# On history matching

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Why calibrate  * In order that you have simulators which can predict well. * In order to be able to assess how good these predictions are (and not just by waiting and seeing!)  How to calibrate First things first, you will want to take into account model discrepancy when you calibrate. Not doing so will provide a less good match between outputs and the observations.  Then try one of these:   * Goodness of fit (e.g. least squares), though difficult with complex simulators * Likelihood function, also challenging for complex simulators * Maximum likelihood via iterated filtering * Data augmented MCMC * Reversible-jump MCMC * Stochastic differential equations   If these are hard (due to the likelihood being intractable) we try to not be reliant on the likelihood and instead rely on outputs from simulator runs then use:   * Particle filtering * Approximate Bayesian Computation * Pseudo-marginal methods * Particle MCMC  Why people don’t always calibrate Where the specific difficulty comes is that most of the methods above relies on a large number of simulator runs, a big problem if runs are computationally expensive (add to this the fact that the task of matching multiple outputs while varying many inputs is computationally intensive). More runs are required if using stochastic simulators. (This similarly affects SA and UA.) This problem isn’t going away with increased computing power, since modellers increase the complexity of their models (to hopefully improve their accuracy) to take advantage of these advances.  This can result in calibration not being (formally) done:  And this problem is getting worse over time. What to do  * Simplify the simulator (will increase simulator inadequacy, which might invalidate work). * Combine *Bayesian history matching* and *emulation* and *model discrepancy*.  What does HM do and why is that good? View HM as a pre-calibration step, or broad calibration step. It might be all the modeller requires.   |  |  |  | | --- | --- | --- | | * Does NOT make full probabilistic statements about the input values most likely to match the sim’s output to the data [*Lifted*].     Emulator use is required for history matching, which, remember, returns a joint probability distribution of the sim outputs for any set of input values [*Lifted*], details of which we shall see later.   * Areas of the input space (labelled *implausible*) that are unlikely to produce an output matching the data are removed, leaving NROY space remaining.          * This process is not performed in one go but iteratively (in steps known as *waves*). |  | This is good since this would involve intractable calculations if attempted (since using Bayesian MCMC, say).  And this is typically justifiable since the sim is probably not thought of as an accurate enough rep of the reality [*Lifted*] to justify such calculations, and HM deals in expectations and variances, things usually of primary interest to modellers.    This exclusion step can be done for a single pair of input and output, without needing to consider the others. For example, if an area of a particular input’s space is deemed implausible for the value of a particular output, this area can simply be ruled out, without needing to consider the behaviour of the other variables, be they inputs or outputs.  Contrast this with Bayesian MCMC or MLE methods which, due to the use of a likelihood function, have to consider all the available information simultaneously.  Also, if the simulator is simply incompatible with the data (due to incorrect modelling assumptions, poor error spec’s or coding errors [*Lifted*], HM will indicate this (by declaring ALL input space implausible.  Contrast this with other methods which will ALWAYS return a posterior distribution, regardless of how well (or if) the sim fits the data.  Step-by-step, the input parameter space shrinks, and the simulator starts to behave more predictably (and more smoothly). Once you’ve arrived at this point, probabilistic statements are easier to make (see later). |  How to carry out HM  1. LHC design to select input values (*n* = 10*p* for training and *nv* = *p* for validation), resulting in *D*, the initial design space. 2. Run the simulator at the points **x∈***D* (*K* times if non-deterministic). 3. If non-deterministic, calculate the mean of the *K* simulator runs (as an estimate of *gj*(**x***i*)) and their variance. 4. Construct an emulator for each of the (, if non-deterministic, mean of the) for each of the *j* = 1, …, *r* output quantities. 5. Choose an implausibility measure and rule out implausible space according to this measure. 6. Sample from non-implausible space:    1. EITHER draw samples from the entire input space, rejecting those which fail the implausible space [*Lifted*]. This is easy to do BUT if the implausible space is just a very small part of the overall input space (which it often is in later waves), this becomes inefficient.    2. OR evolutionary MC.    3. OR say that in wave *i* we have a number of non-implausible points **x**, then:       1. for each of these draw *k* samples from a *p*-variate Normal distn centred on the value of the generating point, with variance chosen s.t. only around 20% of the newly sampled points are non-implausible (ensuring new samples are sufficiently different from the old ones [*Lifted*]       2. the *i*th wave implausibility is evaluated on the new samples 7. Return to 2., but with **x** being a subset of the sample from 6. 8. Once a stopping criterion has been met, stop, e.g.    1. All of the input space has been declared implausible (simulator cannot match the observations given the current error specifications [*Lifted*]) => vary the model discrepancy in order to work out how large it should be in order to result in a match (too large and you need a different (better) simulator).    2. Once emulator variance < obs. unc. + m.d. + ens. var., the non-implausible space now contains acceptable matches and is unlikely to decrease in size in future iterations (w/o revising and decreasing the remaining unc.’s in the system). Check the acceptable matches against the outputs not used in the emulation process.    3. If the simulator runs obtained in the current wave are close enough to the empirical data and we do not wish to continue any further. 9. If 8b. or 8c., investigate the sensitivity and robustness of the non-implausible region to alterations in the uncertainties and model discrepancy. |  | Andrianakis et al (2015)  Can be used w/ model d?  Why?  Why? |

# On “true values” of the parameters…

There are “true values of the inputs” (\*). We don’t know these, and simoutput(variableinputs) will thus be wrong. simoutput(variableinputs, u) will bring us closer and a good or the best-we-can u is arrived at using observations and derpivating posteriors for ‘the true value’ of u. Even if u = θ, simoutput(variableinputs, θ) is still wrong, due to modeldiscrep(variableinputs), so \* can’t mean those inputs which lead to perfect predictions of the outputs.

Using the model

obs = realprocess(variableinputs) + residvarobserror

= ρ simoutput(variableinputs, calibinputs) + modeldiscep(variableinputs) + residvarobserror

with the central part able to be viewed (not perfectly but usefully) as a non-linear regression model. The regression function is defined by the simulator itself, through the (variableinputs, calibinputs) term, with parameters ρ and calibinputs. Then the terms modeldiscep(variableinputs) and residvarobserror can be viewed at residuals. **In this content, the “true value” of** calibinputs **has the same meaning as the true values of regression parameters – the true** calibinputs **is a best-fitting** calibinputs**.**

The calibration inputs will generally have been given concrete physical meanings but the people who built the simulator, but the actual values of these quantities in the real world don’t necessarily equate to θ, an inevitablility when it is accepted the odel can’t ever be a perfect fit. It may be that a

# On the relationship between the simulator output and reality…

One way to model the relationship between the simulator’s output and reality is

realprocess(variableinputs) = ρ simoutput(variableinputs, calibinputs) + modeldiscep(variableinputs)

**Assume you know the calibinputs** and **you can make as many runs of the sim as you want**, in order to observe simoutput(variableinputs, calibinputs) for various variableinputs. Suppose first that to predict realprocess(variableinputs’) at some specific point variableinputs’ we would deem it sufficient to observe simoutput(variableinputs’, calibinputs) (a single run at variableinputs’).