GPs huh? what are they good for?

Richard Wilkinson

University of Nottingham

Gaussian Process Winter School, Sheffield 2014

Talk plan

(a) UQ and computer experiments

(b) Missing technical details

- Sampling GP hyper-parameters and model averaging
- Sequential GP updates

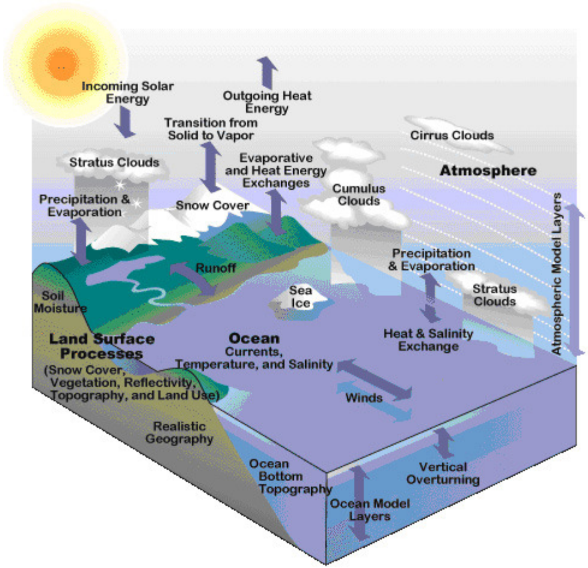
(c) Examples of use

  (i) accelerating Monte Carlo algorithms using GPs

  (ii) simulator discrepancy modelling

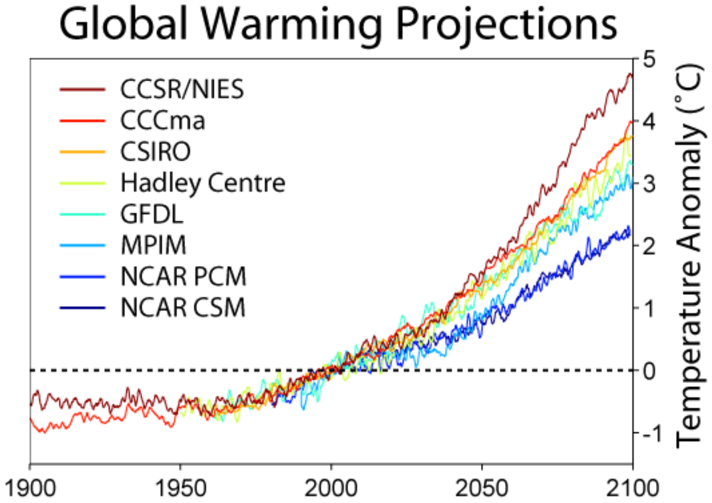
Climate Science

Predicting future climate



Challenges of computer experiments

Climate Predictions, IPCC AR4



Challenges for statistics

Key questions: How do we make inferences about the world from a simulation of it?

Challenges for statistics

Key questions: How do we make inferences about the world from a simulation of it?

- how do we relate simulators to reality? (model error)
- how do we estimate tunable parameters? (calibration)
- how do we deal with computational constraints? (stat. comp.)
- how do we make uncertainty statements about the world that combine models, data and their corresponding errors? (UQ)

There is an inherent a lack of quantitative information on the uncertainty surrounding a simulation - unlike in physical experiments.

Challenges for statistics

Key questions: How do we make inferences about the world from a simulation of it?

- how do we relate simulators to reality? (model error)
- how do we estimate tunable parameters? (calibration)
- how do we deal with computational constraints? (stat. comp.)
- how do we make uncertainty statements about the world that combine models, data and their corresponding errors? (UQ)

There is an inherent a lack of quantitative information on the uncertainty surrounding a simulation - unlike in physical experiments.

All of these questions are tackled by methodology that uses GPs!

Sampling hyperparameters

We've been modelling

$$f(x) \sim GP(0, k_\psi(x, x'))$$

where k_ψ is a covariance function that depends upon unknown parameters.

Sampling hyperparameters

We've been modelling

$$f(x) \sim GP(0, k_\psi(x, x'))$$

where k_ψ is a covariance function that depends upon unknown parameters.
For example, in 1d, using the exponentiated quadratic

$$k_\psi(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\lambda^2}\right)$$

and we might assume a Gaussian variance for the observations

$$y_i | f \sim N(f(x_i), \tau^2)$$

where λ is the length-scale, σ^2 is the GP variance, and τ^2 is the observation variance (nugget variance in kriging)

$$\psi = (\lambda, \sigma^2, \tau^2)$$

Estimating hyper-parameters

How do we estimate the hyper-parameters ψ?

- Most commonly, we use a *plug-in* approach, and fix their value at the MAP or ML estimate

  $$\widehat{\psi} = \arg\max_{\psi} \pi(\mathbf{y}|\mathbf{X}, \psi)\pi(\psi)$$

  and then make predictions using

  $$\pi(y^*|\widehat{\psi}, \mathbf{X}, \mathbf{y}, x^*)$$

Estimating hyper-parameters

How do we estimate the hyper-parameters ψ?

- Most commonly, we use a *plug-in* approach, and fix their value at the MAP or ML estimate

$$\widehat{\psi} = \arg\max_{\psi} \pi(\mathbf{y}|\mathbf{X}, \psi)\pi(\psi)$$

  and then make predictions using

$$\pi(y^*|\widehat{\psi}, \mathbf{X}, \mathbf{y}, x^*)$$

Sometimes, we might instead want to average across the unknown values when making predictions (i.e., be a good Bayesian)

$$\pi(y^*|\mathbf{X}, \mathbf{y}, x^*) = \int \pi(y^*|\psi, \mathbf{X}, \mathbf{y}, x^*)\pi(\psi|\mathbf{X}, \mathbf{y})\mathrm{d}\psi$$

Estimating hyper-parameters

How do we estimate the hyper-parameters ψ?

- Most commonly, we use a *plug-in* approach, and fix their value at the MAP or ML estimate

$$\widehat{\psi} = \arg\max_{\psi} \pi(\mathbf{y}|\mathbf{X}, \psi)\pi(\psi)$$

and then make predictions using

$$\pi(y^*|\widehat{\psi}, \mathbf{X}, \mathbf{y}, x^*)$$

Sometimes, we might instead want to average across the unknown values when making predictions (i.e., be a good Bayesian)

$$\pi(y^*|\mathbf{X}, \mathbf{y}, x^*) = \int \pi(y^*|\psi, \mathbf{X}, \mathbf{y}, x^*)\pi(\psi|\mathbf{X}, \mathbf{y})\mathrm{d}\psi$$

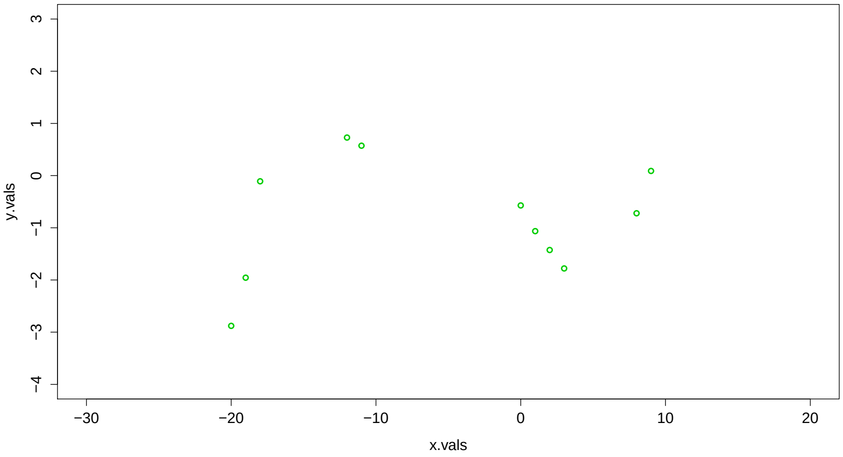
How do we do this?

- Laplace approximation
- Monte Carlo

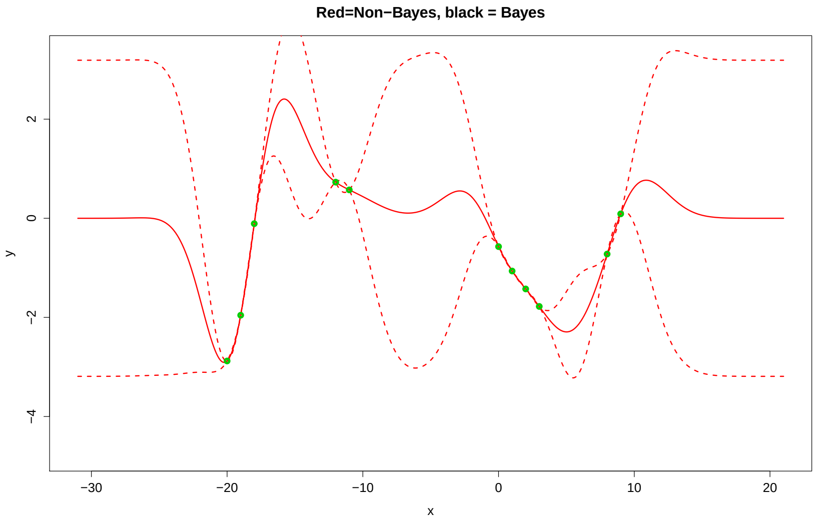
NB: careful choice of prior helps

Numerical demonstration

Toy data, generated from a GP with $\lambda = 3$, $\sigma^2 = 5$, $\tau^2 = 0.1$



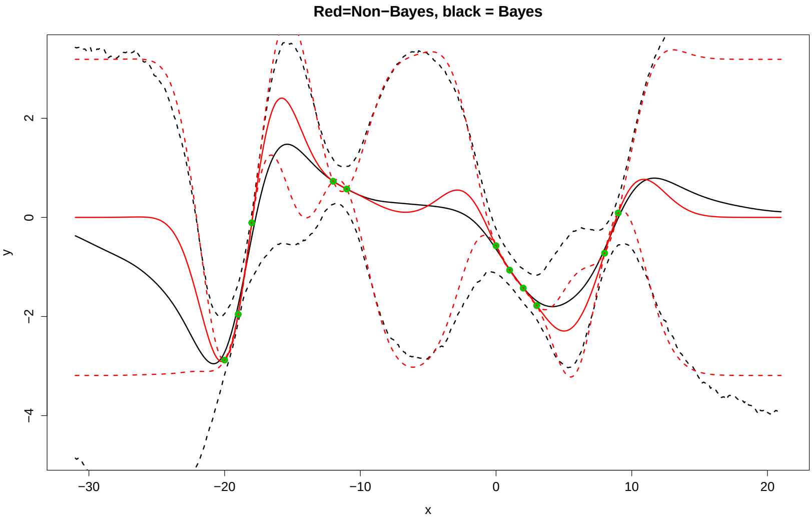
Plug-in (non-Bayesian) predictions $\pi(y^*|\widehat{\psi}, \mathbf{X}, \mathbf{y}, x^*)$



Note that there are regions where there is essentially no uncertainty.

Bayesian (model averaging) vs Non-Bayesian (ML plug-in)

$$\pi(y^*|\mathbf{X},\mathbf{y},x^*) = \int \pi(y^*|\psi,\mathbf{X},\mathbf{y},x^*)\pi(\psi|\mathbf{X},\mathbf{y})\mathrm{d}\psi$$



This is using the true covariance function. For a mis-specified model, the difference can be much greater.

Sampling hyper-parameters

Default approach is to resort to Metropolis-Hastings: build a Markov chain on the parameter space, $\psi_1, \psi_2, \ldots$ so that it converges to $\pi(\psi|\mathbf{X}, \mathbf{y})$

Suppose we're currently at ψ_t

1. Propose a move to ψ' from $q(\psi_t, \cdot)$, e.g. Gaussian random walk
2. Set $\psi_{t+1} = \psi'$ with MH acceptance probability

$$r = \frac{q(\psi', \psi_t)\pi(\mathbf{y}|\mathbf{X}, \psi')\pi(\psi')}{q(\psi_t, \psi')\pi(\mathbf{y}|\mathbf{X}, \psi_t)\pi(\psi_t)}$$

   otherwise set $\psi_{t+1} = \psi_t$

Sampling hyper-parameters

Default approach is to resort to Metropolis-Hastings: build a Markov chain on the parameter space, $\psi_1, \psi_2, \dots$ so that it converges to $\pi(\psi|\mathbf{X}, \mathbf{y})$

Suppose we're currently at ψ_t

1. Propose a move to ψ' from $q(\psi_t, \cdot)$, e.g. Gaussian random walk
2. Set $\psi_{t+1} = \psi'$ with MH acceptance probability

$$r = \frac{q(\psi', \psi_t)\pi(\mathbf{y}|\mathbf{X}, \psi')\pi(\psi')}{q(\psi_t, \psi')\pi(\mathbf{y}|\mathbf{X}, \psi_t)\pi(\psi_t)}$$

   otherwise set $\psi_{t+1} = \psi_t$

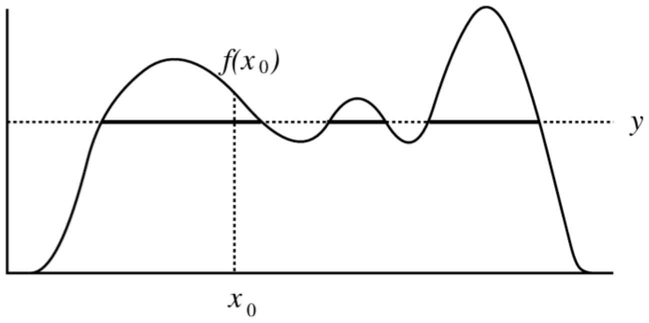
Unfortunately this approach often works poorly, even with extensive tuning of proposals

- Mixing tends to be very poor even in 1d problems

Instead, try slice sampling or Hamiltonian Monte Carlo (HMC).

Slice sampling hyper-parameters

Think $x = \psi$, find $\pi(x|D) \equiv \pi(\psi|X, y)$



To sample from $\pi(x|D)$, introduce auxilliary variable y, and sample from

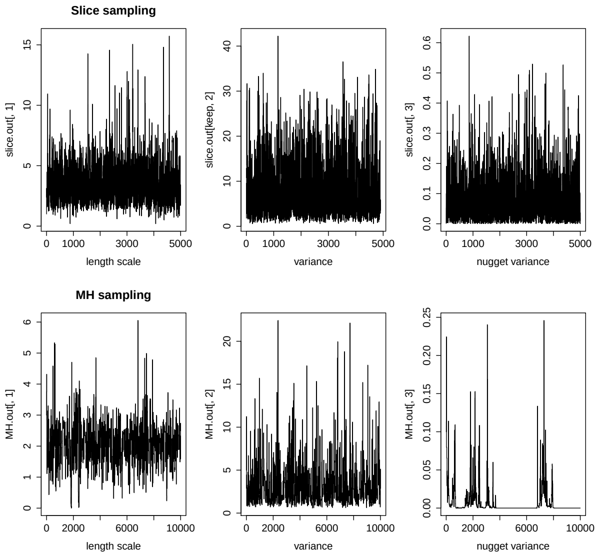
$$\pi(x, y) \sim \mathbb{I}_{y \le f(x)}$$

where $f(x) = \pi(x)l(x) \propto \pi(x|D)$

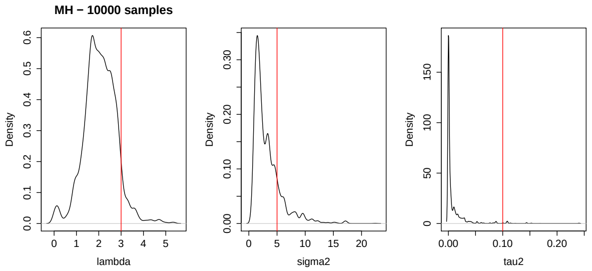
1. $y_{t+1} \sim U[0, f(x_t)]$
2. $x_{t+1} \sim U\{x : f(x) \ge y_{t+1}\}$

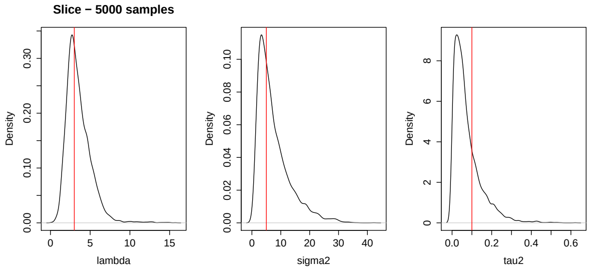
MCMC

Minimal tuning of the MH, no tuning in slice sampling



Numerical demonstration





Example 1: accelerating ABC

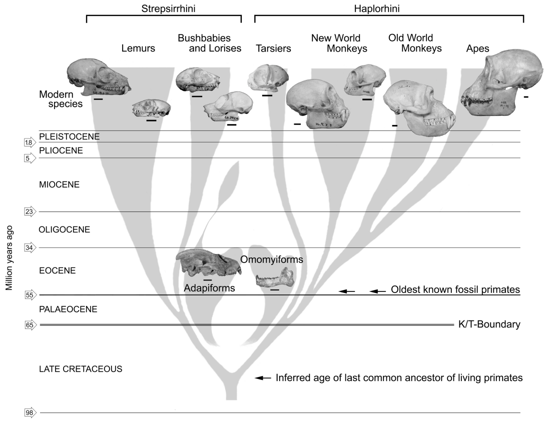
Calibration

- For most simulators we specify parameters θ and i.c.s and the simulator, $f(\theta)$, generates output X.
- We are interested in the inverse-problem, i.e., observe data D, want to estimate parameter values θ which explain this data.

For Bayesians, this is a question of finding the posterior distribution

$$\pi(\theta|\mathcal{D}) \propto \pi(\theta)\pi(\mathcal{D}|\theta)$$

$$\text{posterior} \propto \text{prior} \times \text{likelihood}$$



Approximate Bayesian Computation (ABC)

ABC algorithms are a collection of Monte Carlo methods used for calibrating simulators

- they do not require explicit knowledge of the likelihood function $\pi(x|\theta)$
- inference is done using simulation from the model (they are 'likelihood-free').

ABC methods have become popular in the biological sciences and versions of the algorithm exist in most modelling communities.

Approximate Bayesian Computation (ABC)

ABC algorithms are a collection of Monte Carlo methods used for calibrating simulators

- they do not require explicit knowledge of the likelihood function $\pi(x|\theta)$
- inference is done using simulation from the model (they are 'likelihood-free').

ABC methods have become popular in the biological sciences and versions of the algorithm exist in most modelling communities.
ABC methods can be crude but they have an important role to play.

- Scientists are building simulators (intractable ones), and fitting them to data.
  - There is a need for simple methods that can be credibly applied.
  - Likelihood methods for complex simulators are complex.
  - Modelling is something that can be done well by scientists not trained in complex statistical methods.

Uniform ABC algorithms

Uniform ABC

- Draw θ from $\pi(\theta)$
- Simulate $X \sim f(\theta)$
- Accept θ if $\rho(\mathcal{D}, X) \leq \epsilon$

Uniform ABC algorithms

Uniform ABC

- Draw θ from $\pi(\theta)$
- Simulate $X \sim f(\theta)$
- Accept θ if $\rho(\mathcal{D}, X) \leq \epsilon$

- As $\epsilon \to \infty$, we get observations from the prior, $\pi(\theta)$.
- If $\epsilon = 0$, we generate observations from $\pi(\theta \mid \mathcal{D})$

ϵ reflects the tension between computability and accuracy.

The hope is that $\pi_{ABC}(\theta) \approx \pi(\theta|D, PSH)$ for ϵ small, where PSH='perfect simulator hypothesis'

Uniform ABC algorithms

Uniform ABC

- Draw θ from $\pi(\theta)$
- Simulate $X \sim f(\theta)$
- Accept θ if $\rho(\mathcal{D}, X) \leq \epsilon$

- As $\epsilon \to \infty$, we get observations from the prior, $\pi(\theta)$.
- If $\epsilon = 0$, we generate observations from $\pi(\theta \mid \mathcal{D})$

ϵ reflects the tension between computability and accuracy.

The hope is that $\pi_{ABC}(\theta) \approx \pi(\theta | D, PSH)$ for ϵ small, where PSH='perfect simulator hypothesis'
There are uniform ABC-MCMC, ABC-SMC, ABC-EM, ABC-EP, ABC-MLE algorithms, etc.

Generalized ABC (GABC)

Wilkinson 2008/2013, Fearnhead and Prangle 2012

We can generalise the rejection-ABC algorithm by using arbitrary acceptance kernels:

Generalized rejection ABC (Rej-GABC)

1. $\theta \sim \pi(\theta)$ and $X \sim \pi(x|\theta)$
2. Accept (θ, X) if

$$U \sim U[0,1] \leq \frac{\pi_{ABC}(\theta, x)}{Mg(\theta, x)} = \frac{\pi(D|X)}{\max_x \pi(D|x)}$$

Generalized ABC (GABC)

Wilkinson 2008/2013, Fearnhead and Prangle 2012

We can generalise the rejection-ABC algorithm by using arbitrary acceptance kernels:

Generalized rejection ABC (Rej-GABC)

1 $\theta \sim \pi(\theta)$ and $X \sim \pi(x|\theta)$

2 Accept (θ, X) if

$$U \sim U[0,1] \leq \frac{\pi_{ABC}(\theta, x)}{Mg(\theta, x)} = \frac{\pi(D|X)}{\max_x \pi(D|x)}$$

In uniform ABC we take

$$\pi(D|X) = \begin{cases} 1 & \text{if } \rho(D, X) \leq \epsilon \\ 0 & \text{otherwise} \end{cases}$$

this reduces the algorithm to

2' Accept θ ifF $\rho(D, X) \leq \epsilon$

ie, we recover the *uniform* ABC algorithm.

Problems with Monte Carlo methods

Monte Carlo methods are generally guaranteed to succeed if we run them for long enough.

This guarantee comes at a cost.

- Most methods sample naively - they don't learn from previous simulations.
- They don't exploit known properties of the likelihood function, such as continuity
- They sample randomly, rather than using space filling designs.

This naivety can make a full analysis infeasible without access to a large amount of computational resource.

Problems with Monte Carlo methods

Monte Carlo methods are generally guaranteed to succeed if we run them for long enough.

This guarantee comes at a cost.

- Most methods sample naively - they don't learn from previous simulations.
- They don't exploit known properties of the likelihood function, such as continuity
- They sample randomly, rather than using space filling designs.

This naivety can make a full analysis infeasible without access to a large amount of computational resource.

If we are prepared to lose the guarantee of eventual success, we can exploit the continuity of the likelihood function to learn about its shape, and to dramatically improve the efficiency of our computations.

Likelihood estimation

The GABC framework assumes

$$\begin{aligned}\pi(D|\theta) &= \int \pi(D|X)\pi(X|\theta)\mathrm{dX} \\ &\approx \frac{1}{N}\sum \pi(D|X_i)\end{aligned}$$

where $X_i \sim \pi(X|\theta)$. Or in Wood (2010),

$$\pi(D|\theta) = \phi(D; \mu_\theta, \Sigma_\theta)$$

Likelihood estimation

The GABC framework assumes

$$\begin{aligned}\pi(D|\theta) &= \int \pi(D|X)\pi(X|\theta)\mathrm{dX} \\ &\approx \frac{1}{N}\sum \pi(D|X_i)\end{aligned}$$

where $X_i \sim \pi(X|\theta)$. Or in Wood (2010),

$$\pi(D|\theta) = \phi(D; \mu_\theta, \Sigma_\theta)$$

For many problems, we believe the likelihood is continuous and smooth, so that $\pi(D|\theta)$ is similar to $\pi(D|\theta')$ when $\theta - \theta'$ is small

We can model $\pi(D|\theta)$ using GPs and use the GP model to find the posterior in place of running the simulator.

Example: Ricker Model

The Ricker model is one of the prototypic ecological models.

- used to model the fluctuation of the observed number of animals in some population over time
- It has complex dynamics and likelihood, despite its simple mathematical form.

Ricker Model

- Let N_t denote the number of animals at time t.

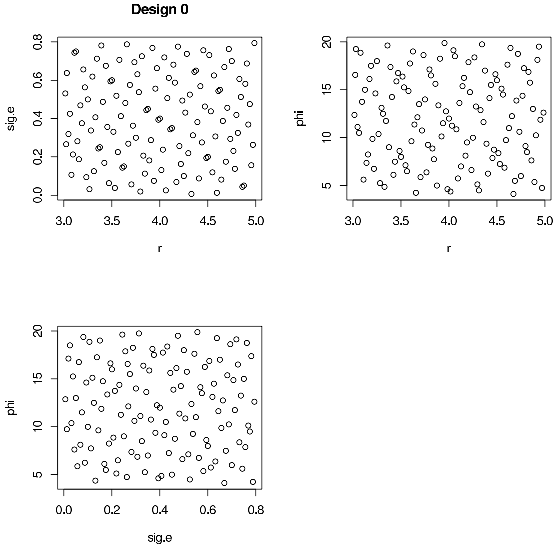
$$N_{t+1} = rN_t e^{-N_t + e_r}$$

  where e_t are independent $N(0, \sigma_e^2)$ process noise
- Assume we observe counts y_t where

$$y_t \sim Po(\phi N_t)$$

Design 1 - 128 pts

We use a Sobol sequence on the prior input space to find a design $\{\theta_i\}_{i=1}^{d}$. We estimate the likelihood at each point in the design, and aim to fit a GP model to estimate the likelihood at θ values not in the design.



History matching waves

The likelihood is too difficult to model, so we model the log-likelihood instead.

$$G(\theta) = \log L(\theta), \qquad \hat{L}(\theta_i) = \frac{1}{N}\sum \pi(D|X_i),\ X_i \sim \pi(X|\theta_i)$$

History matching waves

The likelihood is too difficult to model, so we model the log-likelihood instead.

$$G(\theta) = \log L(\theta), \qquad \hat{L}(\theta_i) = \frac{1}{N}\sum \pi(D|X_i),\ X_i \sim \pi(X|\theta_i)$$

However, the log-likelihood for a typical problem ranges across too wide a range of values.

Consequently, any Gaussian process model will struggle to model the log-likelihood across the entire input range.

History matching waves

The likelihood is too difficult to model, so we model the log-likelihood instead.

$$G(\theta) = \log L(\theta), \qquad \hat{L}(\theta_i) = \frac{1}{N} \sum \pi(D|X_i),\ X_i \sim \pi(X|\theta_i)$$

However, the log-likelihood for a typical problem ranges across too wide a range of values.

Consequently, any Gaussian process model will struggle to model the log-likelihood across the entire input range.

- Introduce waves of history matching, similar to those used in Michael Goldstein's work.
- In each wave, build a GP model that can rule out regions of space as *implausible*.

History matching waves

We say θ is implausible if

$$m(\theta) + 3\sigma < \max_{\theta_i} \log \pi(D|\theta_i) - T$$

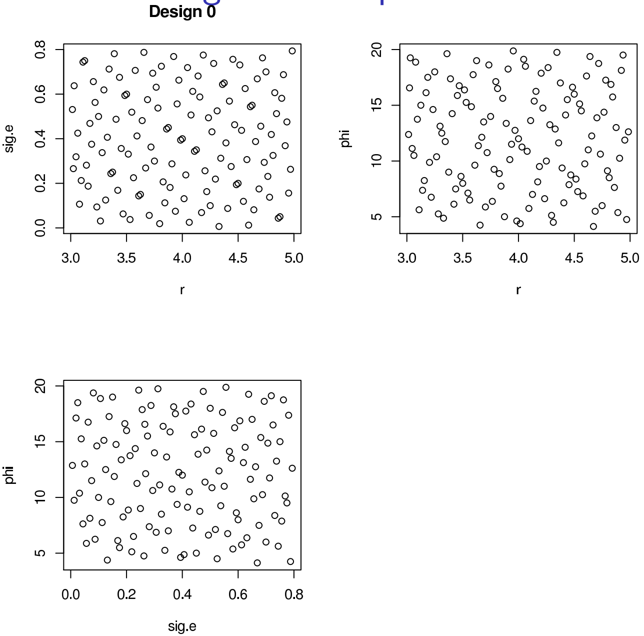
where $m(\theta)$ is the Gaussian process estimate of $\log \pi(D|\theta)$, and σ is the variance of the GP estimate.

- We use threshold $T = 10$ for the Ricker model, as a difference of 10 on the log scale between two likelihoods, means that assigning the θ with the smaller log-likelihood a posterior density of 0 (by saying it is implausible) is a good approximation.

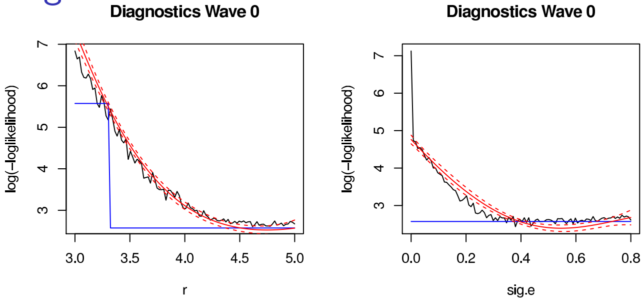
This still wasn't enough in some problems,

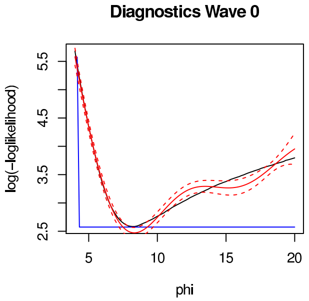
- Instead model $\log(-\log \pi(D|\theta))$ in the first wave
- For the next wave, we begin by using the GPs from the previous waves to decide which parts of the input space are implausible.
- We then extend the design into the not-implausible region and build a new GP
- Use this GP to rule out more space as implausible. Build a new (and more accurate GP) in this region. etc

Results - Design 1 - 128 pts

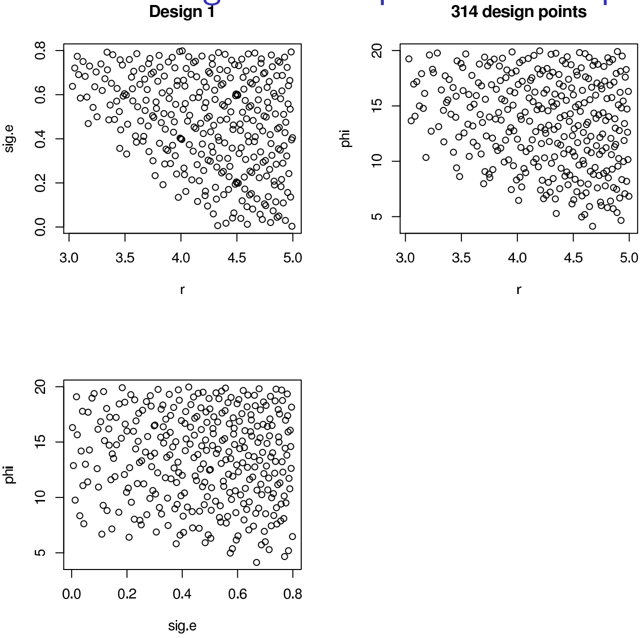


Diagnostics for GP 1 - threshold = 5.6

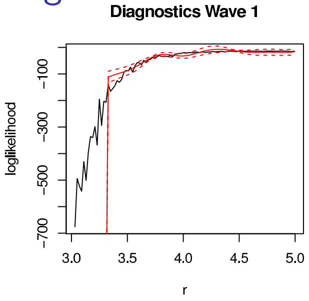


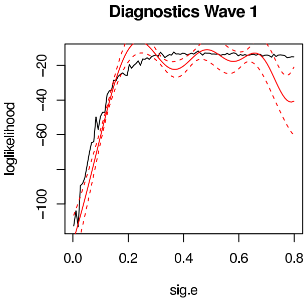


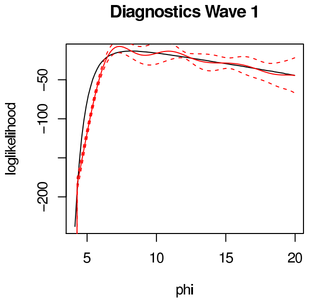
Results - Design 2 - 314 pts - 38% of space implausible



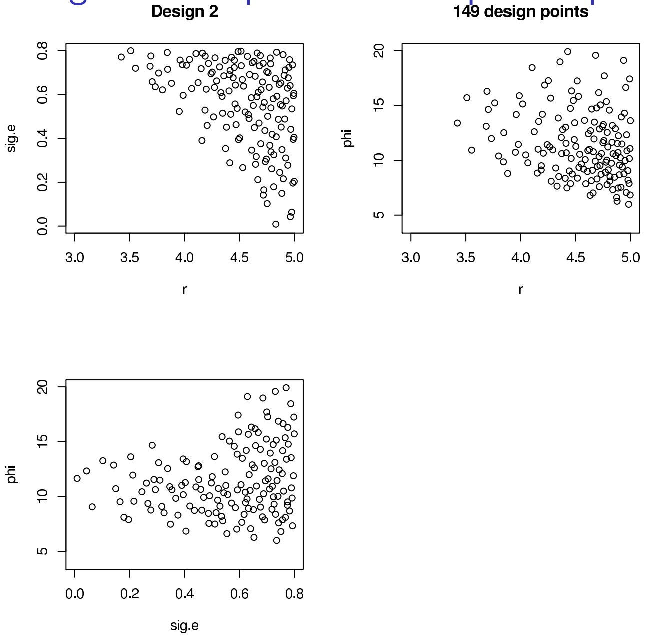
Diagnostics for GP 2 - threshold = -21.8



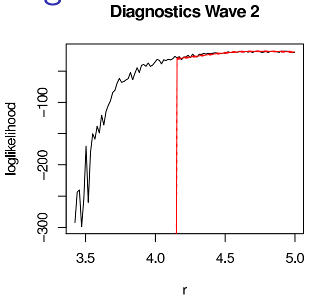


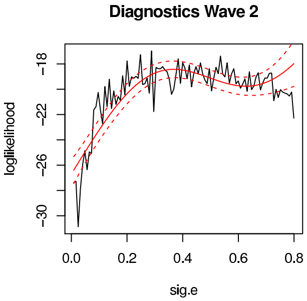


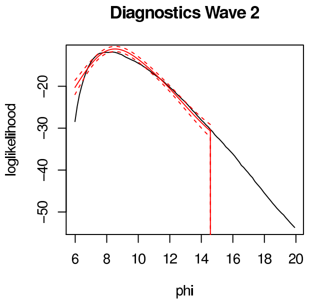
Design 3 - 149 pts - 62% of space implausible



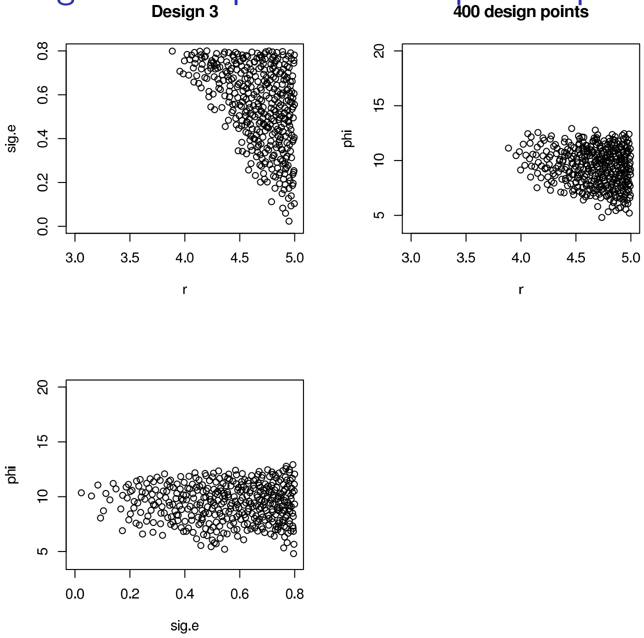
Diagnostics for GP 3 - threshold = -20.7



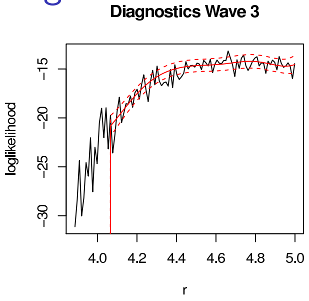


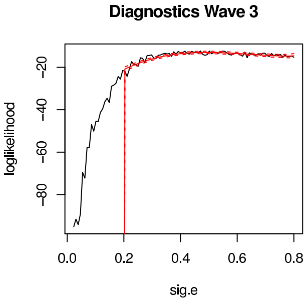


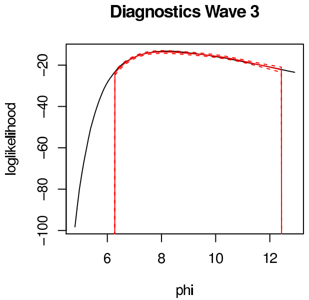
Design 4 - 400 pts - 95% of space implausible



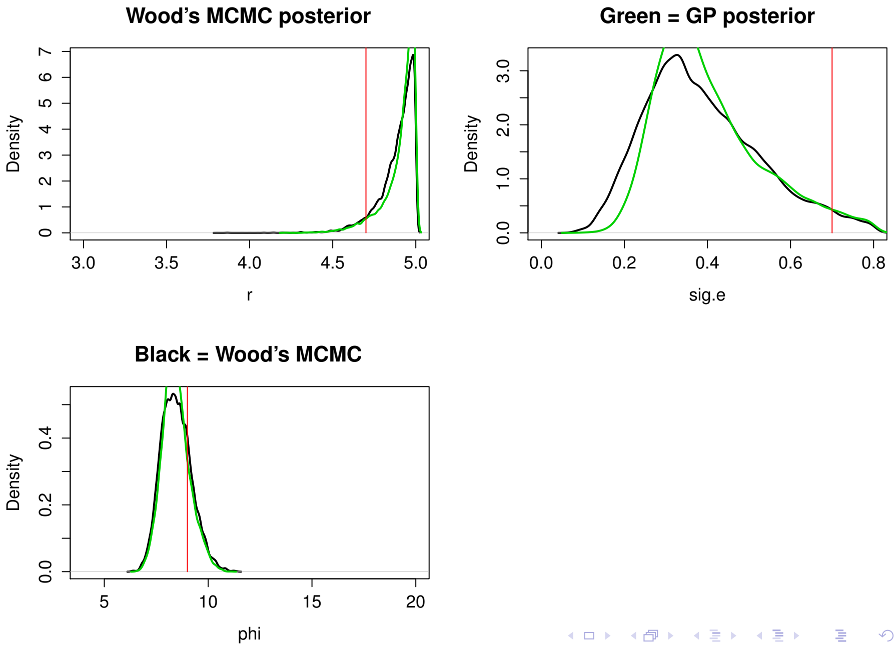
Diagnostics for GP 4 - threshold = -16.4







MCMC Results



Computational details

- The Wood MCMC method used $10^5 \times 500$ simulator runs
- The GP code used $(128 + 314 + 149 + 400) = 991 \times 500$ simulator runs
    - 1/100th of the number used by Wood's method.

By the final iteration, the Gaussian processes had ruled out over 98% of the original input space as implausible,

- the MCMC sampler did not need to waste time exploring those regions.

Other GP accelerated Monte Carlo approaches

There are several different ways of approaching this problem

- Wilkinson 2014 models the log-likelihood $l(\theta)$ as its one dimensional. But, the orders of magnitude variation make it difficult to model
- Meeds and Welling 2014 instead model the mean and variance of the simulator summary

  $$\mathbb{E}S(f(\theta)) \approx g_1(\theta) \qquad \mathbb{V}\mathrm{ar}S(f(\theta)) \approx g_2(\theta)$$

  and then assume $L(\theta) = \Phi(S_{obs}; g_1(\theta), g_2(\theta))$.

  The mean and variance will be easier to model than the likelihood (although usually $\dim S \neq 1$), but not all simulators have a likelihood that can be approximated by a Gaussian.

Other GP accelerated Monte Carlo approaches

There are several different ways of approaching this problem

- Wilkinson 2014 models the log-likelihood $l(\theta)$ as its one dimensional. But, the orders of magnitude variation make it difficult to model
- Meeds and Welling 2014 instead model the mean and variance of the simulator summary

$$\mathbb{E}S(f(\theta)) \approx g_1(\theta) \qquad \mathbb{V}\mathrm{ar}S(f(\theta)) \approx g_2(\theta)$$

$$\text{and then assume } L(\theta) = \Phi(S_{obs}; g_1(\theta), g_2(\theta)).$$

  The mean and variance will be easier to model than the likelihood (although usually $\dim S \neq 1$), but not all simulators have a likelihood that can be approximated by a Gaussian.

Other GP accelerated Monte Carlo methods

- Rasmussen 2003 used GPs to accelerate HMC
- Dahlin and Lindsten 2013 used GPs to accelerate parameter estimation in SSMs
  - Log-likellihood at θ can be estimated by running a particle filter - very expensive.

Active learning

To save computational cost, we want to build the design iteratively

- choose a new design point x_n according to some criterion using the current GP
- run the simulator to find y_n
- update the fit conditional on the new data point

How do we do this efficiently?

Active learning

To save computational cost, we want to build the design iteratively

- choose a new design point x_n according to some criterion using the current GP
- run the simulator to find y_n
- update the fit conditional on the new data point

How do we do this efficiently?

Suppose we have a GP model trained using $n-1$ training points. What does this mean?

- We have the inverse Gram matrix K_{n-1}^{-1} where $(K_{n-1})_{ij} = k(x_i, x_j)$.
- We have estimates of hyper-parameters $\hat{\psi}_{n-1}$.

Active learning

To save computational cost, we want to build the design iteratively

- choose a new design point x_n according to some criterion using the current GP
- run the simulator to find y_n
- update the fit conditional on the new data point

How do we do this efficiently?

Suppose we have a GP model trained using $n-1$ training points. What does this mean?

- We have the inverse Gram matrix K_{n-1}^{-1} where $(K_{n-1})_{ij} = k(x_i, x_j)$.
- We have estimates of hyper-parameters $\hat{\psi}_{n-1}$.

Suppose now we are given a new observation pair (x_n, y_n) that we want to incorporate in our GP training set.

- add a row and column to the previous Gram matrix K_{n-1} to form new matrix K_n, and find the inverse K_n^{-1}
- a naive implementation would have computational cost $O(n^3)$

Sequential GP updates

It is much more efficient to use sequential matrix updates instead.

Sequential GP updates

It is much more efficient to use sequential matrix updates instead. If

$$K_n = \begin{pmatrix} K_{n-1} & k(x_n) \\ k(x_n)^\top & k(x_n, x_n) \end{pmatrix}$$

where $k(x_n) = (k(x_1, x_n), \dots, k(x_{n-1}, x_n))^\top$

Sequential GP updates

It is much more efficient to use sequential matrix updates instead. If

$$K_n = \begin{pmatrix} K_{n-1} & k(x_n) \\ k(x_n)^\top & k(x_n, x_n) \end{pmatrix}$$

where $k(x_n) = (k(x_1, x_n), \ldots, k(x_{n-1}, x_n))^\top$ then

$$K_n^{-1} = \begin{pmatrix} K_{n-1}^{-1} + g_n(x_n) g_n^\top(x_n)/\mu_n(x_n) & g_n(x_n) \\ g_n^\top(x_n) & \mu_n(x_n) \end{pmatrix}$$

with

$$\begin{aligned} g_n(x) &= \mu(x) K_{n-1}^{-1} k(x) \\ \mu_n(x) &= \left(k(x,x) - k(x)^T K_{n-1}^{-1} k(x) \right)^{-1} \end{aligned}$$

Sequential GP updates

It is much more efficient to use sequential matrix updates instead. If

$$K_n = \begin{pmatrix} K_{n-1} & k(x_n) \\ k(x_n)^\top & k(x_n, x_n) \end{pmatrix}$$

where $k(x_n) = (k(x_1, x_n), \ldots, k(x_{n-1}, x_n))^\top$ then

$$K_n^{-1} = \begin{pmatrix} K_{n-1}^{-1} + g_n(x_n) g_n^\top(x_n)/\mu_n(x_n) & g_n(x_n) \\ g_n^\top(x_n) & \mu_n(x_n) \end{pmatrix}$$

with

$$\begin{aligned} g_n(x) &= \mu(x) K_{n-1}^{-1} k(x) \\ \mu_n(x) &= \left(k(x,x) - k(x)^T K_{n-1}^{-1} k(x) \right)^{-1} \end{aligned}$$

This has cost $O(n^2)$ rather than $O(n^3)$!

Demonstration

Suppose we have 1000 observations that arrive sequentially, and we want to train a GP model to predict one step ahead.

Some timings (s):

n	Naive	sequential
100	0.050	0.046
300	4.6	0.9
500	33	3.0
750	145	14
1000	422	29

Hyper-parameters

How do we sequentially deal with hyper parameters?

Hyper-parameters

How do we sequentially deal with hyper parameters?

Naive approach:

- update (with ML say) after adding every m additional data points.
- Note that this involves repeatedly doing the $O(n^3)$ inverse in the optimisation
- We can start the optimiser at the previous estimated hyper parameter value.

Hyper-parameters

How do we sequentially deal with hyper parameters?

Naive approach:

- update (with ML say) after adding every m additional data points.
- Note that this involves repeatedly doing the $O(n^3)$ inverse in the optimisation
- We can start the optimiser at the previous estimated hyper parameter value.

An alternative is to use sequential Monte Carlo (SMC) methods

SMC for GP hyper-parameters

Gramacy and Polson 2012

In SMC we represent the posteriors by a cloud of weighted particles $\{\psi_i, w_i\}$ (e.g. $\psi = (\lambda, \sigma^2, \tau^2)$)

$$\pi(\psi|D) \approx \sum w_i \delta_{\psi_i}(\psi)$$

where $\sum w_i = 1$ and $\delta_x(\cdot)$ is a Dirac delta function centred at x.

SMC for GP hyper-parameters

Gramacy and Polson 2012

In SMC we represent the posteriors by a cloud of weighted particles $\{\psi_i, w_i\}$ (e.g. $\psi = (\lambda, \sigma^2, \tau^2)$)

$$\pi(\psi|D) \approx \sum w_i \delta_{\psi_i}(\psi)$$

where $\sum w_i = 1$ and $\delta_x(\cdot)$ is a Dirac delta function centred at x.

As each new data point arrives we update the posterior by changing the weights to incorporate the new data point

SMC for GP hyper-parameters

Gramacy and Polson 2012

In SMC we represent the posteriors by a cloud of weighted particles $\{\psi_i, w_i\}$ (e.g. $\psi = (\lambda, \sigma^2, \tau^2)$)

$$\pi(\psi|D) \approx \sum w_i \delta_{\psi_i}(\psi)$$

where $\sum w_i = 1$ and $\delta_x(\cdot)$ is a Dirac delta function centred at x.

As each new data point arrives we update the posterior by changing the weights to incorporate the new data point

More formally, given data $\mathcal{D}_{1:n} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, let each particle contain the information

$$S_n^{(i)} = \{K, \lambda, \sigma^2, \tau^2\}$$

where K is the Gram matrix for $\mathcal{D}_{1:n}$. This information is everything we need to make predictions using the Gaussian process.

Particle learning of GP hyper-parameters

We exploit the relationship

$$\begin{aligned}\pi(S_{n+1}|\mathcal{D}_{1:n+1}) &= \int \pi(S_{n+1}|S_n, \mathcal{D}_{n+1})\pi(S_n|\mathcal{D}_{1:n+1})\mathrm{dS_n} \\ &\propto \int \pi(S_{n+1}|S_n, \mathcal{D}_{n+1})\pi(\mathcal{D}_{n+1}|S_n)\pi(S_n|\mathcal{D}_{1:n})\mathrm{dS_n}\end{aligned}$$

Particle learning of GP hyper-parameters

We exploit the relationship

$$\begin{aligned}\pi(S_{n+1}|\mathcal{D}_{1:n+1}) &= \int \pi(S_{n+1}|S_n, \mathcal{D}_{n+1})\pi(S_n|\mathcal{D}_{1:n+1})\mathrm{dS_n} \\ &\propto \int \pi(S_{n+1}|S_n, \mathcal{D}_{n+1})\pi(\mathcal{D}_{n+1}|S_n)\pi(S_n|\mathcal{D}_{1:n})\mathrm{dS_n}\end{aligned}$$

At time t, we have particles $\{S_n^{(i)}\}$ which we can use to approximate $\pi(\psi|\mathcal{D}_{1:n})$ and find $\pi(y^*|x^*, \mathcal{D}_{1:n})$ etc

1. Resample: Draw index $\zeta(i)$ for particle from $\{i\}_{i=1}^N$ according to weights
$$w_i \propto \pi(\mathcal{D}_{n+1}|S_n^{(i)}) = \pi(y_{n+1}|x_{n+1}, \mathcal{D}_n, \psi_n^{(i)})$$
2. Propagate
$$S_{n+1}^{(i)} \sim \pi(S_{n+1}|S_n^{\zeta(i)}, \mathcal{D}_{1:n+1})$$
which means simply finding K_{n+1} from K_n using the sequential matrix update equations.

Particle learning of GP hyper-parameters

This method soon suffers with particle degeneracy if the posteriors are very different to the priors

- i.e., $w_i \approx 1$ for some i, with $w_j \approx 0$ for $j \neq i$
- Can judge degeneracy with the effective sample size

$$ESS = \frac{1}{\sum w_i^2}$$

ESS$= N$ if $w_i = \frac{1}{N} \forall i$, but $ESS = 1$ in the case of extreme degeneracy.

Particle learning of GP hyper-parameters

This method soon suffers with particle degeneracy if the posteriors are very different to the priors

- i.e., $w_i \approx 1$ for some i, with $w_j \approx 0$ for $j \neq i$
- Can judge degeneracy with the effective sample size

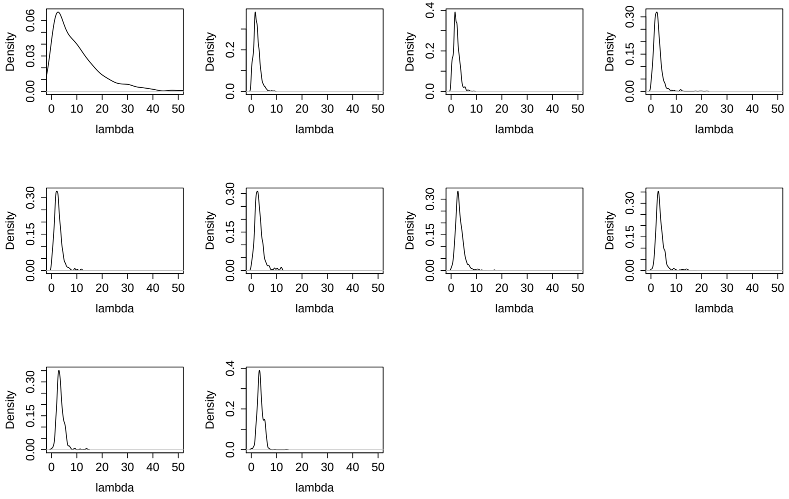
  $$ESS = \frac{1}{\sum w_i^2}$$

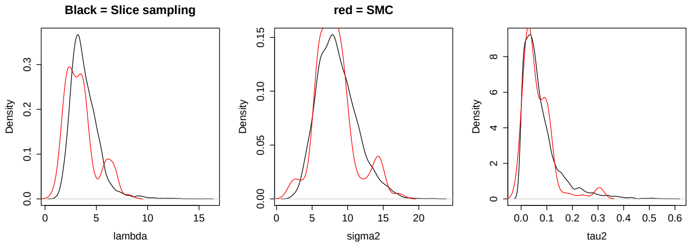
  ESS$= N$ if $w_i = \frac{1}{N} \forall i$, but $ESS = 1$ in the case of extreme degeneracy.

The particles can be replenished by perturbing them with an MCMC sampler that has $\pi(\psi|\mathcal{D}_n)$ as its invariant distribution

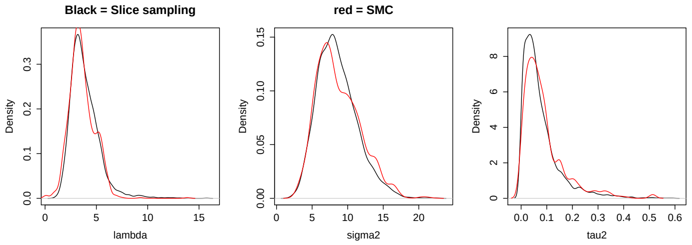
- e.g. run a slice sampler for k iterations
- but this is again $O(n^3)$

Progression of $\pi(\lambda|\mathcal{D}_n), \quad n = 1, \ldots, 10$





$N = 100$



$N = 1000$

In this toy example, the SMC ($N = 1000$, not tuned) takes as long as the slice sampling ($N = 5000$, not tuned)

However, as the number of data points grows, SMC quickly starts to outperform slice sampling in terms of cost (nb this only applies to sequential learning problems).

Conclusions