Fully Bayesian parameter inference for Markov processes

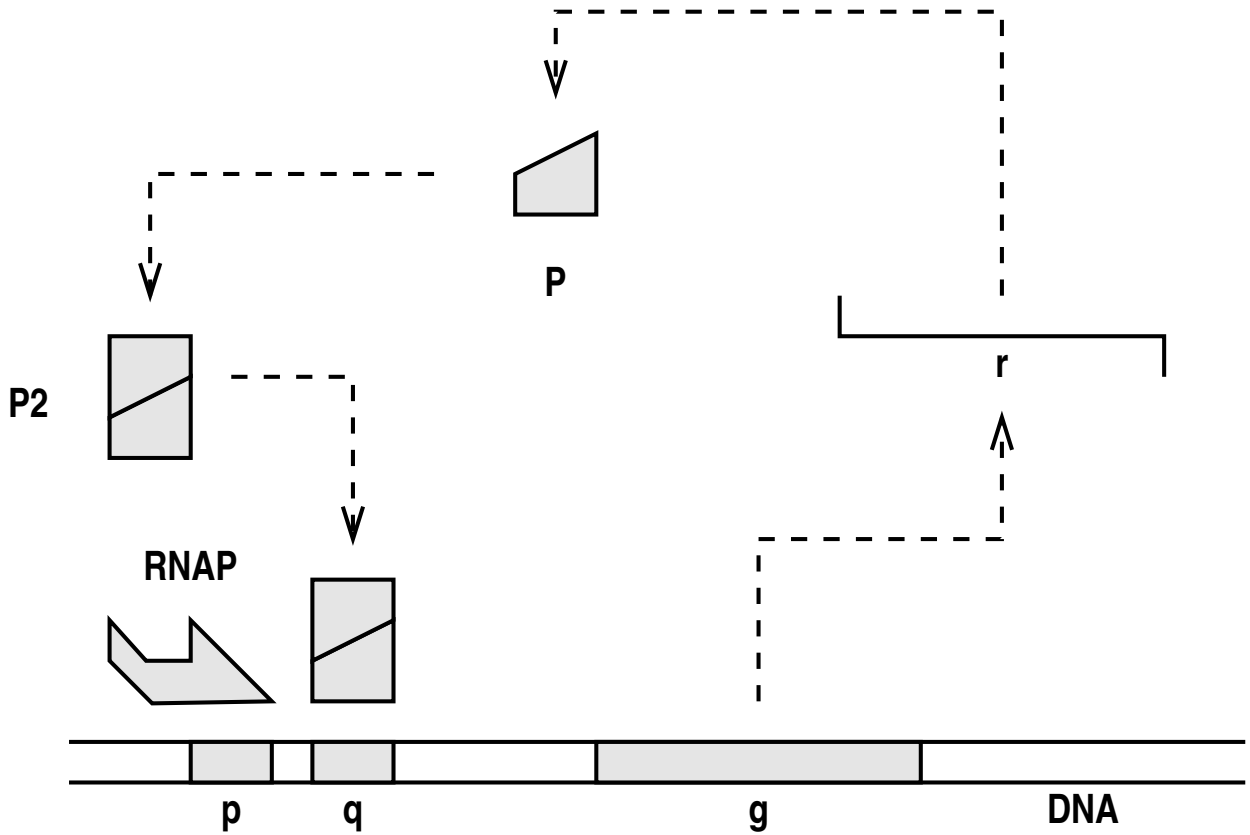
Darren Wilkinson
@darrenjw
darrenjw.github.io
Durham University, U.K.

Bayesian inference in HEP
Durham
26th May 2022
github.com/darrenjw/talks

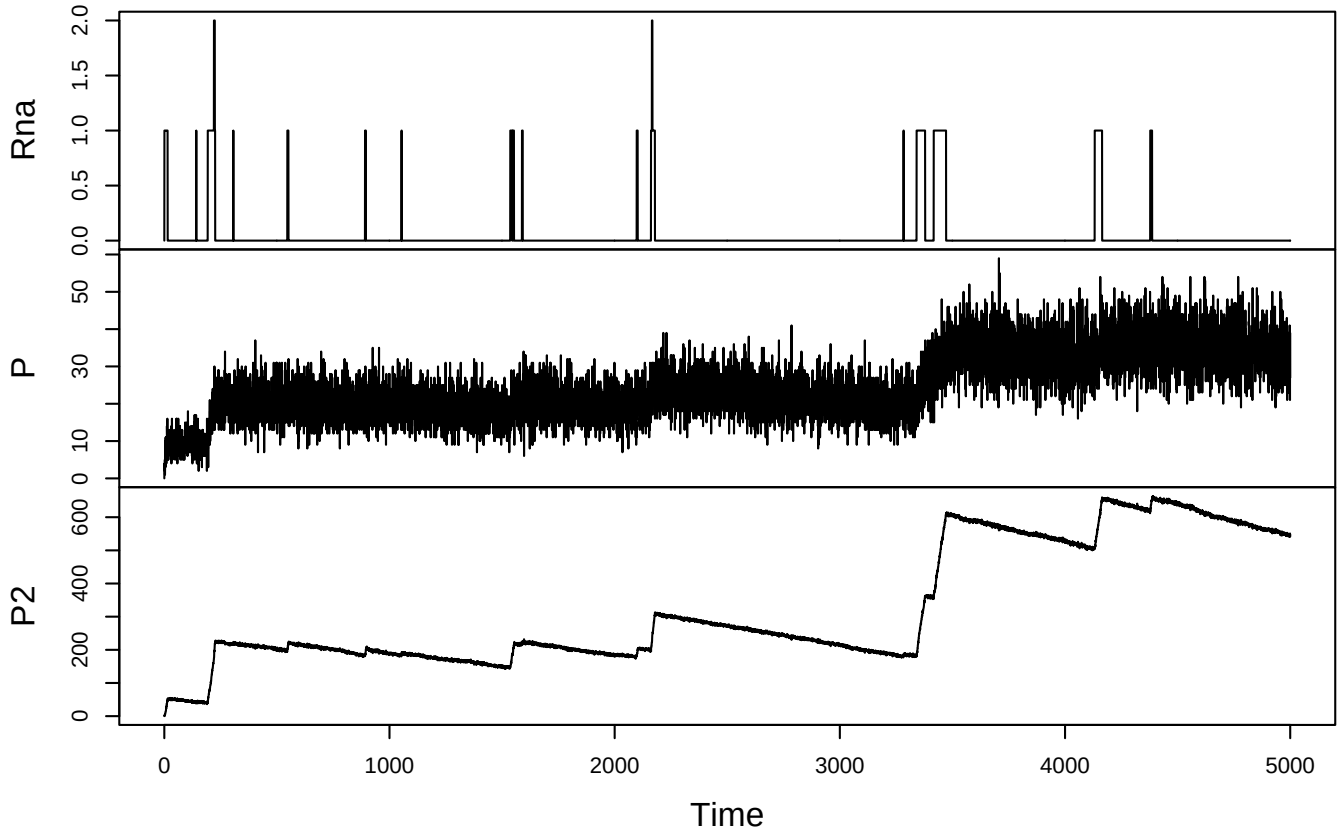
Overview

- Stochastic reaction networks, stochastic simulation and partially observed Markov process (POMP) models
- Modularity: separating model representation from simulation algorithm
- Likelihood-free MCMC for POMP models: separating simulation from inference
- Likelihood-free PMMH pMCMC, ABC and ABC–SMC
- Programming languages and libraries for MCMC-based fully Bayesian inference

Example — genetic auto-regulation



Simulated realisation of the auto-regulatory network



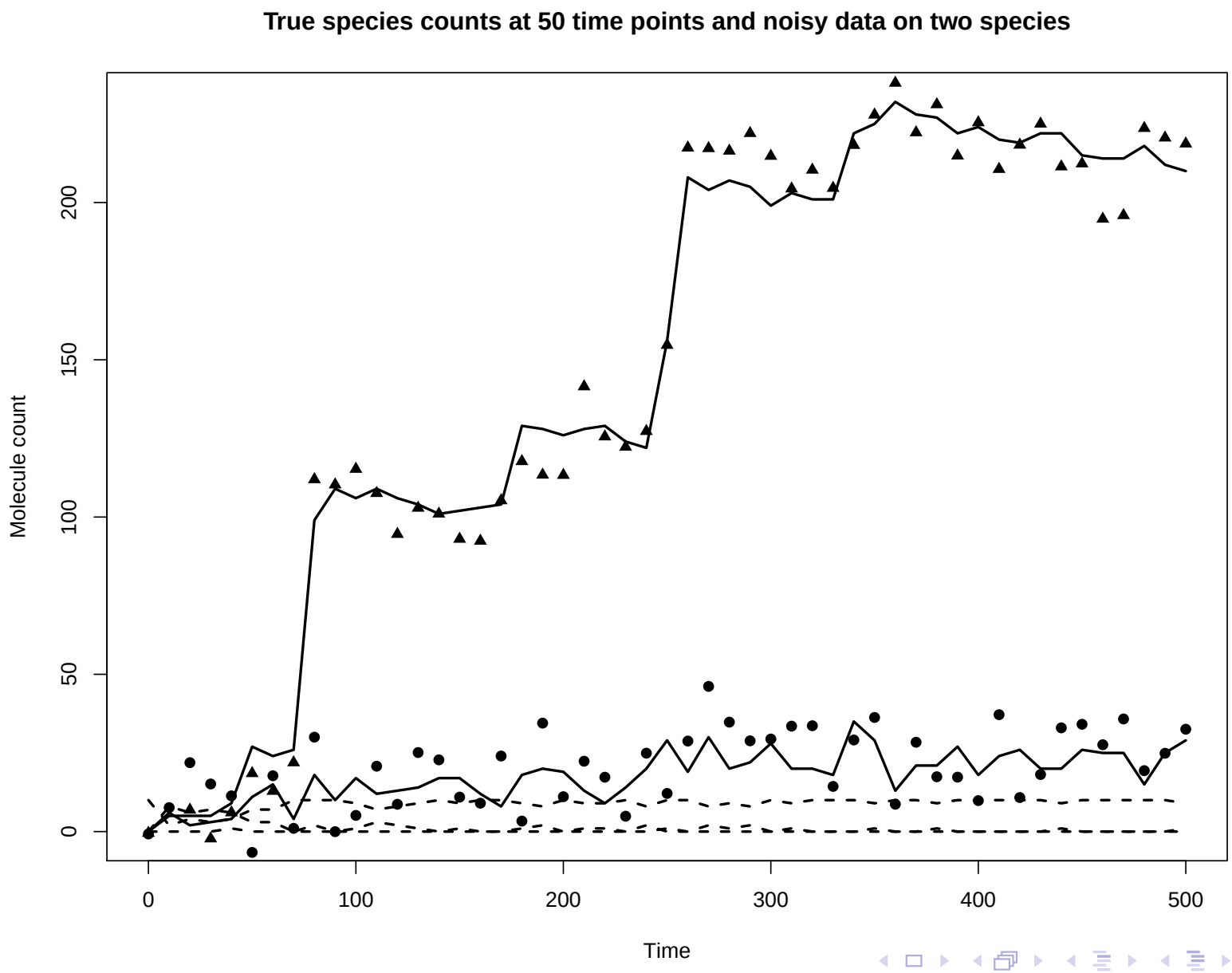
Modularity and simulation algorithms

- Separating models from simulation algorithms has many benefits
- Separating model parameters from models so that simulation algorithms don't need to know about parameters is similarly beneficial
- Models can be simulated in different ways, using different algorithms, and under different assumptions: exact/approximate, discrete/continuous, stochastic/deterministic, well-mixed/spatial, ...
- Model exchange formats, such as SBML, can be useful for understanding some of the issues

Parameter inference

- The auto-regulatory network model contains 5 species and 8 reactions
- Each reaction has an associated rate constant — these 8 rate constants may be subject to uncertainty
- The initial state of the model (5 species levels) may also be uncertain/unknown
- There could also be uncertainty about the structure of the reaction network itself — eg. presence/absence of particular reactions — this can be embedded into the parameter inference problem, but is often considered separately, and is not the subject of this talk
- We will focus here on using time course data on some aspect of one (or more) realisations of the underlying stochastic process in order to make inferences for any unknown parameters of the model

Partial, noisy data on the auto-reg model



Classes of Bayesian Monte Carlo algorithms

In this context there are 3 main classes of MC algorithms:

- ABC algorithms (likelihood–free)
  - Completely general (in principle) "global" Approximate Bayesian Computation algorithms, so just require a forward simulator, and don't rely on (eg.) Markov property, but typically very inefficient and approximate
- POMP algorithms (likelihood–free)
  - Typically "local" (particle) MCMC-based algorithms for Partially Observed Markov Processes, again only requiring a forward simulator, but using the Markov property of the process for improved computational efficiency and "exactness"
- Likelihood-based MCMC algorithms
  - More efficient (exact) MCMC algorithms for POMP models, working directly with the model representation, not using a forward simulator, and requiring the evaluation of likelihoods associated with the sample paths of the stochastic process

Modularity and model decoupling for inference

- Decoupling the model from the inference algorithm is just as important as separation of the model from a forward simulation algorithm
- The key characteristic of likelihood-free (or "plug-and-play") algorithms is that they separate inference algorithms from the forward simulator completely — this strong decoupling has many advantages, with the main disadvantage being the relative inefficiency of the inferential algorithms
- The likelihood-free algorithms rely heavily on forward simulation, so can immediately benefit from improvements in exact and approximate simulation technology
- There is no reason why efficient likelihood-based MCMC algorithms can't also be decoupled from the model representation, but doing so for a reasonably large and flexible class of models seems to be beyond the programming skills of most statisticians...

Partially observed Markov process (POMP) models

- Continuous-time Markov process: $\mathbf{X} = \{X_s | s \geq 0\}$ (for now, we suppress dependence on parameters, θ)
- Think about integer time observations (extension to arbitrary times is trivial): for $t \in \mathbb{N}, \quad \mathbf{X}_t = \{X_s | t-1 < s \leq t\}$
- Sample-path likelihoods such as $\pi(\mathbf{x}_t | x_{t-1})$ can often (but not always) be computed (but are often computationally difficult), but discrete time transitions such as $\pi(x_t | x_{t-1})$ are typically intractable
- Partial observations: $\mathcal{Y} = \{y_t | t = 1, 2, \ldots, T\}$ where

$$y_t | X_t = x_t \sim \pi(y_t | x_t), \qquad t = 1, \ldots, T,$$

where we assume that $\pi(y_t | x_t)$ can be evaluated directly (simple measurement error model)

Bayesian inference for latent process models

- Vector of model parameters, θ, the object of inference
- Prior probability distribution on θ, denoted $\pi(\theta)$
- Conditional on θ, we can simulate realisation of the stochastic process $\mathbf{X}$, with probability model $\pi(\mathbf{x}|\theta)$, which may be intractable
- Observational data $\mathcal{Y}$, determined from $\mathbf{x}$ and θ by a the probability model $\pi(\mathcal{Y}|\mathbf{x},\theta)$ — for "exact" algorithms we typically require that this model is tractable, but for ABC, we only need to be able to simulate from it
- Joint model $\pi(\theta,\mathbf{x},\mathcal{Y}) = \pi(\theta)\pi(\mathbf{x}|\theta)\pi(\mathcal{Y}|\mathbf{x},\theta)$
- Posterior distribution $\pi(\theta,\mathbf{x}|\mathcal{Y}) \propto \pi(\theta,\mathbf{x},\mathcal{Y})$
- If using Monte Carlo methods, easy to marginalise out $\mathbf{x}$ from samples from the posterior to get samples from the parameter posterior $\pi(\theta|\mathcal{Y})$

Likelihood-free PMMH pMCMC

- Particle Markov chain Monte Carlo (pMCMC) methods are a powerful tool for parameter inference in POMP models
- In the variant known as particle marginal Metropolis Hastings (PMMH), a (random walk) MH MCMC algorithm is used to explore parameter space, but at each iteration, a (bootstrap) particle filter (SMC algorithm) is run to calculate terms required in the acceptance probability
- The "magic" of pMCMC is that despite the fact that the particle filters are "approximate", pMCMC algorithms nevertheless have the "exact" posterior distribution of interest (either $\pi(\theta|\mathcal{Y})$ or $\pi(\theta, \mathbf{x}|\mathcal{Y})$) as their target
- If a sophisticated particle filter is used, pMCMC can be a reasonably efficient likelihood-based MCMC method — however, when a simple "bootstrap" particle filter is used, the entire process is "likelihood-free", but still "exact"

Particle MCMC (pMCMC)

- pMCMC is the only obvious practical option for constructing a global likelihood-free MCMC algorithm for POMP models which is exact (Andrieu et al, 2010)
- Start by considering a basic marginal MH MCMC scheme with target $\pi(\theta|\mathcal{Y})$ and proposal $f(\theta^\star|\theta)$ — the acceptance probability is $\min\{1, A\}$ where

$$A = \frac{\pi(\theta^\star)}{\pi(\theta)} \times \frac{f(\theta|\theta^\star)}{f(\theta^\star|\theta)} \times \frac{\pi(\mathcal{Y}|\theta^\star)}{\pi(\mathcal{Y}|\theta)}$$

- We can't evaluate the final terms, but if we had a way to construct a Monte Carlo estimate of the likelihood, $\hat{\pi}(\mathcal{Y}|\theta)$, we could just plug this in and hope for the best:

$$A = \frac{\pi(\theta^\star)}{\pi(\theta)} \times \frac{f(\theta|\theta^\star)}{f(\theta^\star|\theta)} \times \frac{\hat{\pi}(\mathcal{Y}|\theta^\star)}{\hat{\pi}(\mathcal{Y}|\theta)}$$

"Exact approximate" MCMC (the pseudo-marginal approach)

- Remarkably, provided only that $\mathrm{E}[\hat{\pi}(\mathcal{Y}|\theta)] = \pi(\mathcal{Y}|\theta)$, the stationary distribution of the Markov chain will be exactly correct (Beaumont, 2003, Andrieu & Roberts, 2009)
- Putting $W = \hat{\pi}(\mathcal{Y}|\theta)/\pi(\mathcal{Y}|\theta)$ and augmenting the state space of the chain to include W, we find that the target of the chain must be

$$\propto \pi(\theta)\hat{\pi}(\mathcal{Y}|\theta)\pi(w|\theta) \propto \pi(\theta|\mathcal{Y})w\pi(w|\theta)$$

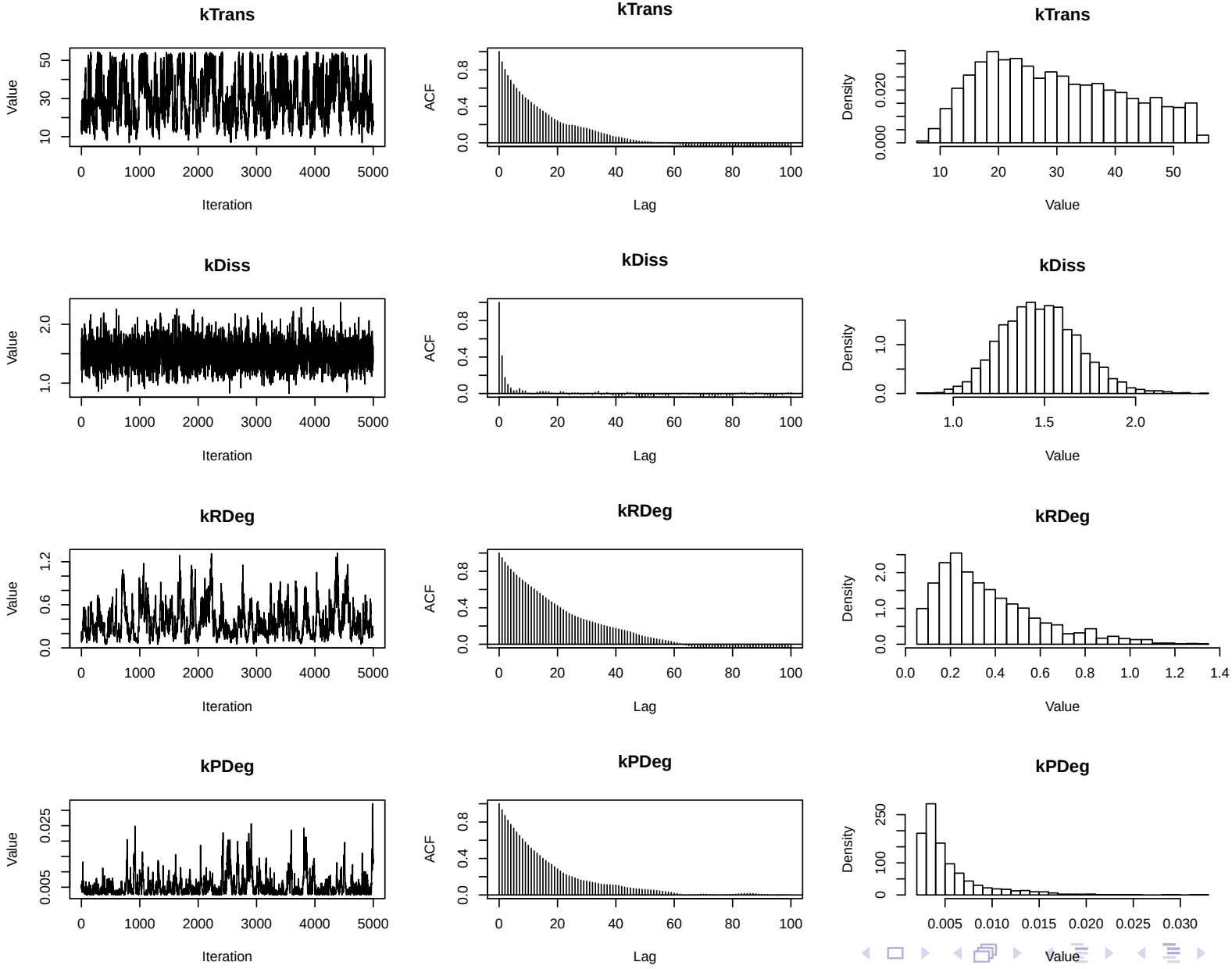
and so then the above "unbiasedness" property implies that $\mathrm{E}(W|\theta) = 1$, which guarantees that the marginal for θ is exactly $\pi(\theta|\mathcal{Y})$

Particle marginal Metropolis-Hastings (PMMH)

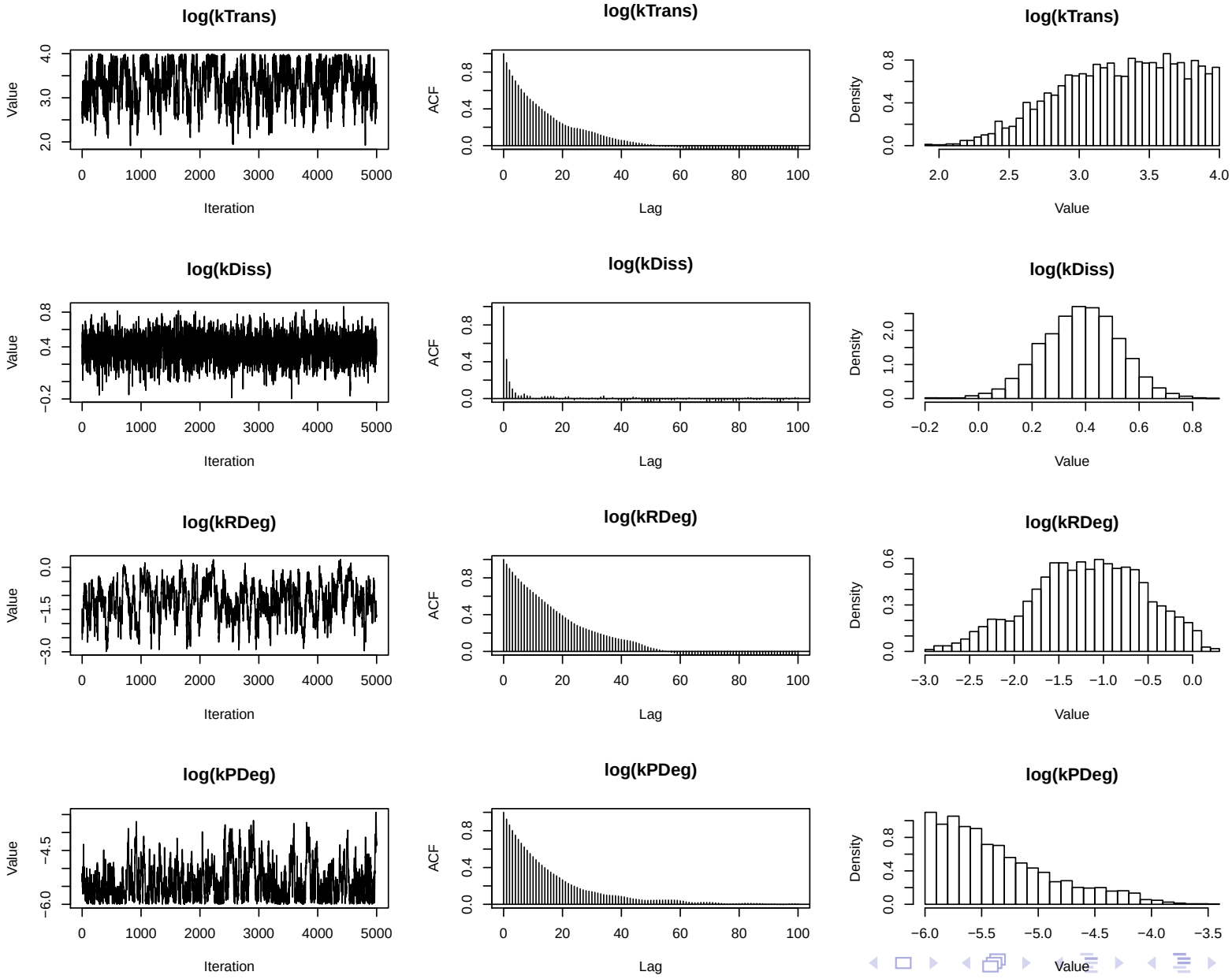
- Likelihood estimates constructed via importance sampling typically have this "unbiasedness" property, as do estimates constructed using a particle filter
- If a particle filter is used to construct the Monte Carlo estimate of likelihood to plug in to the acceptance probability, we get (a simple version of) the particle Marginal Metropolis Hastings (PMMH) pMCMC algorithm
- The full PMMH algorithm also uses the particle filter to construct a proposal for $\mathbf{x}$, and has target $\pi(\theta, \mathbf{x}|\mathcal{Y})$ — not just $\pi(\theta|\mathcal{Y})$
- The (bootstrap) particle filter relies only on the ability to forward simulate from the process, and hence the entire procedure is "likelihood-free"

See Golightly and W (2011) for further details

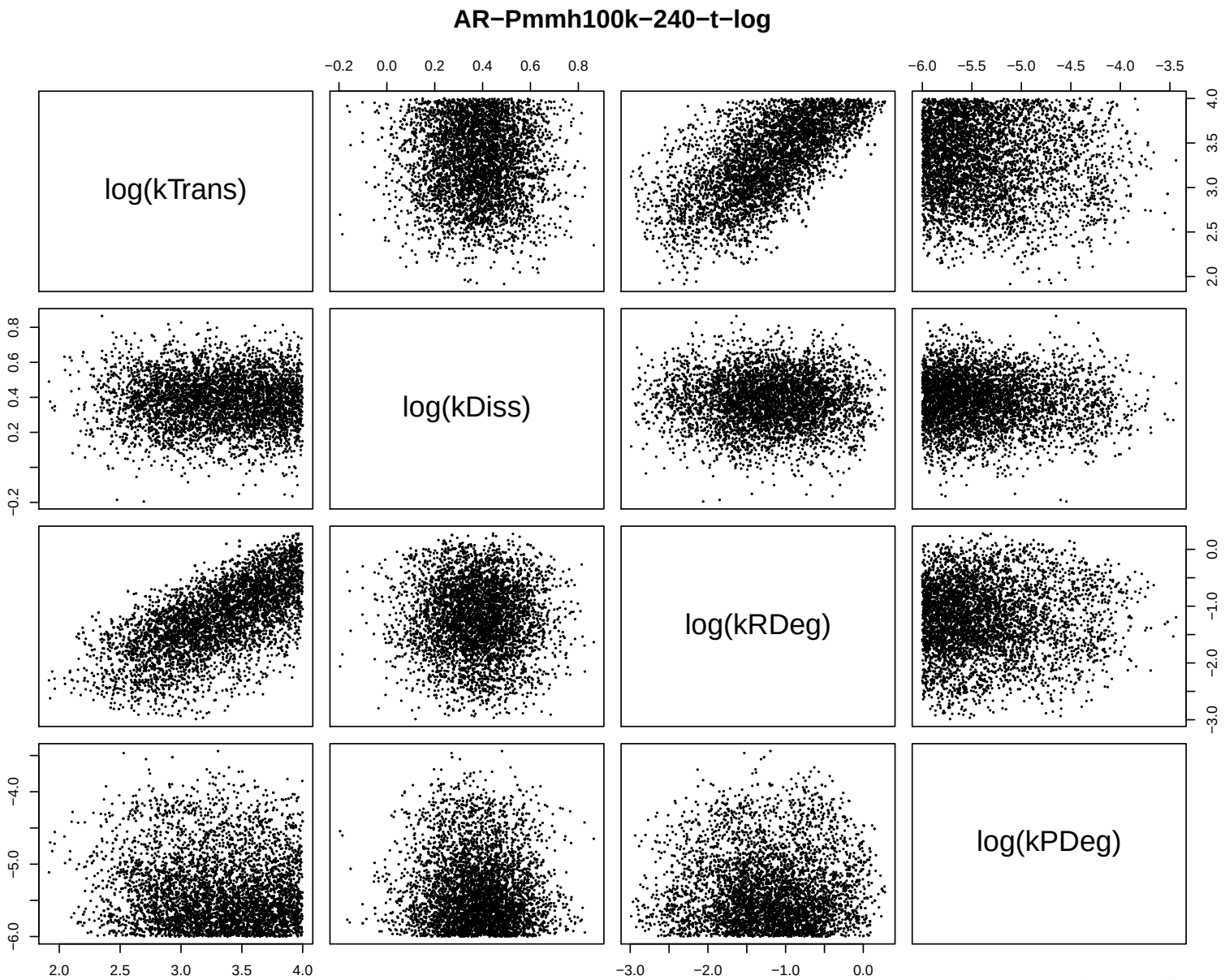
PMMH inference results



PMMH inference results



PMMH inference results



"Sticking" and tuning of PMMH

- As well as tuning the θ proposal variance, it is necessary to tune the number of particles, N in the particle filter — need enough to prevent the chain from sticking, but computational cost roughly linear in N
- Number of particles necessary depends on θ, but don't know θ *a priori*
- Initialising the sampler is non-trivial, since much of parameter space is likely to lead to likelihood estimates which are dominated by noise — how to move around when you don't know which way is "up"?!
- Without careful tuning and initialisation, burn-in, convergence and mixing can all be very problematic, making algorithms painfully slow...

General SIR particle filter

- At time t, we have (after resampling) an equally weighted sample from $\pi(x_t|y_{1:t})$
- At time $t+1$, we want a weighted sample from $\pi(x_{t+1}|y_{1:t+1})$, though in fact useful to construct a sample from $\pi(x_{t+1}, x_t|y_{1:t+1})$, and then marginalise down as required
- Target $\propto \pi(y_{t+1}|x_{t+1})\pi(x_{t+1}|x_t)\pi(x_t|y_{1:t})$ and proposal is $f(x_{t+1}|x_t, y_{t+1})\pi(x_t|y_{1:t})$, for some $f(\cdot)$, leading to unnormalised weight

$$w_{t+1} = \frac{\pi(y_{t+1}|x_{t+1})\pi(x_{t+1}|x_t)}{f(x_{t+1}|x_t, y_{t+1})}$$

- LF choice is $f(x_{t+1}|x_t, y_{t+1}) = \pi(x_{t+1}|x_t)$, otherwise need to evaluate the discrete time transition density

Weights and RN derivatives

- For Markov processes with intractable kernels, make the target $\pi(\mathbf{x}_{t+1}, x_t|y_{1:t+1})$ and then marginalise down to $\pi(x_{t+1}|y_{1:t+1})$ if required
- The proposal path will be of the form $f(\mathbf{x}_{t+1}|x_t, y_{t+1})\pi(x_t|y_{1:t})$, leading to weight

$$w_{t+1} = \pi(y_{t+1}|x_{t+1})\frac{\pi(\mathbf{x}_{t+1}|x_t)}{f(\mathbf{x}_{t+1}|x_t, y_{t+1})}$$

- The expected weight is $\pi(y_{t+1}|y_{1:t})$, as needed for pseudo-marginal MCMC
- Formally,

$$w_{t+1} = \pi(y_{t+1}|x_{t+1})\frac{d\mathbb{P}}{d\mathbb{Q}}(\mathbf{x}_{t+1}|x_t),$$

the RN derivative of the true (unconditioned) diffusion wrt the proposal process

Pros and cons of pMCMC

- Most obvious application is to POMP models — less general than ABC
- It targets the "exact" posterior distribution, irrespective of the choices of tuning constants!
- In practice, for finite length runs, the pMCMC output tends to be slightly under-dispersed relative to the true posterior ("missing the tails")
- Parallelises fine over multiple cores on a single machine, but less well over a cluster
- Although the theory underpinning pMCMC is non-trivial, implementing likelihood-free PMMH is straightforward, and has the advantage that it targets the "exact" posterior distribution

Languages and libraries for MCMC

- Various approaches to creating MCMC algorithms for particular problems:
  - Use a stand-alone probabilistic programming language (PPL)
  - Use a PPL embedded in a general purpose programming language (eg. Python or R)
  - Write a custom sampler in an appropriate programming language (using scientific libraries, tensor computation frameworks, auto-diff frameworks, etc.)
- Stand-alone PPLs:
  - JAGS is old, but quite general and widely used. It can be very inefficient for large, complex models. There are interfaces, `rjags` and `PyJAGS`, for using it from R and Python.
  - Stan is now probably the most popular PPL. It is typically much more efficient than JAGS for complex models, but uses gradients, so can't (directly) handle discrete parameters. There are interfaces such as `rstan` and `PyStan` for using it from R and Python.

Python ecosystem for Bayesian inference and MCMC

- PyMC3 is probably the most popular PPL embedded in Python. It uses Theano for a backend. PyMC4 was going to use TensorFlow, but this was abandoned. PyMC3 is currenly moving to a JAX backend (currently experimental, via NumPyro), but will eventually support multiple backends, via Aesara (a fork of Theano)
- Pyro uses PyTorch as a back-end, but the popular NumPyro fork uses JAX
- JAX is a pure functional eDSL for tensor/array computation and automatic differentiation. It can JIT-compile to run very fast on (multiple) CPU and GPU. It turns out to be extremely well-suited to sampling applications such as MCMC. If you are doing any kind of ML in Python, it's worth spending some time learning about JAX. BlackJAX is a library of samplers for log-posteriors described using JAX.

Functional languages for scalable Bayesian computation

- JAX is a pure functional DSL embedded in Python
- Functional languages are easier to analyse, optimise, compile, parallelise, distribute, automatically differentiate, etc., than imperative languages
- MCMC for logistic regression using multiple languages and libraries: `github.com/darrenjw/logreg`
- DEX is a new experimental functional language for array processing and automatic differentiation (written in Haskell)
- Scala is a general-purpose functional programming language, well-suited to building scalable (MCMC) samplers
- Apache Spark is a Scala framework for distributing big data ML workloads over a (large) cluster (in the cloud)

`github.com/darrenjw/talks`