate for purpose

Dynamic simulators



- ▶ Initial state x_0 is updated recursively by the one-step simulator $f_1(x, c, b)$
 - ▶ $x_t = f_1(x_{t-1}, c_t, b)$
 - ▶ $x_t = f_t(x_0, \mathbf{c}^t, b) = f_1(\dots f_1(f_1(x_0, c_1, b), c_2, b) \dots, c_t, b)$
- ▶ Forcing inputs c_t
 - ▶ Also parameters b in f_1 that apply at all time steps
- ▶ We are interested in the sequence x_1, x_2, \dots, x_T
- ▶ At least 4 approaches to emulating this

I. Treat time as input



- ▶ Emulate x_t as the function

$$f(x_0, t, b) = f_t(x_0, \mathbf{c}^t, b)$$

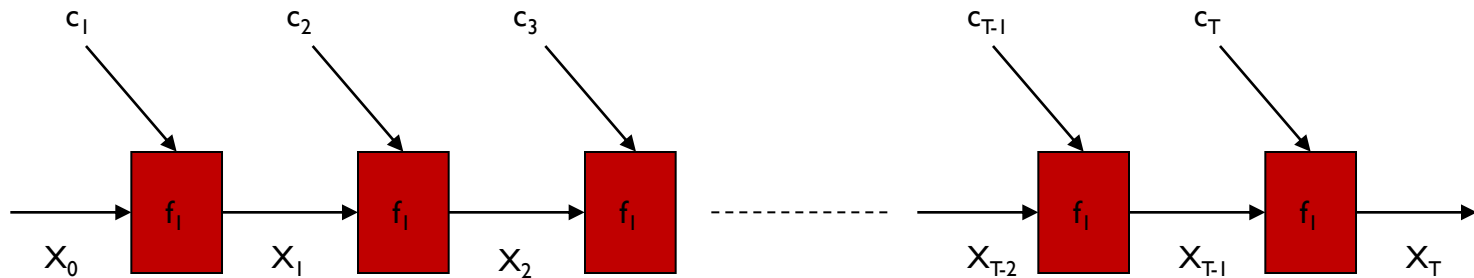
- ▶ Easy to do

- ▶ Spatial coordinates treated as inputs in the nuclear accident example

- ▶ Hard to get the temporal correlation structure right

- ▶ And we have to fix the forcing

2. Multivariate emulation



- ▶ Emulate the vector $\mathbf{x} = (x_1, x_2, \dots, x_T)$ as the multi-output function

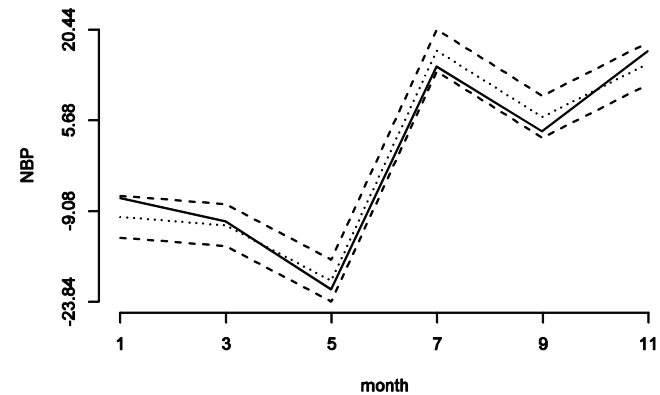
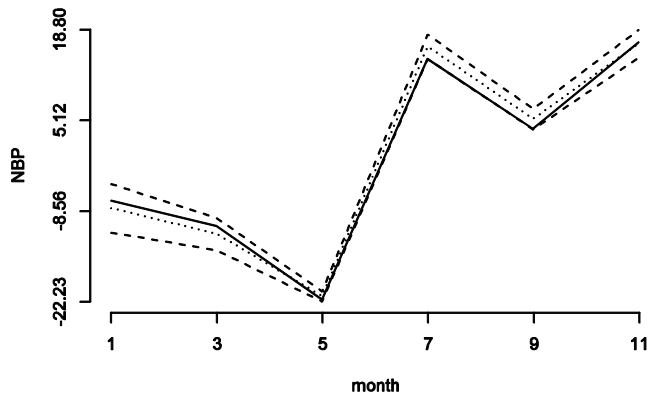
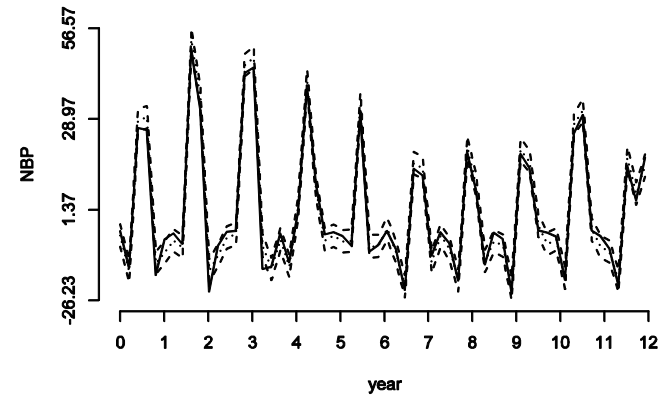
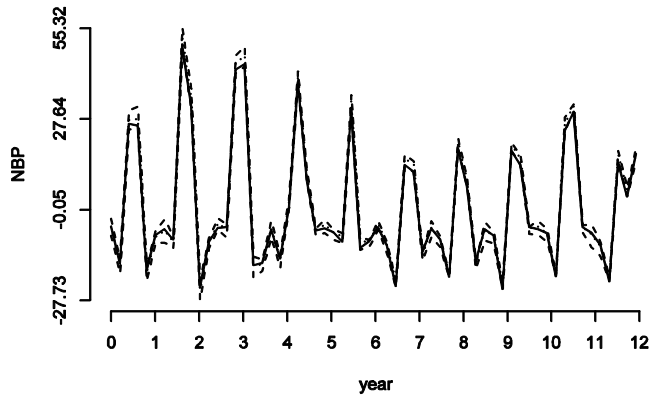
$$\mathbf{f}_T(x_0, \mathbf{b}) = (f_1(x_0, \mathbf{c}^1, \mathbf{b}), f_2(x_0, \mathbf{c}^2, \mathbf{b}), \dots, f_T(x_0, \mathbf{c}^T, \mathbf{b}))$$

- ▶ Simple extension of univariate theory
- ▶ Restrictive covariance structures
 - ▶ And again problematic if forcing is uncertain

Example

- ▶ Time series output from Sheffield Global Dynamic Vegetation Model (SDGVM)
 - ▶ Dynamic model on monthly time-step
 - ▶ Large state vector, forced by rainfall, temperature, sunlight
- ▶ 10 inputs
 - ▶ All others, including forcing, fixed
- ▶ 120 outputs
 - ▶ Monthly values of NBP for ten years
- ▶ Compare multi-output emulator with time-as-input

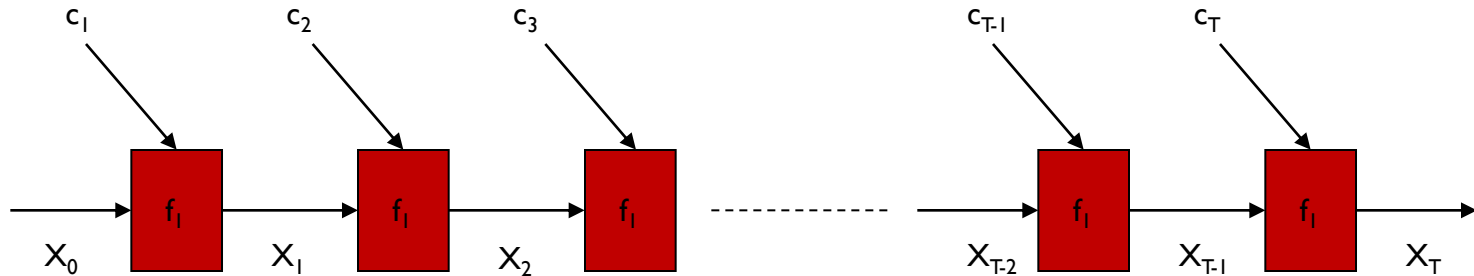
Conti, S. and O'Hagan, A. (2010). Bayesian emulation of complex multi-output and dynamic computer models. *Journal of Statistical Planning and Inference* 140, 640-651.



Multi-output emulator on left, time-as-input on right
 For fixed forcing, both seem to capture dynamics well
 Multi-output performs slightly better and validates much better

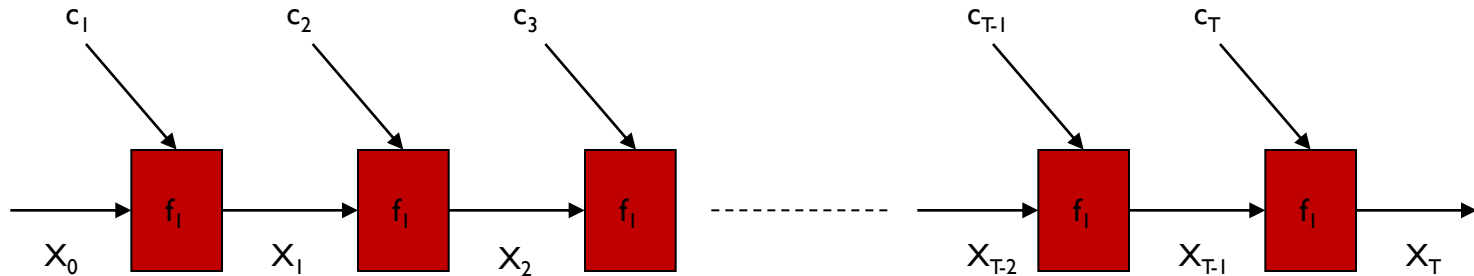


3. functional output emulation



- ▶ Treat x series as a functional output
- ▶ Identify features (e.g. principal components) to summarise function
- ▶ Same theory as for multivariate emulation, but lower dimension
 - ▶ Components often emulated independently
- ▶ Loss of information, no longer reproduces training data
 - ▶ And uncertain forcing is still challenging

4. Recursive emulation



- ▶ Emulate the single step function $f_1(x, c, b)$
- ▶ Iterate the emulator
 - ▶ The focus of this talk
- ▶ Only recursive emulation can
 - ▶ ... realistically treat uncertainty in forcing inputs
 - ▶ ... extend T beyond training data
 - ▶ ... learn dynamically from time series data (data assimilation)

Iterating a function

- ▶ Theoretical challenge:
- ▶ We have a Gaussian process representation for f_1
 - ▶ Conditional on hyperparameters with known distributions
- ▶ We want to characterise the distribution of x_N , which is the result of f_1 iterated N times

$$x_N = f_N(x_0, \mathbf{c}^N, \mathbf{b})$$

- ▶ where $f_t(x_0, \mathbf{c}^t, \mathbf{b}) = f_1(f_{t-1}(x_0, \mathbf{c}^{t-1}, \mathbf{b}), \mathbf{c}_t, \mathbf{b})$
 - ▶ $\mathbf{c}^t = (c_1, c_2, \dots, c_t)$ is the forcing inputs up to time t
 - ▶ x_0 is the initial state vector
 - ▶ and \mathbf{b} represents model parameters
- ▶ The three arguments may also be uncertain

Some features of the problem

- ▶ The one-step function f_1 has many outputs.
 - ▶ The set of state variables
 - ▶ Need to use multi-output emulation
 - ▶ Multivariate Gaussian process
- ▶ And many inputs
 - ▶ But only forcing for a single time point
- ▶ We need to emulate very accurately
 - ▶ Accuracy will degrade with every iteration
- ▶ Because f_1 is deterministic, we have interesting constraints
 - ▶ If x_t returns to some earlier point
 - ▶ ...and receives the same forcing as before
 - ▶ ...then the state will update exactly as before

Exact solution by simulation

- ▶ We can compute properties of the distribution of x_N by Monte Carlo
 - ▶ Draw x_0 , c_1 and b from their distributions (if uncertain)
 - ▶ Draw values of Gaussian process hyperparameters
 - ▶ Draw $x_1 = f_1(x_0, c_1, b)$ from its (normal) distribution
 - ▶ Draw c_2 from its distribution conditional on the sampled c_1
 - ▶ Draw $x_2 = f_1(x_1, c_2, b)$ from its (normal) distribution given that $f_1(x_0, c_1, b) = x_1$
 - ▶ Iterate the process in the last two steps until x_N is generated
 - ▶ Repeat the whole process to generate a sample from the distribution of x_N
- ▶ This may not be appreciably faster than running the original simulator and using Monte Carlo!

Solution for a single step

- ▶ Given assumptions on the form of the GP we can characterise uncertainty about x_1 exactly
- ▶ Assume
 - ▶ GP mean function known or linear in unknown parameters
 - ▶ GP covariance function of squared exponential form
 - ▶ With unknown variance and correlation length parameters
 - ▶ Vague or conjugate prior on mean parameters and variance
 - ▶ Normal prior distribution for inputs x_0, c_1, b
- ▶ Then
 - ▶ Training sample of simulator runs gives emulator as posterior GP
 - ▶ Ignore posterior uncertainty in covariance parameters
 - ▶ We can derive closed form expressions for moments, cdf, etc. of x_1

Outline of derivation

- ▶ **Suppose**
 - ▶ x is one-dimensional, no unknown forcing or parameters
 - ▶ $f_1(x)$ has zero prior mean and covariance function
 - ▶ $c(x, x') = \exp\{-d(x-x')^2\}$
- ▶ **After n training observations $y_i = f_1(x_i)$**
 - ▶ Posterior mean function $m(x) = R(x)y$
 - ▶ Posterior correlation function $r(x, x') = c(x, x') - R(x)A^{-1}R(x')^T$
 - ▶ Where $R(x)$ and A comprise $c(x, x_i)$ and $c(x_i, x_j)$ respectively
- ▶ **Given normal density $p(x)$ for X**
 - ▶ $E[f_1(X)] = E[m(X)] = \int m(x)p(x)dx$
 - ▶ We can evaluate this analytically

Approximate filtering solution

- ▶ If the distribution of x_1 (jointly with c_2 and b) were normal we could repeat this to get the distribution of x_2
- ▶ The assumption of normality is good if f_1 is nearly linear
 - ▶ Generally true for single step function
- ▶ So an approximate solution is to apply this theory iteratively
 - ▶ Compute mean and variance of x_t and assume normality
- ▶ Technical issues
 - ▶ Approximation will deteriorate with number of iterations
 - ▶ Loses the feature of updating the GP at each iteration
 - ▶ Uncertainty in c_t and b needs careful handling

References for Part Two

Bastos, L. S. and O'Hagan, A. (2009). Diagnostics for Gaussian process emulators. *Technometrics* 51, 425-438.

Conti, S., Gosling, J.P., Oakley, J.E. and O'Hagan, A. (2009). Gaussian process emulation of dynamic computer codes. *Biometrika* 96, 663-676.

Also see the MUCM (<http://mucm.ac.uk/>) toolkit for detailed coverage of these topics: