QUT
BRISBANE
AUSTRALIA

# 0. Analysing sloppiness of simulation models: An overview

# What is model sloppiness?

The definition is still a bit nebulous in the literature. We will go with the following definition:

> A **sloppy model** refers to a model where most of the model behaviour is captured through a few tightly constrained *parameter combinations*, which are highly influential on model predictions of the data, but remains insensitive to many loosely constrained *parameter combinations*.

What do we mean by a "parameter combination"?

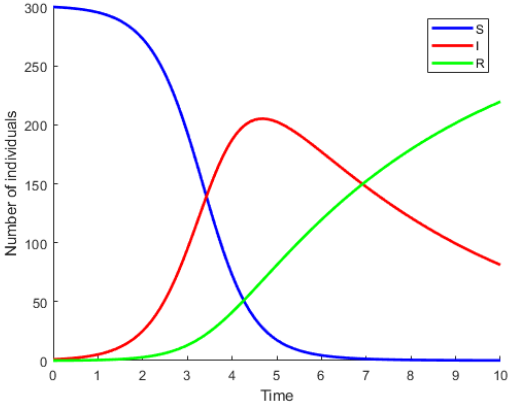
And what does it mean for a parameter combination to be "tightly constrained" or "loosely constrained"?

# Tightly or loosely constrained parameter combinations?

Consider the classic SIR model in epidemiology:

$$\frac{\mathrm{d}S}{\mathrm{d}t} = -\frac{\beta IS}{N}$$
$$\frac{\mathrm{d}I}{\mathrm{d}t} = \frac{\beta IS}{N} - \gamma I$$
$$\frac{\mathrm{d}R}{\mathrm{d}t} = \gamma I$$

where $S$ is the number of susceptible individuals, $I$ is the number of infected individuals $R$ is the number of recovered individuals, and $N = S + I + R$ is the total population.




If we only have noisy data for the infected population up until time $t = 2$, what parameters can we estimate?

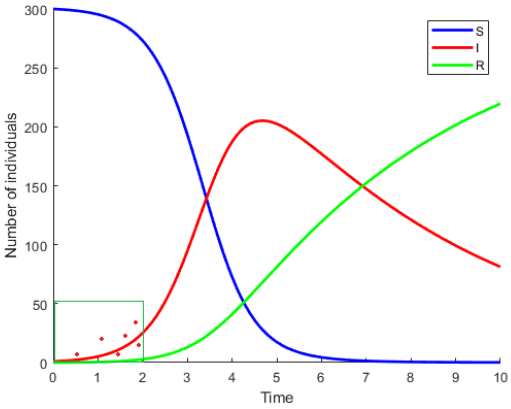
# Tightly or loosely constrained parameter combinations?

$$\frac{\mathrm{d}S}{\mathrm{d}t} = -\frac{\beta IS}{N}$$
$$\frac{\mathrm{d}I}{\mathrm{d}t} = \frac{\beta IS}{N} - \gamma I$$
$$\frac{\mathrm{d}R}{\mathrm{d}t} = \gamma I$$

At the start of the epidemic,
$N \approx S \approx$ constant, so the data parameterises

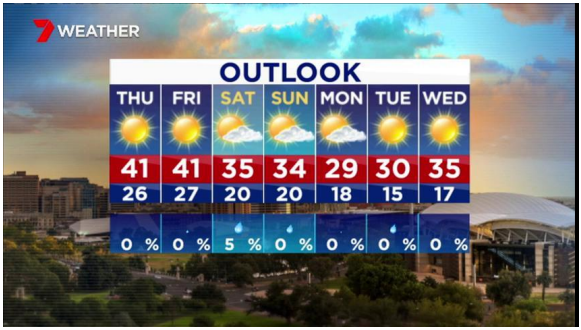
$$\frac{\mathrm{d}I}{\mathrm{d}t} \approx [\beta - \gamma]\, I.$$

$\therefore$ Using this data we will only be able to confidently estimate the *parameter combination* $\beta - \gamma$.




We will know almost nothing about $\beta$ or $\gamma$ separately! Why does this matter?

# So what if some parameters (or parameter combinations) are sloppy?




We might trust our model too much! OR

We might want to change our model! OR

We might want to figure out what data we need to make our model better.

# Analysis of sloppiness: A systematic approach to finding out which parameter combinations are tightly/loosely constrained

Overall process:

1. Calibrate our proposed model to the data.
2. Calculate a specific matrix, called the "sensitivity matrix". This matrix is size $N_\theta \times N_\theta$, where $N_\theta$ is the number of model parameters being estimated from the data.
3. Use this matrix to obtain a list of $N_\theta$ *orthogonal* parameter combinations, ordered from stiffest (we know the most about) to sloppiest (we know the least about).

This ordered list of orthogonal parameter combinations is the primary goal of analysis of model sloppiness!

We can use knowledge of these parameter combinations to potentially inform our decisions about future modelling and/or future data collection efforts.

# *Orthogonal* parameter combinations? Re-consider the epidemic example

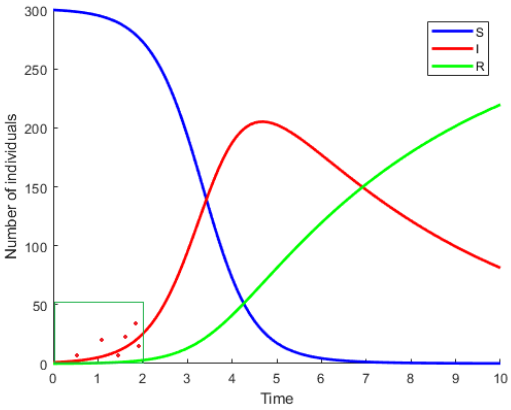
At the start of the epidemic,
$N \approx S \approx$ constant, so the data parameterises

$$\frac{\mathrm{d}I}{\mathrm{d}t} \approx [\beta - \gamma]\, I.$$

So the two orthogonal parameter combinations *likely* to be found from analysis of model sloppiness are:

- Stiff parameter combination: $\beta - \gamma$
- Sloppy parameter combination: $\beta + \gamma$

These two parameter combinations are orthogonal if plotted in two-dimensional parameter space...



# *Orthogonal* parameter combinations? Visualisation in parameter space

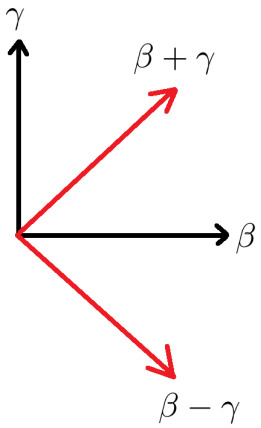
- Stiff parameter combination: $\hat{\theta}_1 = \beta - \gamma$
- Sloppy parameter combination: $\hat{\theta}_2 = \beta + \gamma$

These two parameter combinations are orthogonal if plotted in two-dimensional parameter space...

This example worked well because $\beta$ and $\gamma$ are parameters with the same units (inverse time).

In more complex models, parameters can be several orders of magnitude different from each other, so...

It's usually better to seek parameter combinations that are orthogonal after logarithmic transformation, e.g.:




$$\log(\hat{\theta}_1) = \log(\beta) - \log(\gamma) \quad \text{and} \quad \log(\hat{\theta}_2) = \log(\beta) + \log(\gamma).$$

or equivalently: $\hat{\theta}_1 = \beta/\gamma$ and $\hat{\theta}_2 = \beta\gamma$.

# *Orthogonal* parameter combinations? Movin' into log-space

It's usually better to seek parameter combinations that are orthogonal after logarithmic transformation, e.g.:

$$\log(\hat{\theta}_1) = \log(\beta) - \log(\gamma) \quad \text{and} \quad \log(\hat{\theta}_2) = \log(\beta) + \log(\gamma).$$

or equivalently: $\hat{\theta}_1 = \beta/\gamma$ and $\hat{\theta}_2 = \beta\gamma$.

More generally we will be seeking an ordered list of parameter combinations (from stiffest to sloppiest) of the form

$$\hat{\theta}_j = \theta_1^{v_{j,1}} \times \theta_2^{v_{j,2}} \times \theta_{N_\theta}^{v_{j,N_\theta}}, \quad \text{where} \quad j = 1, ..., N_\theta,$$

and $\theta_1$, $\theta_2$,..., $\theta_{N_\theta}$ are the original parameters of the model.

e.g. for the boxed example above, $\theta_1 = \beta$, $\theta_2 = \gamma$, $v_{1,1} = 1$, $v_{1,2} = -1$, $v_{2,1} = 1$, $v_{2,2} = 1$.

From now on we will refer to these $\hat{\theta}_j$ as either parameter combinations or *eigenparameters*.

# Structure of this course!

Four modules:

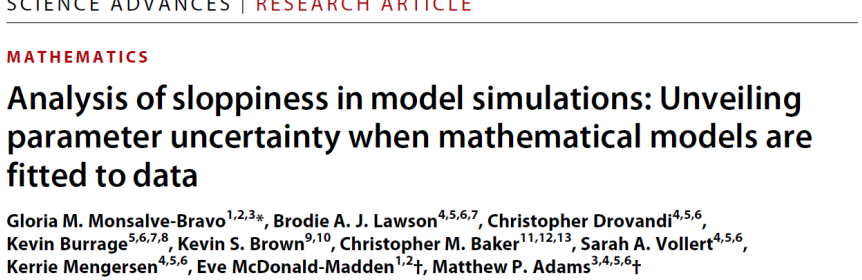
1. Model-data calibration using Bayesian and frequentist approaches.
2. Visualisation of calibration outputs.
3. Analysis of model sloppiness.
4. Interpreting model sloppiness outputs, and connections to other data-informing approaches.

Each module has varying length. In the first 3 modules, we will be switching between slides and Python code as we go through. The last module provides some examples of applications and suggestions for further reading (if of interest!).

# Source of the course!

I am heavily indebted to Gloria Monsalve-Bravo's research without whom this topic would not be presentable in its current form.

**Key reference for this course:** Monsalve-Bravo *et al*. (2022) Analysis of model sloppiness in model simulations: Unveiling parameter uncertainty when mathematical models are fitted to data. *Science Advances* **8**: eabm5952.



SCIENCE ADVANCES | RESEARCH ARTICLE

MATHEMATICS

## Analysis of sloppiness in model simulations: Unveiling parameter uncertainty when mathematical models are fitted to data

Gloria M. Monsalve-Bravo[1,2,3]*, Brodie A. J. Lawson[4,5,6,7], Christopher Drovandi[4,5,6], Kevin Burrage[5,6,7,8], Kevin S. Brown[9,10], Christopher M. Baker[11,12,13], Sarah A. Vollert[4,5,6], Kerrie Mengersen[4,5,6], Eve McDonald-Madden[1,2]†, Matthew P. Adams[3,4,5,6]†

# Useful glossary for the course modules (1 of 4)

**Model** A mathematical description of a model we are fitting to the data. We'll be focusing here on deterministic models. Notation: $\boldsymbol{y}_{\text{model}}$.

**Parameters** A vector of model parameters we want to estimate using the data. Notation: $\boldsymbol{\theta}$.

**Data** The data/observations we are comparing our model to! Notation: $\boldsymbol{y}_{\text{obs}}$, measured at input conditions $\boldsymbol{x}_{\text{obs}}$.

**Calibration** The process of fitting our model $\boldsymbol{y}_{\text{model}}(\boldsymbol{x}, \boldsymbol{\theta})$ to the data $\boldsymbol{y}_{\text{obs}}$ measured at $\boldsymbol{x}_{\text{obs}}$. This process aims to improve our knowledge about the values of $\boldsymbol{\theta}$.

# Useful glossary for the course modules (2 of 4)

**Frequentist statistics** Frequentist statistics/frequentist inference is a particular (philosophical) approach to statistics. It's the one most people learn in first year uni! Examples: Hypothesis testing, z-tests, t-tests, etc.

**Bayesian statistics** Bayesian statistics/Bayesian inference is a particular (philosophical) approach to statistics. It's a bit harder to get into (it's not learned at QUT until second/third year!) but in my opinion is a lot cleaner / more elegant. Examples: MCMC, SMC, anything that has the word 'Bayes' in it.

**Likelihood** Short for *likelihood function*. The probability that the dataset $\boldsymbol{y}_{\text{obs}}$ was obtained from a model $\boldsymbol{y}_{\text{model}}$ with parameters $\boldsymbol{\theta}$. Used in both frequentist and Bayesian statistics. Notation: $\mathcal{L}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta})$.

# Useful glossary for the course modules (3 of 4)

**MLE** Short for *maximum likelihood estimation* OR (confusingly) *maximum likelihood estimator*. One set of parameters $\boldsymbol{\theta}$ which best fit the data. (Commonly the goal of frequentist calibration.) Notation: $\boldsymbol{\theta}_{\mathrm{MLE}}$.

**Samples** A collection of parameter sets $\boldsymbol{\theta}$. For example, if we have $M$ samples, then we have $M$ parameter vectors $\boldsymbol{\theta}_m$, where $m = 1, ..., M$. (Specific to Bayesian inference.) Notation: $\{\boldsymbol{\theta}_m\}_{m=1}^{M}$.

**Prior** Short for *prior distribution*. Our initial/prior beliefs of model parameter values before looking at the data. (Specific to Bayesian inference.) Notation: $p(\boldsymbol{\theta})$.

**Posterior** Short for *posterior distribution*. Our final beliefs of model parameter values at the end, i.e. after looking at the data. (The goal of Bayesian calibration.) Notation: $p(\boldsymbol{\theta}|\boldsymbol{y}_{\mathrm{obs}})$.

# Useful glossary for the course modules (4 of 4)

**Sensitivity matrix** A square matrix of size $N_\theta \times N_\theta$, where $N_\theta$ is the number of model parameters being estimated, whose calculation is required for analysis of model sloppiness. Examples: **H**, **L**, **P**.

**Eigenparameter** A multiplication of model parameters to some index between -1 and 1. Eigenparameters are the key output of analysis of model sloppiness. For example, $\hat{\theta}_1 = \theta_1^{0.6}\theta_2^{-0.4}$. Notation: $\hat{\theta}_j$.

**Sloppy eigenparameter** An eigenparameter whose value causes very little change to the model-data calibration. These eigenparameters are the source of "model sloppiness".

**Stiff eigenparameter** An eigenparameter, for which any change in its value, would cause the model-data calibration to change A LOT. Eigenparameters $\hat{\theta}_j$ are ordered from stiffest to sloppiest, so $\hat{\theta}_1$ is the stiffest eigenparameter.

QUT
BRISBANE
AUSTRALIA

# 1. Model-data calibration using Bayesian and frequentist approaches

Let's say we have some data... and a proposed model.

# Notation!

- Data is in a vector $\boldsymbol{y}_{\text{obs}} = \begin{bmatrix} y_{\text{obs},1} \\ y_{\text{obs},2} \\ \vdots \\ y_{\text{obs},N_{\text{obs}}} \end{bmatrix}$. The length of $\boldsymbol{y}_{\text{obs}}$ is $N_{\text{obs}}$.
- Each data observation $y_{\text{obs},k}$ is associated with some inputs $\boldsymbol{x}_{\text{obs},k}$.
- Model is a function $\boldsymbol{y}_{\text{model}}(\boldsymbol{x}, \boldsymbol{\theta})$.
- The model has parameters in a vector $\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{N_\theta} \end{bmatrix}$.
- All parameters in $\boldsymbol{\theta}$ will be estimated by model-data calibration. The length of $\boldsymbol{\theta}$ is $N_\theta$.

# Example: Fitting a deterministic model of logistic growth to data.

**Data:**

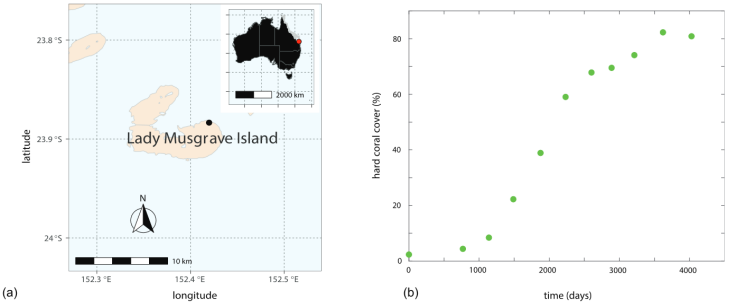



Figure: Hard coral cover vs time at a site in the Great Barrier Reef. Figure from Simpson et al. (2022) *J. Theor. Biol.* **535**: 110998.

**Model:**

$$\frac{\mathrm{d}C}{\mathrm{d}t} = rC\left(1 - \frac{C}{K}\right),$$

$$C(0) = C_0,$$

where $C$ is hard coral cover, $t$ is time, $r$ is the growth rate, $K$ is the carrying capacity, and $C_0$ is the initial hard coral cover.

**Question:** What is $\boldsymbol{y}_{\mathrm{obs}}$, $N_{\mathrm{obs}}$, $\boldsymbol{x}_{\mathrm{obs}}$, $\boldsymbol{y}_{\mathrm{model}}(\boldsymbol{x}, \boldsymbol{\theta})$, $\boldsymbol{\theta}$ and $N_\theta$ in this example?

# Frequentist model-data calibration

Let's recast the calibration problem as an optimisation problem.

**Goal:** Make the elements of $\boldsymbol{y}_{\text{obs}}$ *as close as possible* to corresponding elements of $\boldsymbol{y}_{\text{model}}(\boldsymbol{x}_{\text{obs}}, \boldsymbol{\theta})$.

What should the parameter vector $\boldsymbol{\theta}$ be to achieve this goal?

Standard approach: *minimising least-squares*.

$$\boldsymbol{\theta}_{\text{best}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[ \sum_{k=1}^{N_{\text{obs}}} \left( y_{\text{obs},k} - y_{\text{model}}(\boldsymbol{x}_{\text{obs},k}, \boldsymbol{\theta}) \right)^2 \right]$$

This process is also called "minimising the residual sum of squares".

Every mathematical software program has a package available which attempts to do this!

e.g. MATLAB: fitnlm, R: nls, Python: lmfit

# Let's look at this a bit more probabilistically

Least-squares minimisation for model calibration can also be treated as "maximum likelihood estimation" (MLE). We will soon show how, but for now, let's rewrite

$$\boldsymbol{\theta}_{\text{best}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[ \sum_{k=1}^{N_{\text{obs}}} \left( y_{\text{obs},k} - y_{\text{model}}(\boldsymbol{x}_{\text{obs},k}, \boldsymbol{\theta}) \right)^2 \right]$$

as

$$\boldsymbol{\theta}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[ \sum_{k=1}^{N_{\text{obs}}} \left( y_{\text{obs},k} - y_{\text{model}}(\boldsymbol{x}_{\text{obs},k}, \boldsymbol{\theta}) \right)^2 \right]$$

To do this, we assume that the data $\boldsymbol{y}_{\text{obs}}$ and model $\boldsymbol{y}_{\text{model}}(\boldsymbol{x}, \boldsymbol{\theta})$ are statistically connected by

$$\boxed{\boldsymbol{y}_{\text{obs}} = \boldsymbol{y}_{\text{model}}(\boldsymbol{x}_{\text{obs}}, \boldsymbol{\theta}) + \varepsilon, \quad \text{where} \quad \varepsilon \sim \mathcal{N}(0, \sigma^2)}$$

where $\mathcal{N}(\mu, \sigma^2)$ is a Gaussian distribution with mean $\mu$ and standard deviation $\sigma$.

# Getting even more probabilistic

$$\boxed{\boldsymbol{y}_{\text{obs}} = \boldsymbol{y}_{\text{model}}(\boldsymbol{x}_{\text{obs}}, \boldsymbol{\theta}) + \varepsilon, \quad \text{where} \quad \varepsilon \sim \mathcal{N}(0, \sigma^2)}$$

With the above definition, we can define a *likelihood function* $\mathcal{L}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta})$ which is the probability (i.e. likelihood) that the data we have observed is obtained from a model with parameters $\boldsymbol{\theta}$.

The parameters $\boldsymbol{\theta}$ that *maximise* the likelihood function $\mathcal{L}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta})$ are the parameters that have the *highest probability* of being the true parameters of the model.

These "best-fitting" parameters are the maximum likelihood estimator (MLE), i.e. $\boldsymbol{\theta}_{\text{best}} \equiv \boldsymbol{\theta}_{\text{MLE}}$.

What is the likelihood function $\mathcal{L}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta})$ whose maximum is at $\boldsymbol{\theta}_{\text{MLE}}$?

# Deriving the likelihood function

$$\boxed{\boldsymbol{y}_{\text{obs}} = \boldsymbol{y}_{\text{model}}(\boldsymbol{x}_{\text{obs}}, \boldsymbol{\theta}) + \varepsilon, \quad \text{where} \quad \varepsilon \sim \mathcal{N}(0, \sigma^2)}$$

Let's rewrite this for each element $y_{\text{obs},k}$ of $\boldsymbol{y}_{\text{obs}}$ as:

$$y_{\text{obs},k} = y_{\text{model}}(\boldsymbol{x}_{\text{obs},k}, \boldsymbol{\theta}) + \varepsilon, \quad \text{where} \quad \varepsilon \sim \mathcal{N}(0, \sigma^2)$$

and simplify this slightly to:

$$y_{\text{obs},k} \sim \mathcal{N}\left(y_{\text{model}}(\boldsymbol{x}_{\text{obs},k}, \boldsymbol{\theta}), \sigma^2\right)$$

Since the probability density function for a Gaussian distribution for $x \sim \mathcal{N}(\mu, \sigma^2)$ is (according to Wikipedia!):

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

we have that

$$\mathcal{L}(y_{\text{obs},k}|\boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{y_{\text{obs},k} - y_{\text{model}}(\boldsymbol{x}_{\text{obs},k}, \boldsymbol{\theta})}{\sigma}\right)^2\right)$$

# Deriving the likelihood function (continued)

$$\mathcal{L}(y_{\text{obs},k}|\boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{y_{\text{obs},k} - y_{\text{model}}(\boldsymbol{x}_{\text{obs},k}, \boldsymbol{\theta})}{\sigma}\right)^2\right)$$

If all observations $y_{\text{obs},k}$ in our dataset are independent of each other then:

$$\begin{aligned}\mathcal{L}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta}) &= \mathcal{L}(y_{\text{obs},1}|\boldsymbol{\theta}) \times \mathcal{L}(y_{\text{obs},2}|\boldsymbol{\theta}) \times ... \times \mathcal{L}(y_{\text{obs},N_{\text{obs}}}|\boldsymbol{\theta}) \\ &= \prod_{k=1}^{N_{\text{obs}}} \mathcal{L}(y_{\text{obs},k}|\boldsymbol{\theta})\end{aligned}$$

so

$$\boxed{\mathcal{L}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta}) = \prod_{k=1}^{N_{\text{obs}}} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{y_{\text{obs},k} - y_{\text{model}}(\boldsymbol{x}_{\text{obs},k}, \boldsymbol{\theta})}{\sigma}\right)^2\right)}$$

We now have the likelihood function!!! So how is minimising least squares equivalent to maximising the likelihood function?

# Take the natural logarithm of the Gaussian likelihood function!

$$\mathcal{L}\left(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta}\right) = \prod_{k=1}^{N_{\text{obs}}} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{y_{\text{obs},k} - y_{\text{model}}\left(\boldsymbol{x}_{\text{obs},k}, \boldsymbol{\theta}\right)}{\sigma}\right)^2\right)$$

$$\begin{aligned}
\log \mathcal{L}\left(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta}\right) &= \log\left[\prod_{k=1}^{N_{\text{obs}}} \frac{1}{\sqrt{2\pi}\sigma}\right] + \log\left[\prod_{k=1}^{N_{\text{obs}}} \exp\left(-\frac{1}{2}\left(\frac{y_{\text{obs},k} - y_{\text{model}}\left(\boldsymbol{x}_{\text{obs},k}, \boldsymbol{\theta}\right)}{\sigma}\right)^2\right)\right] \\
&= \sum_{k=1}^{N_{\text{obs}}} \log\left[\frac{1}{\sqrt{2\pi}\sigma}\right] + \sum_{k=1}^{N_{\text{obs}}} \log\left[\exp\left(-\frac{1}{2}\left(\frac{y_{\text{obs},k} - y_{\text{model}}\left(\boldsymbol{x}_{\text{obs},k}, \boldsymbol{\theta}\right)}{\sigma}\right)^2\right)\right] \\
&= N_{\text{obs}} \log\left[\frac{1}{\sqrt{2\pi}\sigma}\right] + \sum_{k=1}^{N_{\text{obs}}} \left(-\frac{1}{2}\left(\frac{y_{\text{obs},k} - y_{\text{model}}\left(\boldsymbol{x}_{\text{obs},k}, \boldsymbol{\theta}\right)}{\sigma}\right)^2\right)
\end{aligned}$$

$$\therefore \quad \boxed{\log \mathcal{L}\left(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta}\right) = -\frac{1}{2} N_{\text{obs}} \log 2\pi - \frac{1}{2} N_{\text{obs}} \log \sigma^2 - \frac{1}{2} \sum_{k=1}^{N_{\text{obs}}} \left(\frac{y_{\text{obs},k} - y_{\text{model}}\left(\boldsymbol{x}_{\text{obs},k}, \boldsymbol{\theta}\right)}{\sigma}\right)^2}$$

# Take the natural logarithm of the Gaussian likelihood function (continued)

$$\therefore \quad \boxed{\log \mathcal{L}\left(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta}\right) = -\frac{1}{2}N_{\text{obs}}\log 2\pi - \frac{1}{2}N_{\text{obs}}\log\sigma^2 - \frac{1}{2}\sum_{k=1}^{N_{\text{obs}}}\left(\frac{y_{\text{obs},k} - y_{\text{model}}(\boldsymbol{x}_{\text{obs},k},\boldsymbol{\theta})}{\sigma}\right)^2}$$

Notice that the first two terms are independent of the parameters $\boldsymbol{\theta}$ of the model, so:

$$\log \mathcal{L}\left(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta}\right) = -\frac{1}{2}\sum_{k=1}^{N_{\text{obs}}}\left(\frac{y_{\text{obs},k} - y_{\text{model}}(\boldsymbol{x}_{\text{obs},k},\boldsymbol{\theta})}{\sigma}\right)^2 + \text{ constant}$$

If we are only interested in the values of $\boldsymbol{\theta}$ which maximise the above expression, we can ignore the constant completely, so maximising the likelihood function is equivalent to *minimising the residual sum of squares*!

$$\begin{aligned}\boldsymbol{\theta}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\text{argmin}}\left[\sum_{k=1}^{N_{\text{obs}}}\left(y_{\text{obs},k} - y_{\text{model}}(\boldsymbol{x}_{\text{obs},k},\boldsymbol{\theta})\right)^2\right] &= \underset{\boldsymbol{\theta}}{\text{argmax}}\left[\log \mathcal{L}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta})\right] \\ &= \underset{\boldsymbol{\theta}}{\text{argmax}}\left[\mathcal{L}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta})\right]\end{aligned}$$

# Summarising the story so far

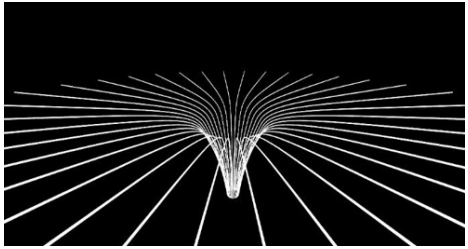
- Model-data calibration can be recast as an optimisation problem: finding the best-fit parameters $\boldsymbol{\theta}_{\mathrm{best}}$.
- These best-fit parameters can be found by least-squares minimisation of the difference between the data $\boldsymbol{y}_{\mathrm{obs}}$ and the model $\boldsymbol{y}_{\mathrm{model}}(\boldsymbol{x}, \boldsymbol{\theta})$ evaluated at input conditions $\boldsymbol{x} = \boldsymbol{x}_{\mathrm{obs}}$.
- The best-fit parameters $\boldsymbol{\theta}_{\mathrm{best}}$ are also called the maximum likelihood estimator (MLE) $\boldsymbol{\theta}_{\mathrm{MLE}}$, because minimising least-squares is equivalent to maximising an appropriately defined likelihood function $\mathcal{L}(\boldsymbol{y}_{\mathrm{obs}}|\boldsymbol{\theta})$.
- Every mathematical software program has a package available which attempts to do this, e.g. MATLAB: fitnlm, R: nls, Python: lmfit.

Let's do some examples of this in Python!

**If you blitz the worked examples in Python without any problems, have a go at doing a frequentist model-data calibration for a different dataset and model of your choice.**

# Should finding the "best-fit" parameters be the end of the story?

- What we have discussed so far is a *frequentist* approach to model-data calibration, where the key goal is to find the "best-fit model parameters".
- On a more practical computational level, least-squares optimisation in mathematical software packages can get "stuck" in local maxima of the likelihood function, or more generally get confused in there is a parameter identifiability issue.



# From frequentist to Bayesian

Bayesian model-data calibration is slightly different: it acknowledges explicitly that there may be no clear "best" parameters $\boldsymbol{\theta}_{\mathrm{MLE}}$, and a number of different parameter sets $\{\boldsymbol{\theta}_m\}_{m=1}^{M}$ may plausibly fit the data well.

However, we need a bit more terminology, and a few more (chosen) assumptions.

| | **Frequentist** | **Bayesian** |
|---|---|---|
| **Inputs:** | Data<br>Model<br>Likelihood function | Data<br>Model<br>Likelihood function<br>Prior distribution |
| **Outputs:** | Best-fit parameters | Posterior distribution |

What is the prior distribution? What is the posterior distribution?

# What is the prior distribution?

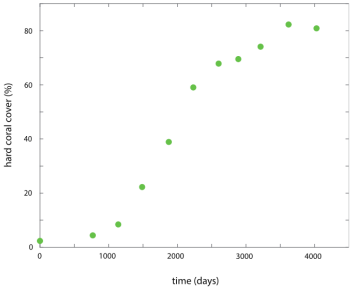
Your *initial beliefs* $p(\boldsymbol{\theta})$ about what you think the parameters $\boldsymbol{\theta}$ are, BEFORE looking at the data.

These beliefs cannot be "I don't know". (Chicken-egg problem!)

These beliefs need to be captured in a probability distribution that *integrates sensibly to one*.

Reconsider the coral model from before.

**Data:**




**Model:**

$$\frac{\mathrm{d}C}{\mathrm{d}t} = rC\left(1 - \frac{C}{K}\right), \quad C(0) = C_0,$$

where $C$ is hard coral cover, $t$ is time, $r$ is the growth rate, $K$ is the carrying capacity, and $C_0$ is the initial hard coral cover.

**Prior:**

We expect $r > 0$, $K > 0$ and $C_0 > 0$. But this is not enough for Bayesian inference.

# Why is a belief that the parameters are positive still not enough?

Remember: we need the prior distribution to *integrate sensibly to one*.

Let's consider the growth rate parameter $r > 0$ as an example.

If we are to construct a probability distribution for $r$, we might first assume that $r$ could take *any* **positive** value with **equal** probability, i.e.

$$p(r) \propto \begin{cases} k, & \text{if } r > 0, \\ 0, & \text{otherwise,} \end{cases}$$

where $k$ is a yet-to-be defined constant.

- If we set $k$ to be zero, then $\int_{-\infty}^{\infty} p(r)\,\mathrm{d}r = 0$ which is not one!
- If we set $k$ to be a finite positive number, then $\displaystyle\int_{-\infty}^{\infty} p(r)\,\mathrm{d}r = \int_{0}^{\infty} k\,\mathrm{d}r = [kr]_{r=0}^{\infty} = \infty.$

We need something more specific!

# A simple, working, prior distribution: Uniform, doubly-bounded, priors!

We could assume that all parameters could take any value with equal probability within finite bounds for each parameter, e.g. $0 < r < r_{\max}$, $0 < K < K_{\max}$, and $0 < C_0 < C_{0,\max}$, where we choose $r_{\max}$, $K_{\max}$ and $C_{0,\max}$ to be sufficiently large so that they are *definitely* larger than what we expect plausible values of $r$, $K$ and $C_0$ to be.

Thus, the prior distribution $p(\boldsymbol{\theta})$ for parameters here $\boldsymbol{\theta}$ is:

$$p(\boldsymbol{\theta}) = p(r)p(K)p(C_0), \quad \text{where} \quad p(r) = \begin{cases} 1/r_{\max}, & \text{if } 0 < r < r_{\max}, \\ 0, & \text{otherwise}, \end{cases}$$

$$p(K) = \begin{cases} 1/K_{\max}, & \text{if } 0 < K < K_{\max}, \\ 0, & \text{otherwise}, \end{cases} \qquad p(C_0) = \begin{cases} 1/C_{0,\max}, & \text{if } 0 < C_0 < C_{0,\max}, \\ 0, & \text{otherwise}. \end{cases}$$

In this case we say that $r$, $K$ and $C_0$ follow *uniform* prior distributions $\mathcal{U}(0, r_{\max})$, $\mathcal{U}(0, K_{\max})$ and $\mathcal{U}(0, C_{0,\max})$.

# What is the posterior distribution?

This is what Bayesian inference aims to find. It is the *final beliefs* $p(\boldsymbol{\theta}|\boldsymbol{y}_{\text{obs}})$ about what you think the parameters $\boldsymbol{\theta}$ are, AFTER looking at the data $\boldsymbol{y}_{\text{obs}}$.

Unlike frequentist model-data calibration, which aims to find "best-fit" parameter values $\boldsymbol{\theta}_{\text{best}} = \boldsymbol{\theta}_{\text{MLE}}$, Bayesian model-data calibration aims to estimate a "probability distribution" $p(\boldsymbol{\theta}|\boldsymbol{y}_{\text{obs}})$ that best captures a plausible range of possible parameter values $\boldsymbol{\theta}$ that fits the data.

Typically this probability distribution is not calculated exactly, but we instead aim to obtain *samples* from it.

- If $p(\boldsymbol{\theta}|\boldsymbol{y}_{\text{obs}})$ is a tightly-constrained distribution with most of the density of this distribution focused in a small area of parameter space $\boldsymbol{\theta}$, then we can be very confident about what the parameter values $\boldsymbol{\theta}$ might be!
- If $p(\boldsymbol{\theta}|\boldsymbol{y}_{\text{obs}})$ is a more messy or "smooshed-out" distribution, we may not be confident at all about what the model parameters are, even after calibrating our model to the available data!

# How do we calculate the posterior distribution?

If we have defined a prior distribution $p(\boldsymbol{\theta})$ and a likelihood function $\mathcal{L}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta})$ (this can be the same likelihood function that is used in minimising least squares!) then the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{y}_{\text{obs}})$ is obtained these via Bayes' theorem:

$$p(\boldsymbol{\theta}|\boldsymbol{y}_{\text{obs}}) \propto p(\boldsymbol{\theta})\ \mathcal{L}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta})$$

Oh so it's just a product. Seems simple enough right? WRONG.

Computing the product above is a nightmare for real models. Hence why we aim for samples instead.

Bayesian statisticians are constantly coming up with new algorithms for approximating Bayes' theorem: MCMC, SMC, variational Bayes', Laplace approximation, ...

You've interacted with the "SMC Down Under" event yesterday – this event is a whole week devoted to SMC!

# Summarising the (Bayesian) story so far

- *Frequentist* model-data calibration is about finding the best-fit parameters $\boldsymbol{\theta}_{\text{best}} = \boldsymbol{\theta}_{\text{MLE}}$.
- *Bayesian* model-data calibration has a philosophically different goal: finding the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{y}_{\text{obs}})$.
- Bayesian model-data calibration explicitly recognises that the data may (or may not) be sufficient to yield confident estimates for parameters $\boldsymbol{\theta}$ of our model.
- For Bayesian model-data calibration, we need to define a prior distribution $p(\boldsymbol{\theta})$ that integrates sensibly to one.

Let's talk a little bit about a SMC algorithm we can use,
and then have a go at using it in Python!

# All you really need to know about the SMC algorithm we will be using!

- The parameter $\gamma$ (gamma) indicates where the algorithm is up to.
- A small value of $\gamma > 0$ means it is near the *start* of the algorithm.
- $\gamma$ increases adaptively throughout the algorithm, usually at an exponential rate!
- $\gamma = 1$ means the algorithm has finished.

```
SMC algorithm is in progress: The current value of gamma is 0.4133209837134866 . Algorithm concludes when gamma = 1.
SMC algorithm is in progress: The current value of gamma is 0.43889464215734186 . Algorithm concludes when gamma = 1.
SMC algorithm is in progress: The current value of gamma is 0.5184561301322355 . Algorithm concludes when gamma = 1.
SMC algorithm is in progress: The current value of gamma is 0.5705573751771703 . Algorithm concludes when gamma = 1.
SMC algorithm is in progress: The current value of gamma is 0.6167687690788846 . Algorithm concludes when gamma = 1.
SMC algorithm is in progress: The current value of gamma is 0.6628745222557269 . Algorithm concludes when gamma = 1.
SMC algorithm is in progress: The current value of gamma is 0.7715615571042858 . Algorithm concludes when gamma = 1.
SMC algorithm is in progress: The current value of gamma is 0.8428387711477583 . Algorithm concludes when gamma = 1.
SMC algorithm is in progress: The current value of gamma is 0.9053365947215604 . Algorithm concludes when gamma = 1.
SMC algorithm is in progress: The current value of gamma is 0.9735882512586018 . Algorithm concludes when gamma = 1.
SMC algorithm is in progress: The current value of gamma is 1 . Algorithm concludes when gamma = 1.
The Sequential Monte Carlo algorithm estimates that the mean values of your model parameters are:
[1.06919769e+00 8.22780846e+01 2.78128360e-03 8.48802556e+00]
The Sequential Monte Carlo algorithm estimates that the standard deviations of your model parameters are:
[6.84792571e-01 6.85007408e+00 1.41828328e-03 1.30351511e+01]
```

# What does $\gamma$ (gamma) mean?

At each iteration $s$ of the algorithm, there is a specific value of $\gamma = \gamma_s$.

The parameter sets are being compared to an *intermediate distribution* given by

$$\boxed{\pi_s(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta})\ \mathcal{L}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta})^{\gamma_s}}$$

Recall Bayes' theorem is

$$p(\boldsymbol{\theta}|\boldsymbol{y}_{\text{obs}}) \propto p(\boldsymbol{\theta})\ \mathcal{L}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta})$$

When $\gamma_s = 0$, $\pi_s(\boldsymbol{\theta}) \to p(\boldsymbol{\theta})$. This is the prior!

When $\gamma_s = 1$, $\pi_s(\boldsymbol{\theta}) \to p(\boldsymbol{\theta}|\boldsymbol{y}_{\text{obs}})$. This is the posterior!

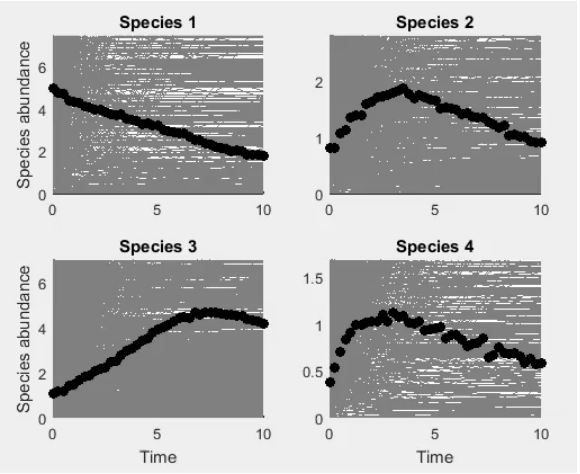
So the SMC algorithm *starts* at the prior, and slowly creeps towards the posterior as it goes on!

# What does the SMC algorithm actually do? And what does it look like?

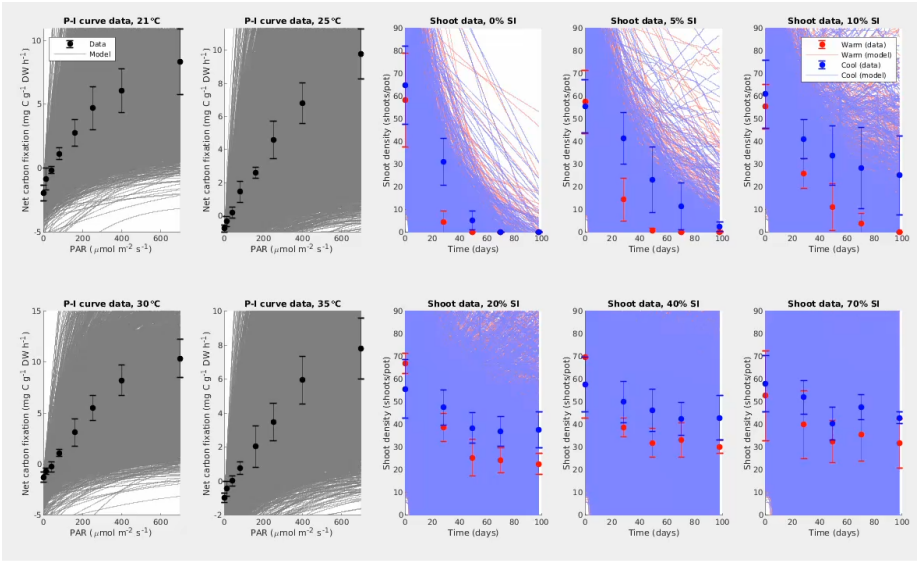
It has $m = 1, ..., M$ parameter sets $\boldsymbol{\theta}_m$, which it tracks through the algorithm. These parameter sets change their *values* as the algorithm progresses.

**The algorithm does three things in a cycle:**

- Deleting the locations of "bad" parameter sets, and replacing them with "duplicates" of good parameter sets (technically: resampling).
- Changing the parameter sets' values a *little* bit (technically: mutation).
- Changing how much each parameter set is "liked" by the algorithm (technically: reweighting).



# One more SMC video... now let's do some Python examples!



# The SMC algorithm we are using: more details (part 1 of 2)

*Resampling:*

- For $m = 1, ..., M$, generate $\boldsymbol{\theta}_m^0$ from $p(\boldsymbol{\theta})$, and set weights $W_m^0 = 1/M$.
- Set $\gamma_0 = 0$ and effective sample size $\text{ESS}_0 = M$.
- Set iteration $s = 1$, effective sample size threshold for resampling $E = M/2$, and effective sample size reduction target $\Delta = 0.05$.

*Reweighting:*

- If setting $\gamma_s = 1$ yields $\text{ESS}_s \geq (1 - \Delta)\text{ESS}_{s-1}$, set $\gamma_s = 1$. Otherwise, for the current set of particles, use the bisection method to identify the value of $\gamma_s$ that yields $\text{ESS}_s = (1 - \Delta)\text{ESS}_{s-1}$, given that we expect $\gamma_{s-1} < \gamma_s < 1$.
- For $m = 1, ..., M$, set $\boldsymbol{\theta}_m^s = \boldsymbol{\theta}_m^{s-1}$, and calculate weights $w_m^s = W_m^{s-1}\pi_s(\boldsymbol{\theta}_m^s)/\pi_{s-1}(\boldsymbol{\theta}_m^{s-1})$.
- Normalise the weights so that $W_m^s = w_m^s / \sum_{m=1}^M w_m^s$ for $m = 1, ..., M$.
- If $\gamma_s = 1$, perform a resampling step (see next slide) and end the SMC algorithm.

# The SMC algorithm we are using: more details (part 2 of 2)

*Resampling:*

- Calculate the effective sample size, $\text{ESS} = 1/\sum_{m=1}^{M}(W_m^s)^2$.
- If $\text{ESS} < E$, resample particles from $\{\boldsymbol{\theta}_m^s, W_m^s\}$, and set $W_m^s = 1/M$ for $m = 1, ..., M$.

*Mutation:*

- If this is the first mutation step $r$ within the current iteration $s$, set $r = 1$.
- For $m = 1, ..., M$, generate the proposed particle $\boldsymbol{\theta}_m^* \sim q_{s,r}(\boldsymbol{\theta}|\boldsymbol{\theta}_m^s)$.
- For $m = 1, ..., M$, with probability $\min\left\{1, \frac{\pi_s(\boldsymbol{\theta}_m^*|\boldsymbol{y})q_{s,r}(\boldsymbol{\theta}_m^s|\boldsymbol{\theta}_m^*)}{\pi_s(\boldsymbol{\theta}_m^s|\boldsymbol{y})q_{s,r}(\boldsymbol{\theta}_m^*|\boldsymbol{\theta}_m^s)}\right\}$, set $\boldsymbol{\theta}_m^s = \boldsymbol{\theta}_m^*$, else leave $\boldsymbol{\theta}_m^s$ unchanged.
- If less than 95% of the $M$ particles have changed their position at the current iteration $s$, set $r = r + 1$ and repeat the *Mutation* step. Otherwise, increment $s = s + 1$ and go to *Reweighting*.

QUT
BRISBANE
AUSTRALIA

# 2. Visualisation of calibration outputs

# Overview

Now we are going to do some visualisation!

1. Confidence interval predictions (MLE)
2. Median and credible interval predictions (Bayesian inference)
3. Parameter histograms
4. Marginal distributions (using kernel density estimation)
5. Bivariate scatter plots (posterior vs prior)

# [1] Confidence interval predictions (MLE)

Many software packages that do nonlinear regression using least-squares fitting have an ability to produce confidence intervals.

There are generally two types:

1. Confidence intervals for the uncertainty in the *best-fit predictions* of the model.
2. Confidence intervals for where a "new" data point is likely to be observed, given your model.

The first type has narrower confidence bounds, the second type has broader confidence bounds.

Which one you report depends on the objectives of your study, and your assumptions about the source of the "noise" in your data, relative to your model.

# [1] Confidence interval predictions (MLE) continued

In MATLAB these two types are differentiated by the optional 'Prediction' option used when MATLAB's *predict* function is used.

- If 'Prediction' is set to 'curve' (default), it is the first type: confidence intervals for the uncertainty in the best-fit predictions.
- If 'Prediction' is set to 'observation', it is the second type: confidence intervals for where a new data point is likely to observed.

We will show one way of getting the *first type* in Python, without going into the maths of how they are obtained.

We will then return to these concepts later when constructing similar bounds for Bayesian inference (as the mathematics behind them are easier to explain explicitly).

# [1] Confidence interval predictions (MLE) continued

Remember:

$$\pm 1\sigma \approx 68\% \text{ confidence interval,}$$

$$\pm 2\sigma \approx 95\% \text{ confidence interval,}$$

$$\pm 3\sigma \approx 99.7\% \text{ confidence interval.}$$

Let's now do this in Python!

# [2] Median and credible interval predictions (Bayesian inference)

Reminder: We have $M = 1000$ different model-data fits, all of which have equal probabilistic weight!

We no longer talk about *confidence* intervals (CI), as we are no longer in frequentist statistics.

Instead we talk about *credible* intervals (CI).

However, we can still borrow concepts from frequentist statistics, e.g. by focusing on:

- Median prediction (loosely $\approx$ mean)
- 68% central credible interval (loosely $\approx \pm 1\sigma$)
- 95% central credible interval (loosely $\approx \pm 2\sigma$)

# [2] Median and credible interval predictions (Bayesian inference)

- Median prediction (loosely $\approx$ mean)
- 68% central credible interval (loosely $\approx \pm 1\sigma$)
- 95% central credible interval (loosely $\approx \pm 2\sigma$)

It's going to be easier to think about these 3 concepts in terms of "percentiles" across our simulations.

- Median = 50th percentile
- 68% central credible interval = bounded between 16th percentile and 84th percentile
- 95% central credible interval = bounded between 2.5th percentile and 97.5th percentile

So our task now is to find 5 percentiles across our ensemble of $M = 1000$ members:

$$\Rightarrow \text{2.5th, 16th, 50th, 84th, 97.5th percentiles.}$$

# [2] Median and credible interval predictions (Bayesian inference)

- Median = 50th percentile
- 68% central credible interval = bounded between 16th percentile and 84th percentile
- 95% central credible interval = bounded between 2.5th percentile and 97.5th percentile

So our task now is to find 5 percentiles across our ensemble of $M = 1000$ members:

$$\Rightarrow \text{2.5th, 16th, 50th, 84th, 97.5th percentiles.}$$

Doing this for the 1000 ensemble members is *loosely equivalent* to the mean, $\pm 1\sigma$ and $\pm 2\sigma$ confidence intervals we previously calculated in Python from MLE.

Let's do this now in Python for our ensemble from Bayesian inference!

# [2] Median and credible interval predictions (Bayesian inference)

We've now calculated credible intervals for the uncertainty in the best-fitting models.

What about the credible intervals for where a "new" data point is likely to observed, given your model? This is a different credible interval!

Recall that we have assumed the data $\boldsymbol{y}_{\text{obs}}$ and model $\boldsymbol{y}_{\text{model}}(\boldsymbol{x}, \boldsymbol{\theta})$ are statistically connected by

$$\boxed{\boldsymbol{y}_{\text{obs}} = \boldsymbol{y}_{\text{model}}(\boldsymbol{x}_{\text{obs}}, \boldsymbol{\theta}) + \varepsilon, \quad \text{where} \quad \varepsilon \sim \mathcal{N}(0, \sigma^2)}$$

where $\mathcal{N}(\mu, \sigma^2)$ is a Gaussian distribution with mean $\mu$ and standard deviation $\sigma$.

The credible intervals we calculated before in Python described the uncertainty in $\boldsymbol{y}_{\text{model}}(\boldsymbol{x}, \boldsymbol{\theta})$.

Our task now is to calculate the uncertainty in $\boldsymbol{y}_{\text{obs}} = \boldsymbol{y}_{\text{model}}(\boldsymbol{x}, \boldsymbol{\theta}) + \varepsilon$...

We had $M = 1000$ values of $\boldsymbol{y}_{\text{model}}(\boldsymbol{x}, \boldsymbol{\theta})$ for each $\boldsymbol{x}$, all equally weighted so that was easy.

But there are an infinite number of values of $\varepsilon$ since it follows a Gaussian distribution!

# [2] Median and credible interval predictions (Bayesian inference)

Our task now is to calculate the uncertainty in $\boldsymbol{y}_{\text{obs}} = \boldsymbol{y}_{\text{model}}(\boldsymbol{x}, \boldsymbol{\theta}) + \varepsilon$...

Let's approximate $\infty$ by $M = 1000$, and invoke the following procedure.

For $m = 1, 2, ..., M$:

1. Calculate $\boldsymbol{y}_{\text{model}}(\boldsymbol{x}, \boldsymbol{\theta}_m)$ for all input conditions $\boldsymbol{x}$ of interest.
2. Sample $\varepsilon \sim \mathcal{N}(0, \sigma_m)$ and save this sampled value temporarily as $\varepsilon_m$.
3. Add $\varepsilon_m$ from Step 2 to each $\boldsymbol{y}_{\text{model}}(\boldsymbol{x}, \boldsymbol{\theta}_m)$ in Step 1. Treat $y_{\text{model}}(\boldsymbol{x}_k, \boldsymbol{\theta}_m) + \varepsilon_m$ as the $m$th sample of a potential new observation $y_{\text{obs},k}$ measured at input conditions $\boldsymbol{x}_k$.
4. Store all $M$ samples of $\boldsymbol{y}_{\text{obs}}$, for all input conditions $\boldsymbol{x}$ of interest.

Voila! We now have $M = 1000$ samples of potential new data points for each $\boldsymbol{x}$ of interest! We've been tricksy here too: Since one value of $\varepsilon_m$ is used across all $\boldsymbol{x}$ for one sample $m$, we will also reduce the amount of noisy "wiggles" in our credible interval plots.

# [2] Median and credible interval predictions (Bayesian inference)

For $m = 1, 2, ..., M$:

1. Calculate $\boldsymbol{y}_{\text{model}}(\boldsymbol{x}, \boldsymbol{\theta}_m)$ for all input conditions $\boldsymbol{x}$ of interest.
2. Sample $\varepsilon \sim \mathcal{N}(0, \sigma_m)$ and save this sampled value temporarily as $\varepsilon_m$.
3. Add $\varepsilon_m$ from Step 2 to each $\boldsymbol{y}_{\text{model}}(\boldsymbol{x}, \boldsymbol{\theta}_m)$ in Step 1. Treat $y_{\text{model}}(\boldsymbol{x}_k, \boldsymbol{\theta}_m) + \varepsilon_m$ as the $m$th sample of a potential new observation $y_{\text{obs},k}$ measured at input conditions $\boldsymbol{x}_k$.
4. Store all $M$ samples of $\boldsymbol{y}_{\text{obs}}$, for all input conditions $\boldsymbol{x}$ of interest.

Technically, the above algorithm is an example of *Monte Carlo simulation*:

- When we can't *analytically* calculate the probability distribution of a desired variable (e.g. the distribution of potential new observations $y_{\text{obs}}$), but we *can* sample from it,
- Sample the variable a large number of times, and
- Assume that your sample approximates the distribution that you wanted to calculate!

# [2] Median and credible interval predictions (Bayesian inference)

For $m = 1, 2, ..., M$:

1. Calculate $\boldsymbol{y}_{\text{model}}(\boldsymbol{x}, \boldsymbol{\theta}_m)$ for all input conditions $\boldsymbol{x}$ of interest.
2. Sample $\varepsilon \sim \mathcal{N}(0, \sigma_m)$ and save this sampled value temporarily as $\varepsilon_m$.
3. Add $\varepsilon_m$ from Step 2 to each $\boldsymbol{y}_{\text{model}}(\boldsymbol{x}, \boldsymbol{\theta}_m)$ in Step 1. Treat $y_{\text{model}}(\boldsymbol{x}_k, \boldsymbol{\theta}_m) + \varepsilon_m$ as the $m$th sample of a potential new observation $y_{\text{obs},k}$ measured at input conditions $\boldsymbol{x}_k$.
4. Store all $M$ samples of $\boldsymbol{y}_{\text{obs}}$, for all input conditions $\boldsymbol{x}$ of interest.

Recall:

- Median = 50th percentile
- 68% central credible interval = bounded between 16th percentile and 84th percentile
- 95% central credible interval = bounded between 2.5th percentile and 97.5th percentile

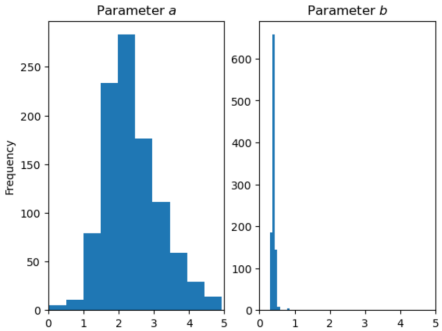
So our task is *again* to find 5 percentiles across our ensemble of $M = 1000$ members: 2.5th, 16th, 50th, 84th, 97.5th, but *now* for $\boldsymbol{y}_{\text{obs}}$ instead of $\boldsymbol{y}_{\text{model}}$. **Let's do this now in Python!**

# [3] Parameter histograms

A first way to check what the distribution of parameter values estimated by Bayesian inference is to construct a histogram of the values of all the parameters from each of the posterior samples.

**Note:** This is only appropriate if all posterior samples are equally weighted.

**Also:** If your priors are uniform, there is value is setting the $x$-axis limits of your histograms to equal the *bounds* of your priors, as this helps to see how much information the data has given you about each parameter's value.




Let's do this now in Python!

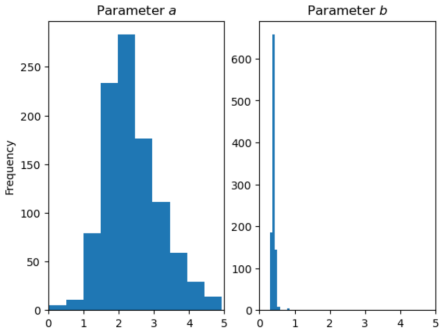
# [4] Marginal distributions (using kernel density estimation)

Parameter histograms are easy to interpret, but they are a bit blocky. Also, the information we get from them depends on the bin width:

- If the bin width is too big, we can't see any patterns.
- If the bin width is too small we end up a bunch of 1's and 0's!

**Solution:** Kernel density estimation (KDE).

Kernel density estimation is a way of smoothing out the histograms for each parameter $\theta_i \in \boldsymbol{\theta}$ so that we get an approximation of each parameter $\theta_i$'s posterior probability distribution $p(\theta_i|\boldsymbol{y}_{\text{obs}})$. In Bayesian inference, these distributions are called "marginal distributions".



# [4] Marginal distributions (using kernel density estimation)

There are a number of different mathematical ways to do kernel density estimation, we are not going to cover that here. Let's go with default settings (to start with).

In Python, we can use, e.g. *stats.gaussian_kde* from scipy, or *density* from kalepy.

In MATLAB, we can use the function *ksdensity*.

All of these functions use a Gaussian kernel as the default, which means they have non-zero density from $-\infty$ to $\infty$. *This is a problem (!)* when we are attempting to approximate a distribution which has strict lower and upper bounds (e.g. due to a uniform prior distribution)!

We will show how this issue can be resolved by using a boundary correction called "reflection". This method takes the density outside the boundaries and moves it inside the boundaries, treating the boundaries a bit like a mirror.

Let's do this now in Python!

# [4] Marginal distributions (using kernel density estimation)

Now armed with a method of approximating the marginal distributions of model parameters – kernel density estimation using a reflection boundary correction – let's look at all parameters of our model and compare

Best-fit parameter values from maximum likelihood estimation **versus** marginal distributions from Bayesian inference!

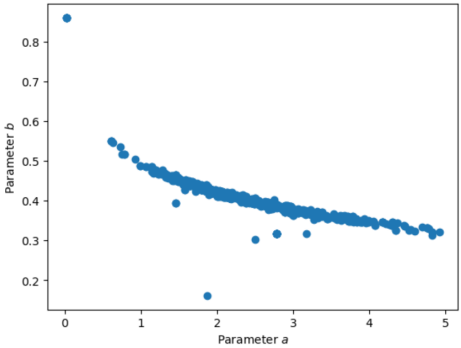
(It's Python time again!)

# [5] Bivariate scatter plots (posterior vs prior)

The last tool in our visualisation arsenal (pre-sloppiness) is a bivariate scatter plot.

These plots are literally plotting one parameter $\theta_i$ on the $x$-axis against another parameter $\theta_j$ on the $y$-axis. We then take note of any trends that we see.

These plots tell us how parameter values are correlated with each other!




Let's create some bivariate scatter plots in Python!

However, what if the relationships are more complicated than this? What if *the most important correlations* between parameters are between three, four, or more parameters?

This is one of the motivations for analysing sloppiness, which we will get to in the next module!

QUT
BRISBANE
AUSTRALIA

# 3. Analysis of model sloppiness (Part 1)

# Overview of the mathematical process of analysing sloppiness

1. Construct a sensitivity matrix of size $N_\theta \times N_\theta$. (We typically exclude $\sigma$ from the construction of this matrix.)
2. Find the eigenvalues $\lambda_j$ and eigenvectors $\boldsymbol{v}_j$ of the sensitivity matrix, where $j = 1, ..., N_\theta$.
3. Reorder the eigenvalues from largest to smallest, so that $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_{N_\theta}$, and reorder the eigenvectors accordingly.
4. If not already done, rescale eigenvalues $\lambda_j \leftarrow \lambda_j/\lambda_1$. This makes all $\lambda_j \leq 1$.
5. If not already done, rescale eigenvectors $\boldsymbol{v}_j \leftarrow \boldsymbol{v}_j/v_{j,\max}$, where $v_{j,\max}$ is the element of the eigenvector $\boldsymbol{v}_j$ possessing the largest absolute magnitude *within* this eigenvector. This makes all elements $v_{j,i}$ satisfy $-1 \leq v_{j,i} \leq 1$, and guarantees that at least one element of each $\boldsymbol{v}_j$ is exactly $+1$.
6. Report eigenparameters $\hat{\theta}_j = \theta_1^{v_{j,1}}\theta_2^{v_{j,2}}...\theta_{N_\theta}^{v_{j,N_\theta}}$, where $j = 1, ..., N_\theta$. (We typically remove factors $\theta_i^{v_{j,i}}$ in this product that possess $-0.2 < v_{j,i} < 0.2$.)

# Sensitivity matrices for frequentist model-data calibrations

Two sensitivity matrices proposed for analysing model sloppiness in frequentist model-data calibrations:

- Hessian matrix **H**.
- Levenberg-Marquardt Hessian matrix **L**.

Hessian matrix is more accurate, but requires more computations.

They are both trying to calculate the *same* sensitivity matrix, so if you are sure one of them is working, no need to do the other one!

*Aside:* You'll notice that these matrices, especially **H**, bear strong resemblance to the Fisher information matrix, although we won't focus on this connection too much here.

# The Hessian matrix: The "hammer approach"

Hessian matrix $\mathbf{H}$ evaluated at $\boldsymbol{\theta}_{\mathrm{MLE}}$, with elements $H_{ij}$ given by:

$$\boxed{H_{ij} = -\left.\frac{\partial^2 \log \mathcal{L}\left(\boldsymbol{y}_{\mathrm{obs}}|\boldsymbol{\theta}\right)}{\partial \log \theta_i \, \partial \log \theta_j}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{\mathrm{MLE}}}}$$

One computational approach for estimating this second derivative is finite differencing:

$$H_{ij} \approx \frac{-1}{\delta^2}\left[\log \mathcal{L}\left(\boldsymbol{y}_{\mathrm{obs}}|\boldsymbol{\theta}_{i\uparrow,j\uparrow}\right) - \log \mathcal{L}\left(\boldsymbol{y}_{\mathrm{obs}}|\boldsymbol{\theta}_{i\uparrow,j\downarrow}\right) - \log \mathcal{L}\left(\boldsymbol{y}_{\mathrm{obs}}|\boldsymbol{\theta}_{i\downarrow,j\uparrow}\right) + \log \mathcal{L}\left(\boldsymbol{y}_{\mathrm{obs}}|\boldsymbol{\theta}_{i\downarrow,j\downarrow}\right)\right]$$

where $\delta$ is a user-chosen small scalar, that acts "like" a step-size.

- Choosing $\delta$ to be *too big* will yield a bad approximation of $\mathbf{H}$.
- Choosing $\delta$ to be *too small* can also yield a bad approximation of $\mathbf{H}$ due to floating point errors in your software program.
- In practice we found that $\delta$ values of between $10^{-4}$ and $10^{-2}$ yield reasonable (and reproducible) results.

# The Hessian matrix: The "hammer approach" (continued)

$$H_{ij} \approx \frac{-1}{\delta^2}\left[\log \mathcal{L}\left(\boldsymbol{y}_{\rm obs}|\boldsymbol{\theta}_{i\uparrow,j\uparrow}\right) - \log \mathcal{L}\left(\boldsymbol{y}_{\rm obs}|\boldsymbol{\theta}_{i\uparrow,j\downarrow}\right) - \log \mathcal{L}\left(\boldsymbol{y}_{\rm obs}|\boldsymbol{\theta}_{i\downarrow,j\uparrow}\right) + \log \mathcal{L}\left(\boldsymbol{y}_{\rm obs}|\boldsymbol{\theta}_{i\downarrow,j\downarrow}\right)\right]$$

The log-likelihood functions in the above expression are evaluated at four parameter sets $\boldsymbol{\theta}_{i\uparrow,j\uparrow}$, $\boldsymbol{\theta}_{i\uparrow,j\downarrow}$, $\boldsymbol{\theta}_{i\downarrow,j\uparrow}$ and $\boldsymbol{\theta}_{i\downarrow,j\downarrow}$, which are all very close to $\boldsymbol{\theta}_{\rm MLE}$, but have two parameter values differing by a small value that depends on $\delta$.

- $\boldsymbol{\theta}_{i\uparrow,j\uparrow}$ is $\boldsymbol{\theta} = \boldsymbol{\theta}_{\rm MLE}$, except with: $\theta(i) \leftarrow \theta(i)\left(1+\frac{\delta}{2}\right)$, and $\theta(j) \leftarrow \theta(j)\left(1+\frac{\delta}{2}\right)$.
- $\boldsymbol{\theta}_{i\uparrow,j\downarrow}$ is $\boldsymbol{\theta} = \boldsymbol{\theta}_{\rm MLE}$, except with: $\theta(i) \leftarrow \theta(i)\left(1+\frac{\delta}{2}\right)$, and $\theta(j) \leftarrow \theta(j)\left(1-\frac{\delta}{2}\right)$.
- $\boldsymbol{\theta}_{i\downarrow,j\uparrow}$ is $\boldsymbol{\theta} = \boldsymbol{\theta}_{\rm MLE}$, except with: $\theta(i) \leftarrow \theta(i)\left(1-\frac{\delta}{2}\right)$, and $\theta(j) \leftarrow \theta(j)\left(1+\frac{\delta}{2}\right)$.
- $\boldsymbol{\theta}_{i\downarrow,j\downarrow}$ is $\boldsymbol{\theta} = \boldsymbol{\theta}_{\rm MLE}$, except with: $\theta(i) \leftarrow \theta(i)\left(1-\frac{\delta}{2}\right)$, and $\theta(j) \leftarrow \theta(j)\left(1-\frac{\delta}{2}\right)$.

# The Hessian matrix: The "hammer approach" (continued)

$$H_{ij} \approx \frac{-1}{\delta^2}\left[\log \mathcal{L}\left(\boldsymbol{y}_{\mathrm{obs}}|\boldsymbol{\theta}_{i\uparrow,j\uparrow}\right) - \log \mathcal{L}\left(\boldsymbol{y}_{\mathrm{obs}}|\boldsymbol{\theta}_{i\uparrow,j\downarrow}\right) - \log \mathcal{L}\left(\boldsymbol{y}_{\mathrm{obs}}|\boldsymbol{\theta}_{i\downarrow,j\uparrow}\right) + \log \mathcal{L}\left(\boldsymbol{y}_{\mathrm{obs}}|\boldsymbol{\theta}_{i\downarrow,j\downarrow}\right)\right]$$

Note that log-likelihood functions $\log \mathcal{L}\left(\boldsymbol{y}_{\mathrm{obs}}|\boldsymbol{\theta}\right)$, which are evaluated to calculate the Hessian matrix, contain dependence on the noise parameter $\sigma$ hidden within them. Recall that:

$$\log \mathcal{L}\left(\boldsymbol{y}_{\mathrm{obs}}|\boldsymbol{\theta}\right) = -\frac{1}{2}N_{\mathrm{obs}}\log 2\pi - \frac{1}{2}N_{\mathrm{obs}}\log \sigma^2 - \frac{1}{2}\sum_{k=1}^{N_{\mathrm{obs}}}\left(\frac{y_{\mathrm{obs},k} - y_{\mathrm{model}}(\boldsymbol{x}_{\mathrm{obs},k},\boldsymbol{\theta})}{\sigma}\right)^2$$

The first two terms pose no problems, because the same constant (dependent on $\sigma$) gets added and subtracted twice in the Hessian formula above. (These terms also disappear in the general formula for $H$ due to its reliance on differentiation with respect to log-parameters excluding $\sigma$!)

The factor of $\sigma^{-2}$ in the third term will appear in all $H_{ij}$, so it rescales the entire matrix $H$.

Since our only use of $H$ is to calculate its eigenvalues and eigenvectors, and these will be unaffected by any rescaling of the whole matrix, we can safely choose any arbitrary (but not ridiculous) value of $\sigma$! **Let's do our first analysis of model sloppiness in Python!**

# The Levenberg-Marquardt Hessian matrix: Faster, more approximate

Levenberg-Marquardt Hessian matrix $\mathbf{L}$ evaluated at $\boldsymbol{\theta}_{\mathrm{MLE}}$, with elements $L_{ij}$ given by:

$$L_{ij} = \sum_{k=1}^{N_{\mathrm{obs}}} \frac{\partial r_k}{\partial \log \theta_i} \frac{\partial r_k}{\partial \log \theta_j}\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{\mathrm{MLE}}}, \quad \text{where} \quad r_k(\boldsymbol{\theta}) = \frac{1}{\sigma}\left(y_{\mathrm{obs},k} - y_{\mathrm{model},k}(\boldsymbol{x}, \boldsymbol{\theta})\right).$$

The matrix $\mathbf{L}$ provides a fast approximation of $\mathbf{H}$.

We can again estimate the derivatives via finite differencing, using a user-chosen scalar $\delta$:

$$\frac{\partial r_k}{\partial \log \theta_i}\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{\mathrm{MLE}}} \approx \frac{1}{\delta}\left[r_k(\boldsymbol{\theta}_{i\uparrow}) - r_k(\boldsymbol{\theta}_{i\downarrow})\right], \text{ where}$$

$\boldsymbol{\theta}_{i\uparrow}$ is $\boldsymbol{\theta} = \boldsymbol{\theta}_{\mathrm{MLE}}$, with: $\theta(i) \leftarrow \theta(i)\left(1 + \frac{\delta}{2}\right)$. $\boldsymbol{\theta}_{i\downarrow}$ is $\boldsymbol{\theta} = \boldsymbol{\theta}_{\mathrm{MLE}}$, with: $\theta(i) \leftarrow \theta(i)\left(1 - \frac{\delta}{2}\right)$.

# Battle of the frequentist matrices

If finite differencing is used to approximate **H** and **L**:

- The Hessian matrix **H** requires $\mathcal{O}(N_\theta^2)$ evaluations of the model to compute,
- The Levenberg-Marquardt Hessian matrix **L** requires $\mathcal{O}(N_\theta)$ evaluations of the model to compute,

so **L** is a lot faster to evaluate than **H**!

This may not matter if your model has a small number of parameters *and* it is computationally cheap to evaluate.

However, this is an important point if your model has a large number of parameters *or* the model is computationally expensive to evaluate.

Let's calculate the Levenberg-Marquardt Hessian matrix **L** in Python!

QUT
BRISBANE
AUSTRALIA

# 3. Analysis of model sloppiness (Part 2)

# Sensitivity matrices for Bayesian model-data calibrations

Two sensitivity matrices proposed for analysing model sloppiness in Bayesian model-data calibrations:

- Principal Component Analysis (PCA) Hessian matrix **P**.
- Likelihood-informed subspace (LIS)-based sensitivity matrix **G**.

PCA Hessian matrix **P** is a **no-brainer** if you have already gone to the effort of doing a Bayesian model-data calibration. It is extremely fast to compute, and easy to interpret.

LIS-based matrix **G** is slower to compute, and has a specific purpose: attempting to identify what information has been gained from the data *relative* to the prior.

The LIS-based matrix **G** should not be used by itself – it is a complement to, and should only be used in comparison to, the PCA Hessian matrix **P**.

# The PCA Hessian matrix: "A real bargain"

The PCA Hessian matrix $\mathbf{P}$ is simply the inverse of the posterior covariance matrix $\mathbf{\Sigma}$:

$$\boxed{\mathbf{P} = \mathbf{\Sigma}^{-1}}$$

where $\mathbf{\Sigma}^{-1}$ is here calculated in terms of the natural logarithms of model parameters $\log \boldsymbol{\theta}$.

If you've already got a set of $M$ equally weighted samples $\{\boldsymbol{\theta}_m\}_{m=1}^{M}$ from the posterior (e.g. by doing MCMC or SMC sampling), $\mathbf{\Sigma}$ can be approximated by the sample covariance matrix $\hat{\mathbf{\Sigma}}$:

$$\mathbf{\Sigma} \approx \hat{\mathbf{\Sigma}} = \frac{1}{M-1} \sum_{m=1}^{M} \left(\log \boldsymbol{\theta}_m - \overline{\log \boldsymbol{\theta}}\right) \left(\log \boldsymbol{\theta}_m - \overline{\log \boldsymbol{\theta}}\right)^{\top},$$

where $\overline{\log \boldsymbol{\theta}}$ is the estimated posterior mean for the natural logarithm of parameters,

$$\overline{\log \boldsymbol{\theta}} = \frac{1}{M} \sum_{m=1}^{M} \log \boldsymbol{\theta}_m.$$

# The PCA Hessian matrix: “A real bargain” (continued)

Advantageously, this means that if you already have a large number of posterior samples from MCMC or SMC, the PCA Hessian matrix $\mathbf{P}$ requires **no** evaluations of the model at all to compute it!

(Aside: If you are instead using variational Bayesian inference or Laplace approximation to estimate the posterior, this may already yield an estimate of the covariance matrix $\mathbf{\Sigma}$ that you need for calculating $\mathbf{P} = \mathbf{\Sigma}^{-1}$!)

# The LIS-based sensitivity matrix: "The odd one out"

The goal of the likelihood-informed subspace (LIS)-based sensitivity matrix $\mathbf{G}$ is to *eliminate* the effects of the prior distribution when performing analysis of model sloppiness.

This makes $\mathbf{G}$ unique amongst the four sensitivity matrices we cover in this course.

However, it is also the most complicated and computationally-intensive sensitivity matrix to compute, of the four we discuss here. We first introduce its formal definition, and in later slides discuss how to *actually* calculate it from a large number of posterior samples from MCMC or SMC.

The LIS-based sensitivity matrix $\mathbf{G}$ is an integral over all parameter space $\boldsymbol{\theta}$ of the prior-preconditioned Hessian matrix $\boldsymbol{\Psi}(\boldsymbol{\theta})$ with respect to the posterior $p\left(\boldsymbol{\theta}|\boldsymbol{y}_{\mathrm{obs}}\right)$,

$$\mathbf{G} = \int_{\boldsymbol{\theta}} \boldsymbol{\Psi}(\boldsymbol{\theta})\, p\left(\boldsymbol{\theta}|\boldsymbol{y}_{\mathrm{obs}}\right)\, \mathrm{d}\boldsymbol{\theta}, \qquad \text{where} \qquad \boldsymbol{\Psi}(\boldsymbol{\theta}) = \mathbf{L}_{\mathbf{p}}^{\top}\mathbf{H}(\boldsymbol{\theta})\mathbf{L}_{\mathbf{p}}.$$

# The LIS-based sensitivity matrix: "The odd one out" (continued)

$$\mathbf{G} = \int_{\boldsymbol{\theta}} \boldsymbol{\Psi}(\boldsymbol{\theta})\, p\left(\boldsymbol{\theta}|\boldsymbol{y}_{\mathrm{obs}}\right)\, \mathrm{d}\boldsymbol{\theta}, \qquad \text{where} \qquad \boldsymbol{\Psi}(\boldsymbol{\theta}) = \mathbf{L}_{\mathbf{p}}^{\top}\mathbf{H}(\boldsymbol{\theta})\mathbf{L}_{\mathbf{p}}.$$

In the above equation, $\mathbf{L}_{\mathbf{p}}$ is the Cholesky decomposition of the covariance matrix of the prior distribution $p(\boldsymbol{\theta})$, which is thereafter used for the prior-preconditioning of the Hessian matrix $\mathbf{H}(\boldsymbol{\theta})$ to obtain $\boldsymbol{\Psi}(\boldsymbol{\theta})$.

The intention of the prior-preconditioned matrix $\boldsymbol{\Psi}(\boldsymbol{\theta})$ is to remove (as much as is practical) the effects of the prior from its values.

Note that we are now needing to *integrate* a transformed form of the Hessian matrix *over all* parameter space! That seems a bit nasty, but we can again approximate $\infty$ by $M = 1000$, and thus assume that the average of $\boldsymbol{\Psi}(\boldsymbol{\theta}_m)$ across our $M = 1000$ posterior samples $\boldsymbol{\theta}_m$ obtained from SMC is equal to the desired integral. (This approach to approximating integrals is technically called *Monte Carlo integration*.)

# The LIS-based sensitivity matrix: "The odd one out" (continued)

$$\mathbf{G} = \int_{\boldsymbol{\theta}} \boldsymbol{\Psi}(\boldsymbol{\theta})\, p\left(\boldsymbol{\theta}|\boldsymbol{y}_{\text{obs}}\right)\, \mathrm{d}\boldsymbol{\theta}, \qquad \text{where} \qquad \boldsymbol{\Psi}(\boldsymbol{\theta}) = \mathbf{L}_{\mathbf{p}}^{\top}\mathbf{H}(\boldsymbol{\theta})\mathbf{L}_{\mathbf{p}}.$$

The steps to compute $\mathbf{G}$ from a large number of posterior samples is as follows:

1. Calculate the covariance matrix $\boldsymbol{\Omega}$ of the prior distribution (remember: with respect to log-parameters!).
2. Use Cholesky decomposition on $\boldsymbol{\Omega}$ to obtain the lower triangular matrix $\mathbf{L}_{\mathbf{p}}$ that satisfies $\mathbf{L}_{\mathbf{p}}\mathbf{L}_{\mathbf{p}}^{\top} = \boldsymbol{\Omega}$.
3. For each of the $M$ posterior samples $\{\boldsymbol{\theta}_m\}_{m=1}^{M}$, calculate its Hessian matrix $\mathbf{H}(\boldsymbol{\theta}_m)$. (This is by far the most *computationally intensive step* of the whole algorithm, so we might use e.g. the Levenberg-Marquardt matrix $\mathbf{L}$ to approximate $\mathbf{H}$ instead).
4. For each of the $M$ Hessian matrices $\mathbf{H}(\boldsymbol{\theta}_m)$, calculate the associated prior-preconditioned matrix $\boldsymbol{\Psi}(\boldsymbol{\theta}_m) = \mathbf{L}_{\mathbf{p}}^{\top}\mathbf{H}(\boldsymbol{\theta}_m)\mathbf{L}_{\mathbf{p}}$.
5. Estimate $\mathbf{G}$ by averaging all prior-preconditioned matrices, i.e. $\mathbf{G} \approx \dfrac{1}{M}\displaystyle\sum_{m=1}^{M} \boldsymbol{\Psi}(\boldsymbol{\theta}_m)$.

QUT
BRISBANE
AUSTRALIA

# 4. Interpreting model sloppiness outputs, and connections to other data-informing approaches

# Interpretations of sloppiness, and connections to other approaches

Spoilers: there are many! Here's a few.

1. Practical vs structural identifiability
2. Model reduction
3. Design of future measurements
4. Profile likelihood analysis
5. Analysing the model-data calibration process

# [1] Practical vs structural identifiability

So far we have shown that analysis of model sloppiness can unveil the identifiability of model parameters (and eigenparameters) given some data.

What if, in the limit of $\infty$ data, some parameters (and/or eigenparameters) are still not identifiable?

**Structurally identifiable** – Parameter values can be estimated well if an $\infty$ amount of noise-free data is available.

**Practically identifiable** – Parameter values can be estimated well if a practically measurable amount of data is obtained.

Algebraic approaches can be used to assess structural identifiability. Practical identifiability is more nebulous... analysis of model sloppiness is a *blunt hammer* for investigating this!

Further reading: Browning *et al.* (2020) Identifiability analysis for stochastic differential equation models in systems biology. *J. R. Soc. Interface* **17**: 20200652.

# [2] Model reduction

So a model has many sloppy eigenparameters. Can we use this knowledge to reduce the complexity of the model while still fitting the data well?

Some options:

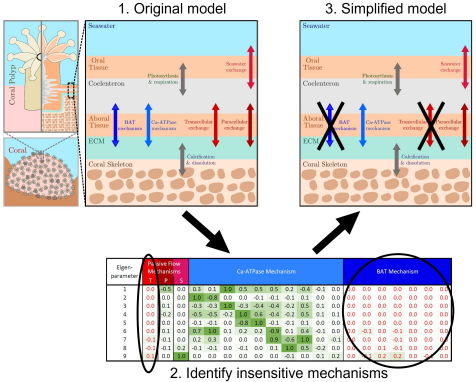
- "Strategic" model reduction by analysing model sloppiness.
- Manifold boundary approximation method (MBAM).

These methods provide different ways of reducing the number of parameters in the model.

Both methods therefore simplify the model, but in different ways.

# [2] Model reduction

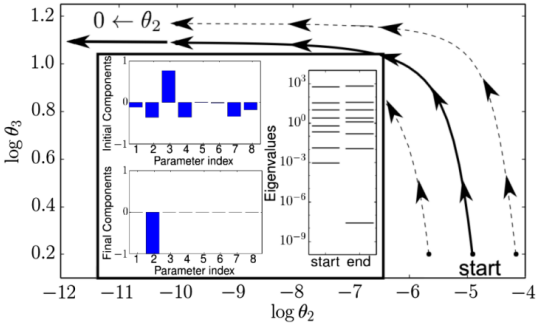
**“Strategic” model reduction by analysing model sloppiness:**

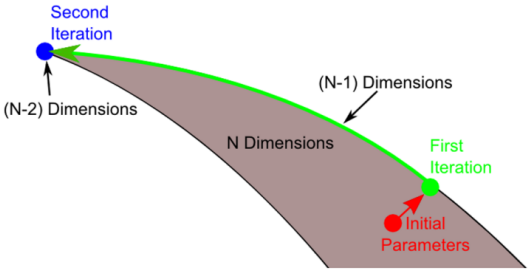



Further reading: Vollert *et al.* (2023) Strategic model reduction by analysing model sloppiness. *Env. Mod. Softw.* **159**: 105578.

# [2] Model reduction

**"Manifold boundary approximation method (MBAM):**




Further reading: Transtrum & Qiu (2014) Model reduction by manifold boundaries. *Phys. Rev. Lett.* **113**: 098701.

# [2] Model reduction: Comparison

**"Strategic" model reduction by analysing model sloppiness**

1. Perform analysis of model sloppiness.
2. Identify mechanisms whose parameters all do not appear in the stiffest eigenparameters.
3. Remove these mechanisms.
4. Recalibrate the reduced model to the data.
5. Repeat until happy.

**Manifold boundary approximation method (MBAM)**

1. Perform something *akin* to analysis of sloppiness.
2. Find the sloppiest eigenparameter, and move in parameter space in the direction of this eigenparameter until the sensitivity matrix becomes singular.
3. Algebraically evaluate the simplified model at the end of this path to remove one parameter.
4. Recalibrate the reduced model to the data.
5. Repeat until happy.

# [3] Design of future measurements

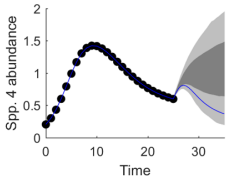
Analysis of model sloppiness can assist with design of future experiments.

For example, if $\hat{\theta}_1 = \theta_1 \theta_2^{-0.4}$ is the stiffest eigenparameter, then any measurement of $\theta_1$ will automatically reduce our uncertainty for $\theta_2$ because (presumably) the value of $\hat{\theta}_1$ is already narrowly constrained! And vice versa for a measurement of $\theta_2$!

So a measurement of $\theta_1$ (or $\theta_2$) may be **doubly-useful** for improving parameter values.

But this pre-supposes that all we care about is parameter uncertainty!

Reduced parameter uncertainty $\not\propto$ Reduced forecast uncertainty



# [3] Design of future measurements: Optimal design

However, if we want to more generally choose the best experiment or field measurement that maximises some pre-defined utility $U$, e.g.

- $\uparrow U$ indicates reduced parameter uncertainty, or
- $\uparrow U$ indicates that a model forecast will better distinguish between potential system outcomes,

we can more generally use "optimal design of experiments":

$$\boldsymbol{d}^* = \underset{\boldsymbol{d} \in \boldsymbol{D}}{\operatorname{argmax}} \, \mathbb{E}\left\{U(\boldsymbol{d}, \boldsymbol{\theta}, \boldsymbol{y}_{\text{obs}})\right\}$$

The mathematical procedures for this are well-defined for Bayesian model-data calibrations, although computational cost can be a problem.

Further reading: Ryan *et al.* (2016) A review of modern computational algorithms for Bayesian optimal design. *Int. Stat. Rev.* **84**: 128-154.

# [4] Profile likelihood analysis

Computational cost is a plague on Bayesian implementations in general! In the examples we did in Python, computational costs were far greater in the **SMC algorithm** than in any other calculations that we did!

An alternative approach to parameter identifiability, in frequentist calibrations, is to:

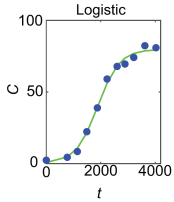
1. Find the MLE; recall this is
$$\boldsymbol{\theta}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\text{argmax}} \left[\mathcal{L}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta})\right]$$
2. Split the parameter set $\boldsymbol{\theta}$ into two groups: "interest parameters" $\boldsymbol{\psi}$ and "nuisance parameters" $\boldsymbol{\lambda}$, so $\boldsymbol{\theta} = (\boldsymbol{\psi}, \boldsymbol{\lambda})$.
3. Hold the interest parameters $\boldsymbol{\psi}$ constant (e.g. $\boldsymbol{\psi} = \boldsymbol{\psi}^*$), and recalculate the MLE:
$$\boldsymbol{\theta}_{\text{MLE}}|_{\boldsymbol{\psi}=\boldsymbol{\psi}^*} = \underset{\boldsymbol{\lambda}}{\text{argmax}} \left[\mathcal{L}\left(\boldsymbol{y}_{\text{obs}} \mid \boldsymbol{\theta}|_{\boldsymbol{\psi}=\boldsymbol{\psi}^*}\right)\right]$$
4. Repeat Step 3 for a range of $\boldsymbol{\psi}^*$ values, then see how $\mathcal{L}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta})$ varies with $\boldsymbol{\theta}_{\text{MLE}}|_{\boldsymbol{\psi}=\boldsymbol{\psi}^*}$.

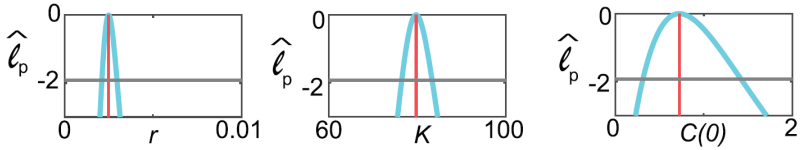
# [4] Profile likelihood analysis

This procedure of looking along the "ridges" of the loglikelihood function, is part of a broader "profile likelihood-based approach".

This approach is computationally efficient, and is appropriate if it is easy to calculate the MLE numerous times.

For example, fitting the logistic growth model $\frac{\mathrm{d}C}{\mathrm{d}t} = rC\,(1 - C/K)$, $C(0) = C_0$ to coral reef recovery data and looking at how the rescaled log-likelihood (called here $\hat{\ell}_p$) varies over interest parameters $\psi^* = r$, $\psi^* = K$ or $\psi^* = C(0)$:

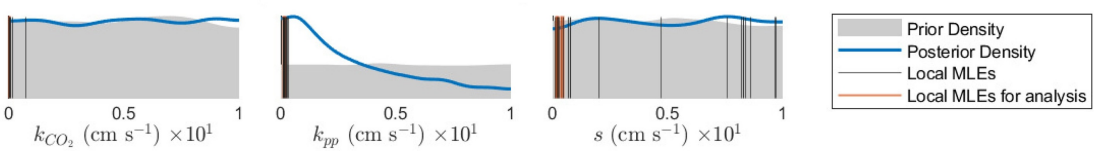



Further reading: Simpson *et al.* (2022) Parameter identifiability and model selection for sigmoid population growth models. *J. Theor. Biol.* **535**: 110998.

# [5] Analysing the model-data calibration process

When estimating parameters for very complex models using Bayesian inference, and large uncertainty remains after looking at the data ...

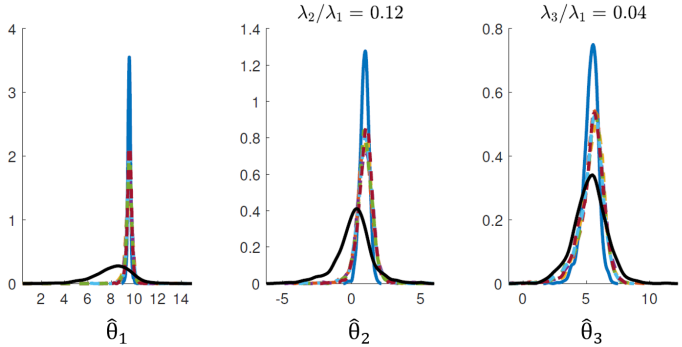



Marginal distribution for first 3 (of 20) parameters, from using Bayesian inference to fit a coral calcification model to data (Vollert *et al.* (2023)).

... how do we know if the code used in our model-data calibration procedure is actually working?

# [5] Analysing the model-data calibration process

We can also report the marginal distributions of the *stiffest eigenparameters* from analysis of model sloppiness:




Further reading: Botha *et al.* (2022) Component-wise iterative Kalman inversion for static Bayesian models with unknown measurement error covariance. *arXiv preprint* 2206.02451.

# Summing up

In this course we have discussed how to analyse model sloppiness, using both frequentist and Bayesian approaches.

In this last module, we have explored how model sloppiness links to related and extension topics, including parameter identifiability, model reduction, design of future experiments, and improvement of calibration algorithms.

I sincerely hope that some small part of this course was useful to some of you, in some way.

If you ever want to ask questions about any of the content in this course, I'm only an email away (mp.adams@qut.edu.au).

Thank you for listening!