Biips: A software for **B**ayesian **i**nference with **i**nteracting **p**article **s**ystems

Probabilistic Programming Reading Group

Adrien Todeschini[†], François Caron[*], Pierrick Legrand[†], Pierre Del Moral[‡] and Marc Fuentes[†]

[†]Inria Bordeaux, [*]Univ. Oxford, [‡]UNSW Sydney

Oxford, October 2014

Outline

Summary

Context

Biips = **B**ayesian **i**nference with **i**nteracting **p**article **s**ystems

Bayesian inference

- Sample from a posterior distribution $p(X|Y) = \frac{p(X,Y)}{p(Y)}$
- High dimensional, arbitrary complexity
- Simulation methods: MCMC, SMC...

Motivation

- Last 20 years: success of SMC in many applications
- No general and easy-to-use software for SMC

Context

Biips = **B**ayesian **i**nference with **i**nteracting **p**article **s**ystems

Bayesian inference

- Sample from a posterior distribution $\boldsymbol{p(X|Y) = \frac{p(X,Y)}{p(Y)}}$
- High dimensional, arbitrary complexity
- Simulation methods: MCMC, SMC...

Motivation

- Last 20 years: success of SMC in many applications
- No general and easy-to-use software for SMC

Context

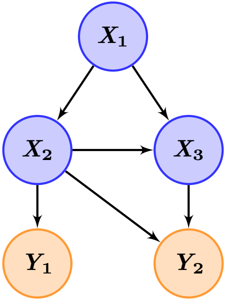
Biips = **B**ayesian **i**nference with **i**nteracting **p**article **s**ystems

Objectives

- BUGS language compatible
- Extensibility: user-defined functions/samplers
- Black-box SMC inference engine
- Interfaces with popular software: Matlab/Octave, R
- Post-processing

Summary

Graphical models

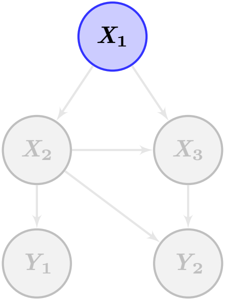


Directed acyclic graph

The graph displays a factorization of the joint distribution:

$$p(x_{1:3}, y_{1:2})$$

Graphical models

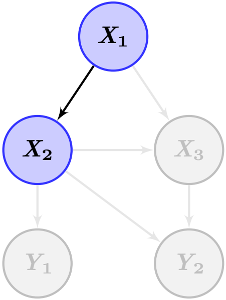


Directed acyclic graph

The graph displays a factorization of the joint distribution:

$$p(x_{1:3}, y_{1:2}) = p(x_1)\ p(x_2|x_1)\ p(y_1|x_2)\ p(x_3|x_1, x_2)\ p(y_2|x_2, x_3)$$

Graphical models

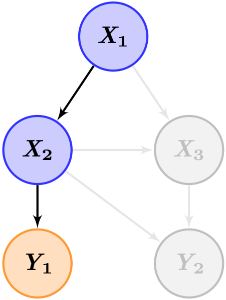


Directed acyclic graph

The graph displays a factorization of the joint distribution:

$$p(x_{1:3}, y_{1:2}) = p(x_1)\ p(x_2|x_1)\ p(y_1|x_2)\ p(x_3|x_1, x_2)\ p(y_2|x_2, x_3)$$

Graphical models

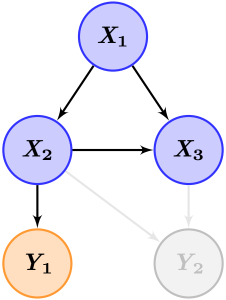


Directed acyclic graph

The graph displays a factorization of the joint distribution:

$$p(x_{1:3}, y_{1:2}) = p(x_1)\ p(x_2|x_1)\ p(y_1|x_2)$$
$$p(x_3|x_1, x_2)\ p(y_2|x_2, x_3)$$

Graphical models

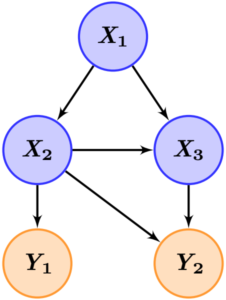


Directed acyclic graph

The graph displays a factorization of the joint distribution:

$$p(x_{1:3}, y_{1:2}) = p(x_1)\ p(x_2|x_1)\ p(y_1|x_2)\ p(x_3|x_1, x_2)\ p(y_2|x_2, x_3)$$

Graphical models



Directed acyclic graph

The graph displays a factorization of the joint distribution:

$$p(x_{1:3}, y_{1:2}) = p(x_1)\ p(x_2|x_1)\ p(y_1|x_2)\ p(x_3|x_1, x_2)\ p(y_2|x_2, x_3)$$

BUGS language

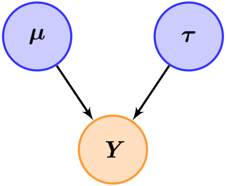
- S-like declarative language for describing graphical models
- Stochastic relations
- Deterministic relations

BUGS language

- S-like declarative language for describing graphical models
- Stochastic relations
- Deterministic relations

Linear regression:

```
model {
   Y ~ dnorm(mu, tau)



}
```

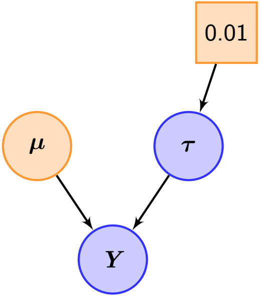


BUGS language

- S-like declarative language for describing graphical models
- Stochastic relations
- Deterministic relations

Linear regression:

```
model {
   Y ~ dnorm(mu, tau)
   tau ~ dgamma(0.01, 0.01)



}
```

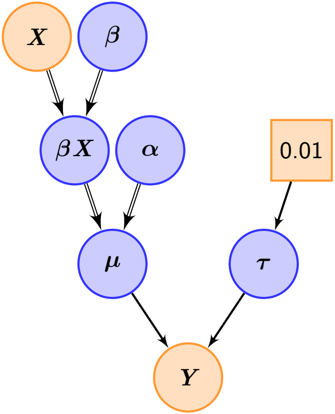


BUGS language

- S-like declarative language for describing graphical models
- Stochastic relations
- Deterministic relations

Linear regression:

```
model {
   Y ~ dnorm(mu, tau)
   tau ~ dgamma(0.01, 0.01)
   mu <- beta * X + alpha


}
```

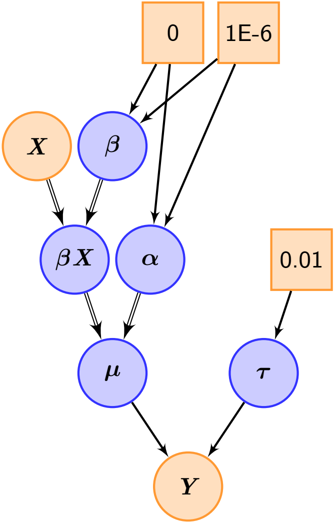


BUGS language

- S-like declarative language for describing graphical models
- Stochastic relations
- Deterministic relations

Linear regression:

```
model {
    Y ~ dnorm(mu, tau)
    tau ~ dgamma(0.01, 0.01)
    mu <- beta * X + alpha
    alpha ~ dnorm(0, 1E-6)
    beta ~ dnorm(0, 1E-6)
}
```

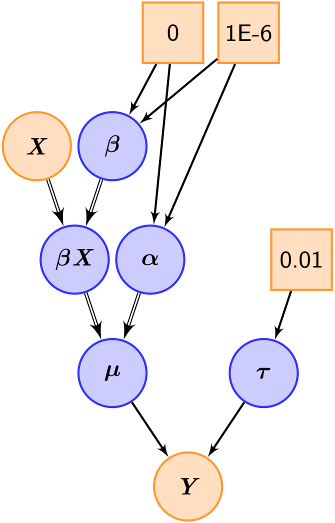


BUGS language

- S-like declarative language for describing graphical models
- Stochastic relations
- Deterministic relations

Linear regression:

```
model {
    Y ∼ dnorm(mu, tau)
    tau ∼ dgamma(0.01, 0.01)
    mu <- beta * X + alpha
    alpha ∼ dnorm(0, 1E-6)
    beta ∼ dnorm(0, 1E-6)
}
```



Goal:

Estimate $p(\alpha, \beta, \tau | X, Y)$

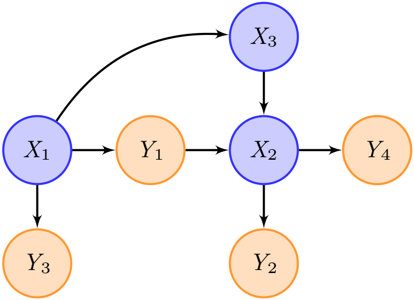
BUGS software using MCMC

BUGS = **B**ayesian inference **U**sing **G**ibbs **S**ampling

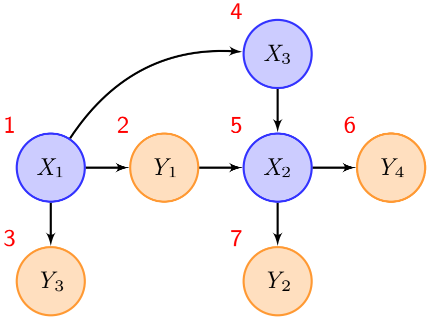
- WinBUGS, OpenBUGS, JAGS [Plummer, 2012]
- Expert system automatically derives MCMC methods (Gibbs, Slice, Metropolis, ...) in a 'black-box' fashion
- Very popular among practitioners, applying MCMC methods to a wide range of applications [Lunn et al., 2012]

Summary

Ordering of the graph

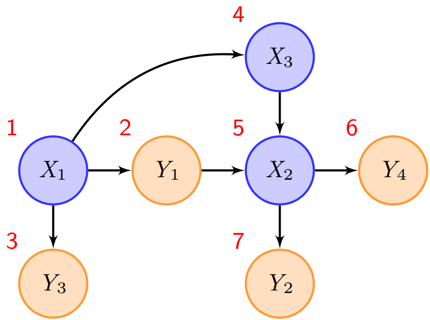


Ordering of the graph

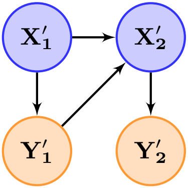


Topological sort (with priority to measurement nodes):
$(X_1, Y_1, Y_3, X_3, X_2, Y_4, Y_2)$

Ordering of the graph



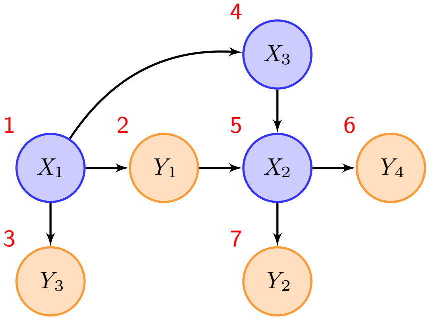
Rearrangement of the directed acyclic graph:



Topological sort (with priority to measurement nodes):

$$(\underbrace{X_1}_{\mathbf{X}'_1}, \underbrace{Y_1, Y_3}_{\mathbf{Y}'_1}, \underbrace{X_3, X_2}_{\mathbf{X}'_2}, \underbrace{Y_4, Y_2}_{\mathbf{Y}'_2})$$

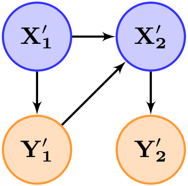
Ordering of the graph



Topological sort (with priority to measurement nodes):

$$(\underbrace{X_1}_{\mathbf{X}'_1}, \underbrace{Y_1, Y_3}_{\mathbf{Y}'_1}, \underbrace{X_3, X_2}_{\mathbf{X}'_2}, \underbrace{Y_4, Y_2}_{\mathbf{Y}'_2})$$

Rearrangement of the directed acyclic graph:



The statistical model decomposes as

$$p(x'_1, x'_2, y'_1, y'_2) = p(x'_1)p(y'_1|x'_1) p(x'_2|x'_1, y'_1)p(y'_2|x'_2)$$

SMC algorithm

More generally, assume that we have sorted variables $(X_1, Y_1, \ldots, X_n, Y_n)$.
The statistical model decomposes as

$$p(x_{1:n}, y_{1:n}) = p(x_1)p(y_1|x_1) \prod_{t=2}^{n} p(x_t|\mathrm{pa}(x_t))p(y_t|\mathrm{pa}(y_t))$$

where $\mathrm{pa}(x)$ denotes the set of parents of variable x.

SMC algorithm

- A.k.a. interacting MCMC, particle filtering, sequential Monte Carlo methods (SMC) ...
- Sequentially sample from conditional distributions of increasing dimension

$$\pi_1(x_1|y_1) \rightarrow \pi_2(x_{1:2}|y_{1:2}) \rightarrow \ldots \rightarrow \pi_n(x_{1:n}|y_{1:n})$$

where, for $t = 1, \ldots, n$

$$\pi_t(x_{1:t}|y_{1:t}) = \frac{p(x_{1:t}, y_{1:t})}{p(y_{1:t})}$$

Two stochastic mechanisms:

- **Mutation/Exploration**
- **Selection**

[Doucet et al., 2001, Del Moral, 2004, Doucet and Johansen, 2010]

SMC Algorithm

Standard SMC algorithm

For $t = 1, \dots, n$

- For $i = 1, \dots, N$
  - Sample: $X_{t,t}^{(i)} \sim q_t$ and let $X_{t,1:t}^{(i)} = (\widetilde{X}_{t-1,1:t-1}^{(i)}, X_{t,t}^{(i)})$
  - Weight: $w_t^{(i)} = \frac{\pi(y_t|\text{pa}(y_t))\pi(x_{t,t}^{(i)}|\text{pa}(x_{t,t}^{(i)}))}{q_t(x_{t,t}^{(i)})}$
  - Normalize: $W_t^{(i)} = \frac{w_t^{(i)}}{\sum_{j=1}^N w_t^{(j)}}$
- Resample: $\{X_{t,1:t}^{(i)}, W_t^{(i)}\}_{i=1,\dots,N} \to \{\widetilde{X}_{t,1:t}^{(i)}, \frac{1}{N}\}_{i=1,\dots,N}$

Outputs

- Weighted particles $(W_t^{(i)}, X_{t,1:t}^{(i)})_{i=1,\dots,N}$ for $t = 1, \dots, n$
- Estimate of the marginal likelihood $\widehat{Z} = \prod_{t=1}^n \left(\frac{1}{N}\sum_{i=1}^N w_t^{(i)}\right)$

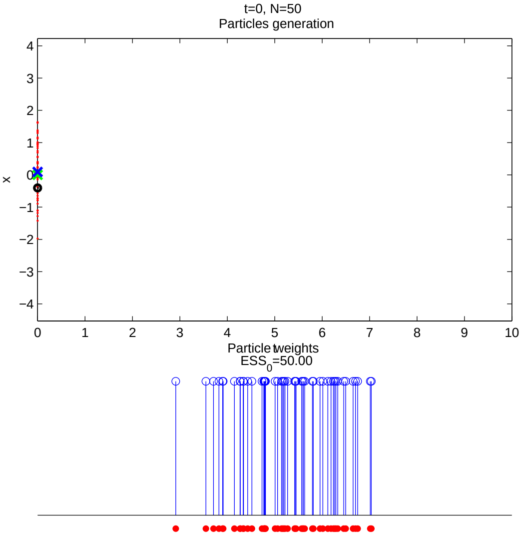
SMC algorithm

Marginal distributions

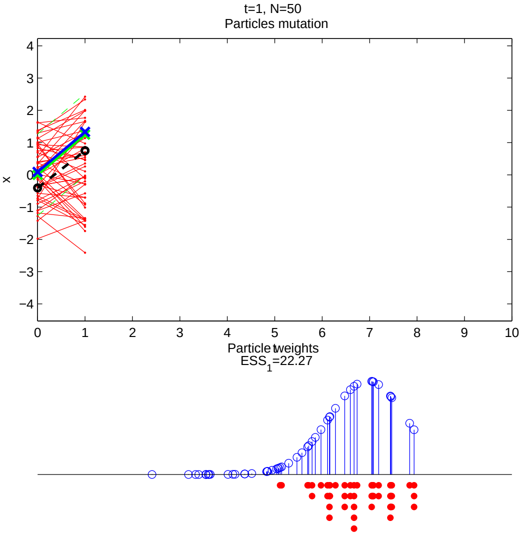
$$\pi_1(x_1|y_1) \quad \rightarrow \quad \pi_2(x_{1:2}|y_{1:2}) \quad \rightarrow \ldots \rightarrow \quad \pi_n(x_{1:n}|y_{1:n})$$

Filtering:	$\pi_1(x_1\|y_1)$	$\rightarrow$	$\pi_2(x_2\|y_{1:2})$	$\rightarrow \ldots \rightarrow$	$\pi_n(x_n\|y_{1:n})$
Smoothing:	$\pi_1(x_1\|y_{1:n})$	$\rightarrow$	$\pi_2(x_2\|y_{1:n})$	$\rightarrow \ldots \rightarrow$	$\pi_n(x_n\|y_{1:n})$

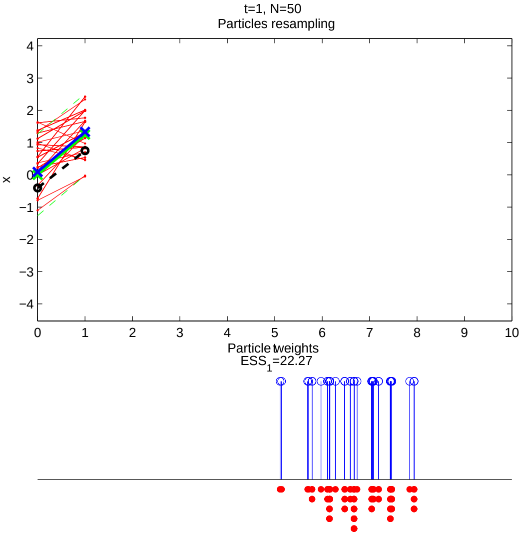
Example: hidden Markov/state space model



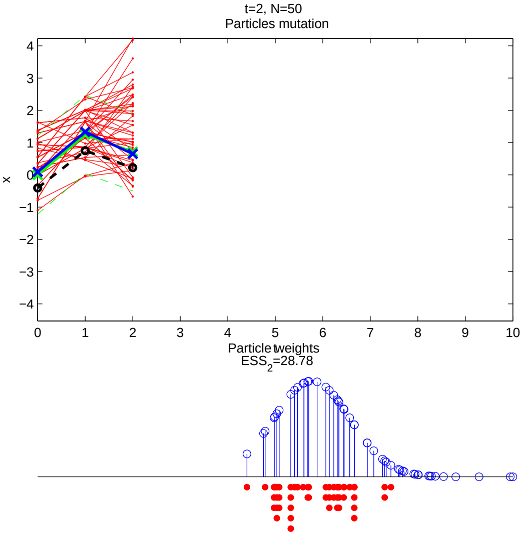
Example: hidden Markov/state space model



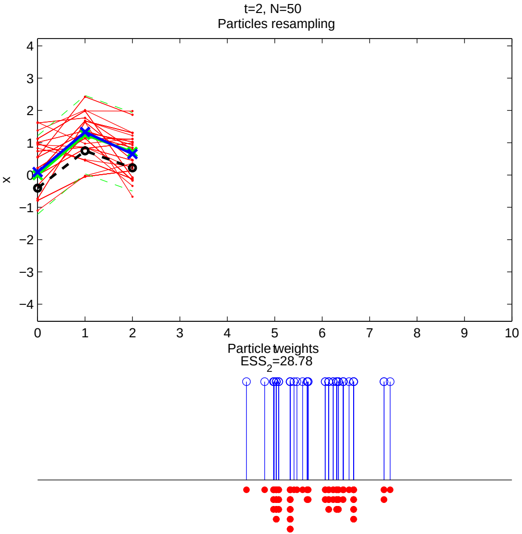
Example: hidden Markov/state space model



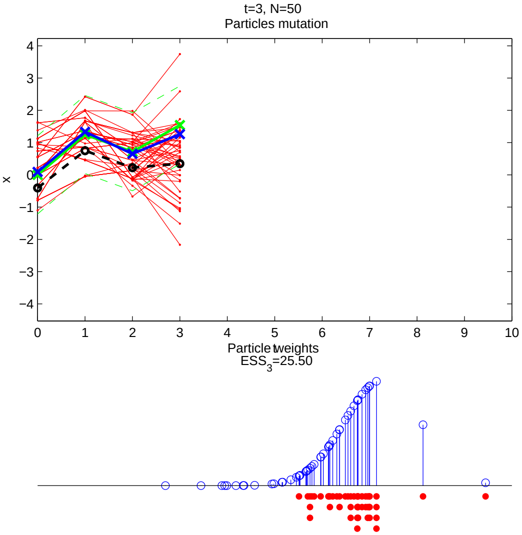
Example: hidden Markov/state space model



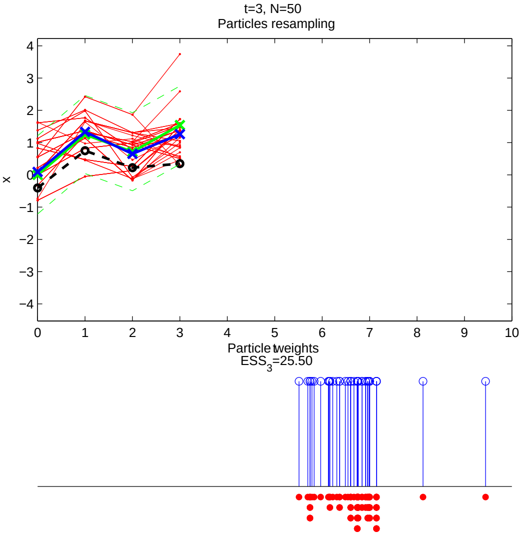
Example: hidden Markov/state space model



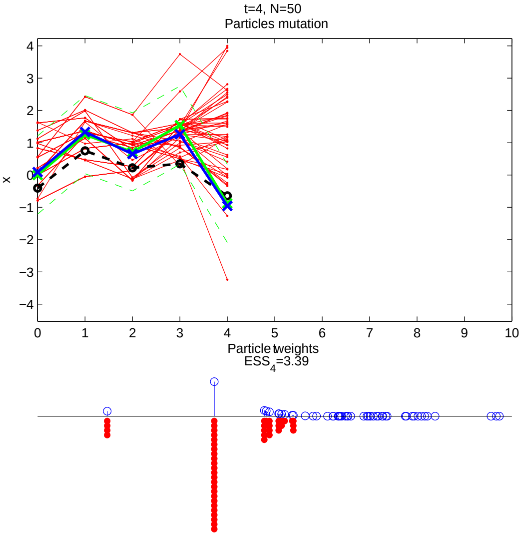
Example: hidden Markov/state space model



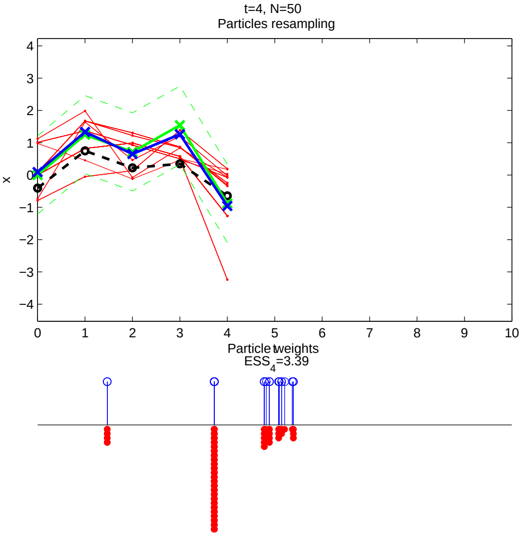
Example: hidden Markov/state space model



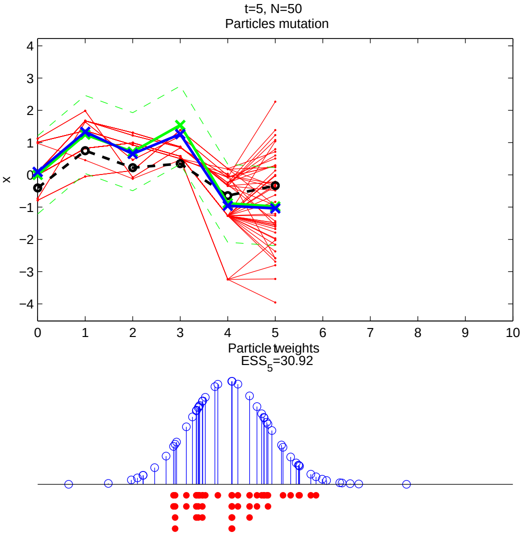
Example: hidden Markov/state space model



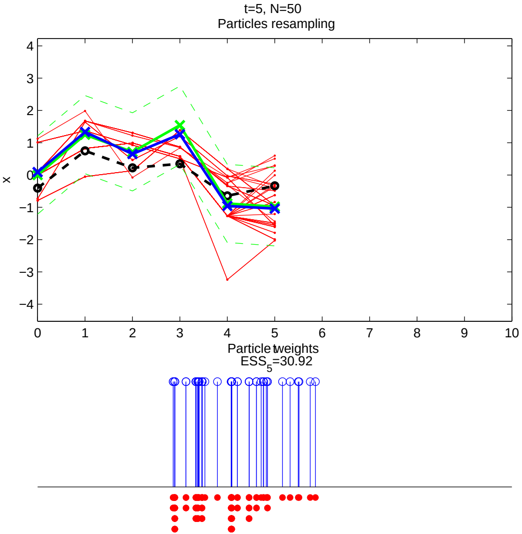
Example: hidden Markov/state space model



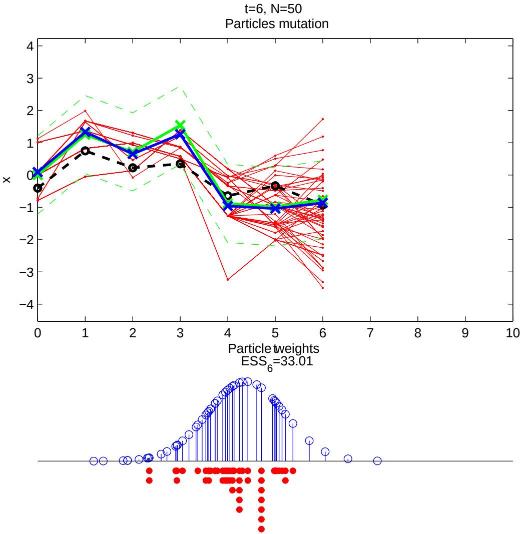
Example: hidden Markov/state space model



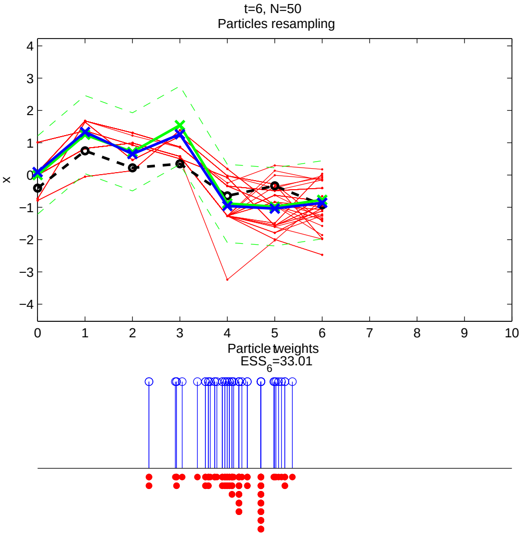
Example: hidden Markov/state space model



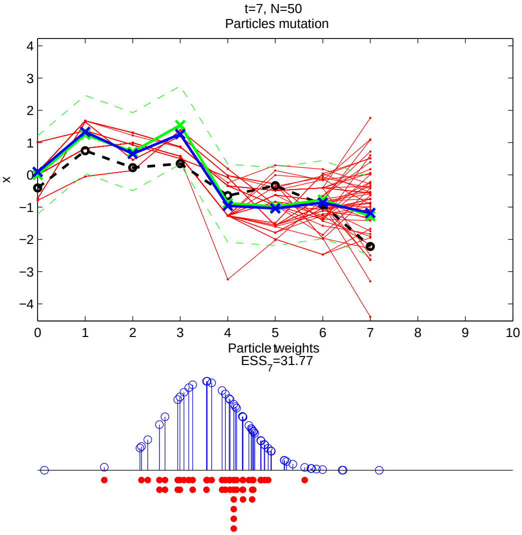
Example: hidden Markov/state space model



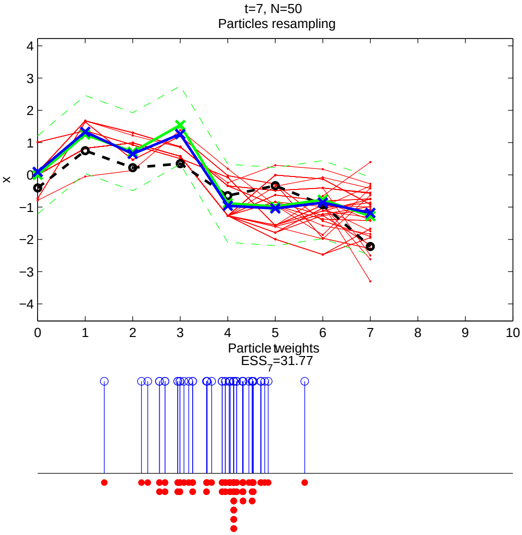
Example: hidden Markov/state space model



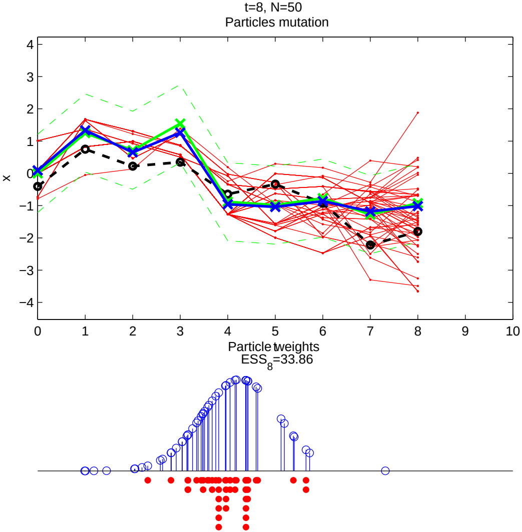
Example: hidden Markov/state space model



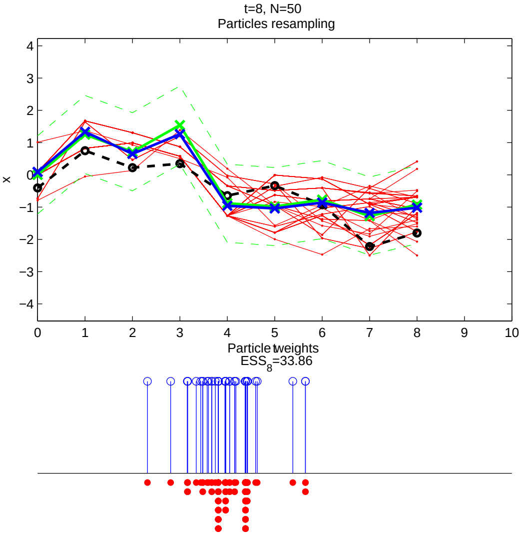
Example: hidden Markov/state space model



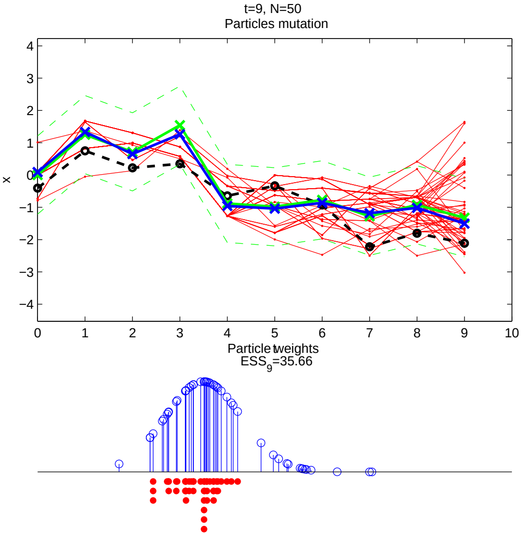
Example: hidden Markov/state space model



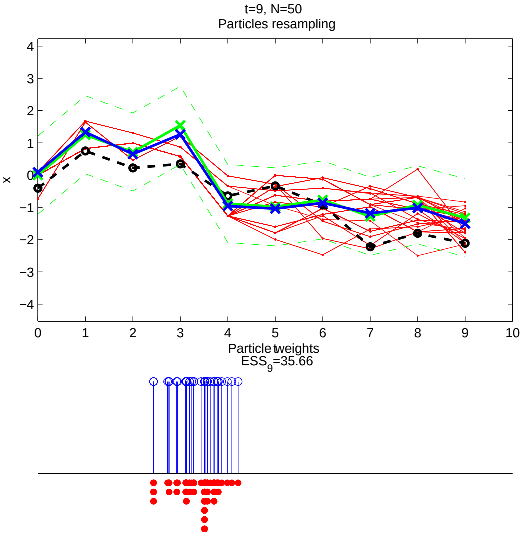
Example: hidden Markov/state space model



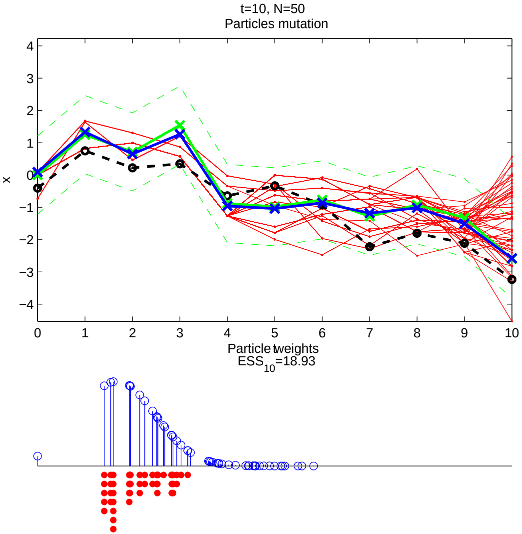
Example: hidden Markov/state space model



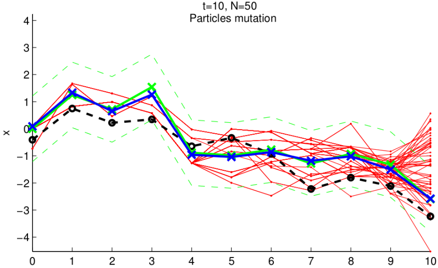
Example: hidden Markov/state space model



Example: hidden Markov/state space model



Limitations and diagnosis of SMC algorithms



For a given $t \leq n$, for each unique value $X'^{(k)}_{n,t}$, $k = 1, \ldots, K_{n,t}$, let $W'^{(k)}_{n,t} = \sum_{i | X^{(i)}_t = X'^{(k)}_t} W^{(i)}_n$ be its associated total weight. A measure of the quality of the approximation of the posterior distribution $p(x_{t:n} | y_{1:n})$ is given by the smoothing effective sample size (**SESS**):

$$\mathbf{SESS}_t = \frac{1}{\sum_{k=1}^{K_{n,t}} \left(W'^{(k)}_{n,t} \right)^2} \qquad (1)$$

with $1 \leq \mathbf{SESS}_t \leq N$.

Summary

Context

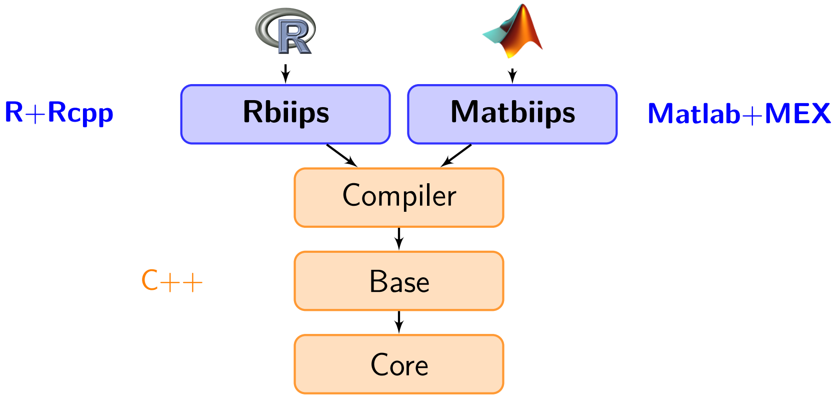
Graphical models and BUGS language

SMC

Matbiips

Particle MCMC

Technical implementation



- Interfaces: Matlab/Octave, R
- Multi-platform: Windows, Linux, Mac OSX
- Free and open source (GPL)

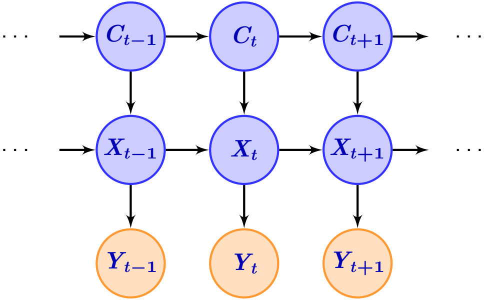
Switching Stochastic Volatility (SSV)

Let Y_t be the response variable and X_t the unobserved log-volatility of Y_t. For $t = 1, \ldots, n$

$$X_t | (X_{t-1} = x_{t-1}, C_t = c_t) \sim \mathcal{N}(\alpha_{c_t} + \phi x_{t-1}, \sigma^2)$$
$$Y_t | X_t = x_t \sim \mathcal{N}(0, \exp(x_t))$$

The regime variables C_t follow a two-state Markov process with transition probabilities

$$p_{ij} = \Pr(C_t = j | C_{t-1} = i), \quad \text{for } i, j = 1, 2$$



SSV model in BUGS language

switch_stoch_volatility.bug

```
model
{
  c[1] ~ dcat(pi[c0,])
  mu[1] <- alpha[1]*(c[1]==1) + alpha[2]*(c[1]==2)  + phi*x0
  x[1] ~ dnorm(mu[1], 1/sigma^2)
  y[1] ~ dnorm(0, exp(-x[1]))
  for (t in 2:t_max)
  {
    c[t] ~ dcat(ifelse(c[t-1]==1, pi[1,], pi[2,]))
    mu[t] <- alpha[1]*(c[t]==1) + alpha[2]*(c[t]==2) + phi*x[t-1]
    x[t] ~ dnorm(mu[t], 1/sigma^2)
    y[t] ~ dnorm(0, exp(-x[t]))
  }
}
```

Model compilation

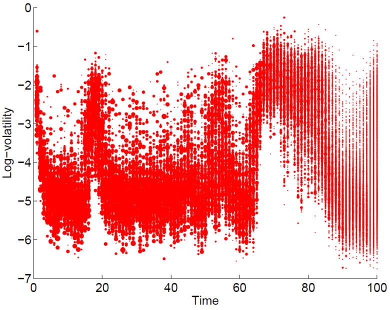
Matbiips

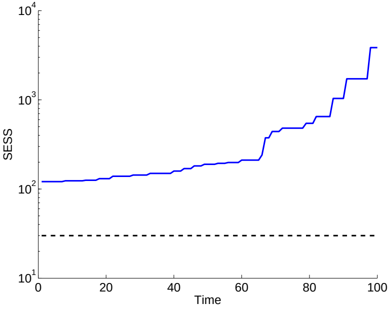
```
sigma = .4;alpha = [-2.5; -1]; phi = .5; c0 = 1; x0 = 0; t_max =
    200;
pi = [.9, .1; .1, .9];
data = struct('t_max', t_max, 'sigma', sigma,...
        'alpha', alpha, 'phi', phi, 'pi', pi, 'c0', c0, 'x0', x0);
model_file = 'switch_stoch_volatility.bug';

model = biips_model(model_file, data, 'sample_data', true);
data = model.data;
```

SMC samples

Matbiips

```
n_part = 5000;
variables = {'x'};

out_smc = biips_smc_samples(model, variables, n_part);
diag_smc = biips_diagnosis(out_smc);
```

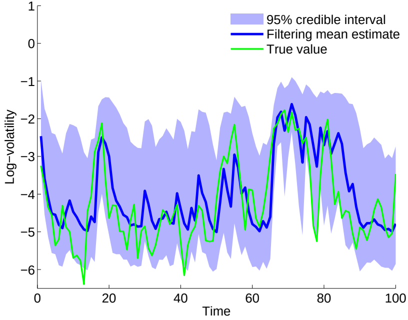




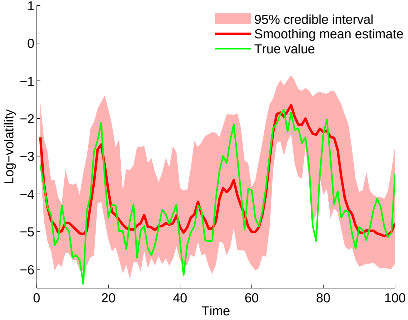
Summary statistics

Matbiips

```
summ = biips_summary(out_smc, 'probs', [.025, .975]);
x_f_mean = summ.x.f.mean; x_f_quant = summ.x.f.quant;
x_s_mean = summ.x.s.mean; x_s_quant = summ.x.s.quant;
```



(c) Filtering

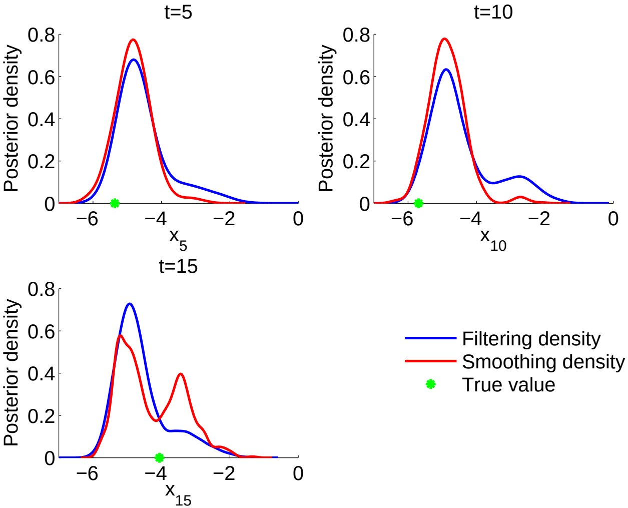


(d) Smoothing

Kernel density estimates

Matbiips

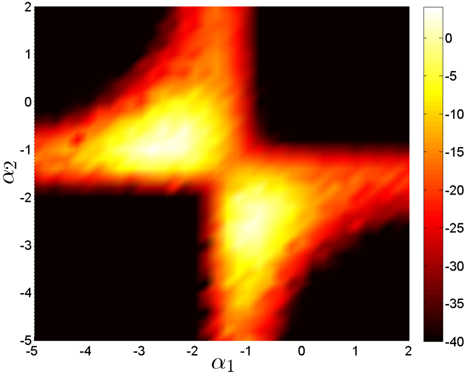
```
kde_smc = biips_density(out_smc);
```



Sensitivity analysis

Matbiips

```
n_part = 50;
param_names = {'alpha'};
[A, B] = meshgrid(-5:.2:2, -5:.2:2);
param_values = {[A(:), B(:)]'};

out_sens = biips_smc_sensitivity(model, param_names, param_values,
    n_part);
```



Summary

Context

Graphical models and BUGS language

SMC

Matbiips

Particle MCMC

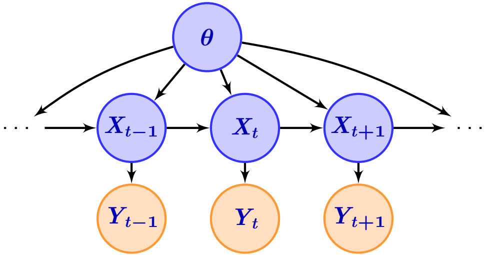
Particle MCMC

Recent algorithms that use SMC algorithms within a MCMC algorithm

- Particle Independant Metropolis-Hastings (PIMH)
- Particle Marginal Metropolis-Hastings (PMMH)

[Andrieu et al., 2010]

Static parameter estimation



Due to the successive resamplings, SMC estimations of $p(\theta|y_{1:n})$ might be poor.

The PMMH splits the variables in the graphical model into two sets:

- a set of variables X that will be sampled using a SMC algorithm
- a set $\theta = (\theta_1, \dots, \theta_p)$ sampled with a MH proposal

PMMH

Standard PMMH algorithm

Set $\widehat{Z}(0) = 0$ and initialize $\theta(0)$

For $k = 1, \dots, n_{\text{iter}}$,

- Sample $\theta^\star \sim \nu$
- Run a SMC to approximate $p(x_{1:n}|y_{1:n}, \theta^\star)$ with output $(X_{1:n}^{\star(i)}, W_n^{\star(i)})_{i=1,\dots,N}$ and $\widehat{Z}^\star$
- With probability

$$\min\left(1, \frac{\nu(\theta^\star|\theta(k-1))p(\theta^\star)\widehat{Z}^\star}{\nu(\theta(k-1)|\theta^\star)p(\theta(k-1))\widehat{Z}(k-1)}\right)$$

  set $X_{1:n}(k) = X_{1:n}^{\star(\ell)}$, $\theta(k) = \theta^\star$ and $\widehat{Z}(k-1) = \widehat{Z}^\star$, where $\ell \sim \text{Discrete}(W_n^{\star(1)}, \dots, W_n^{\star(N)})$
- otherwise, keep previous iteration values

Outputs

- MCMC samples $(X_{1:n}(k), \theta(k))_{k=1,\dots,n_{\text{iter}}}$

Static parameter estimation in the SSV model

We consider the following prior on the parameters $\boldsymbol{\alpha}$, $\boldsymbol{\pi}$, $\boldsymbol{\phi}$ and $\boldsymbol{\tau}$:

$$\begin{aligned}
\alpha_1 &= \gamma_1 \\
\alpha_2 &= \gamma_1 + \gamma_2 \\
\gamma_1 &\sim \mathcal{N}(0, 100) \\
\gamma_2 &\sim \mathcal{TN}_{(0,+\infty)}(0, 100)
\end{aligned}$$

$$\begin{aligned}
\frac{1}{\sigma^2} &\sim \mathbf{Gamma}(2.001, 1) \\
\phi &\sim \mathcal{TN}_{(-1,1)}(0, 100) \\
\pi_{11} &\sim \mathbf{Beta}(10, .5) \\
\pi_{22} &\sim \mathbf{Beta}(10, .5)
\end{aligned}$$

[Carvalho and Lopes, 2007]

SSV model with unknown parameters in BUGS language

switch_stoch_volatility_param.bug

```
model
{
  gamma[1] ~ dnorm(0, 1/100)
  gamma[2] ~ dnorm(0, 1/100) T(0,)
  alpha[1] <- gamma[1]
  alpha[2] <- gamma[1] + gamma[2]
  phi ~ dnorm(0, 1/100) T(-1,1)
  tau ~ dgamma(2.001, 1)
  sigma <- 1/sqrt(tau)
  pi[1,1] ~ dbeta(10, .5)
  pi[1,2] <- 1.00 - pi[1,1]
  pi[2,2] ~ dbeta(10, .5)
  pi[2,1] <- 1.00 - pi[2,2]
  ...
}
```

Matbiips

```
model_file = 'switch_stoch_volatility_param.bug';
model = biips_model(model_file, data, 'sample_data', sample_data);
data = model.data;
```

PMMH samples

Run a PMMH sampler to approximate
$p(\alpha_1, \alpha_2, \sigma, \pi_{11}, \pi_{22}, \phi, X_{1,T}, C_{1:T} | Y_{1:T})$.

Matbiips

```
n_burn = 2000;
n_iter = 40000;
thin = 10;
n_part = 50;
param_names = {'gamma[1,1]','gamma[2,1]', 'phi', 'tau', 'pi[1,1]',
    'pi[2,2]'};
latent_names = {'x','alpha[1,1]','alpha[2,1]', 'sigma'};

inits = {-1, 1,.5,5,.8,.8};
obj_pmmh = biips_pmmh_init(model, param_names, 'inits', inits, '
    latent_names', latent_names);

obj_pmmh = biips_pmmh_update(obj_pmmh, n_burn, n_part);
[obj_pmmh, out_pmmh, log_marg_like_pen, log_marg_like] =...
    biips_pmmh_samples(obj_pmmh, n_iter, n_part,'thin', thin);
```

Posterior samples

α_1

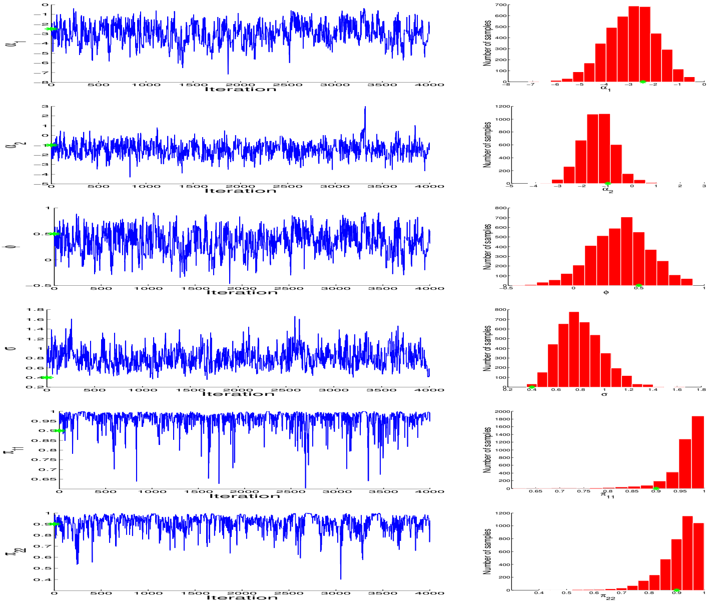
α_2

ϕ

σ

π_{11}

π_{22}



Other features of *Biips*

- Backward smoothing algorithm
- Particle Independent Metropolis-Hastings algorithm
- Automatic choice of the proposal distribution including **Optimal/Conditional samplers**: Gaussian-Gaussian, Beta-Bernoulli, Finite discrete
- Easy BUGS language extensions with user-defined Matlab/R functions

Related software

using MCMC

- WinBUGS, OpenBUGS [Lunn et al., 2000, Lunn et al., 2012], JAGS [Plummer, 2003]
- Stan [Stan Development Team, 2013]

using SMC

- SMCTC [Johansen, 2009]
- LibBi [Murray, 2013]

using both

- Venture [Mansinghka et al., 2014], Anglican [Wood et al., 2014]

Conclusion

- BUGS language compatible
- Extensibility: user-defined functions/samplers
- Black-box SMC inference engine
- Interfaces with popular software: Matlab/Octave, R
- Post-processing

Bibliography I

Andrieu, C., Doucet, A., and Holenstein, R. (2010).
Particle markov chain monte carlo methods.
Journal of the Royal Statistical Society B, 72:269–342.

Carvalho, C. M. and Lopes, H. F. (2007).
Simulation-based sequential analysis of Markov switching stochastic volatility models.
Computational Statistics & Data Analysis, 51(9):4526–4542.

Del Moral, P. (2004).
Feynman-Kac Formulae. Genealogical and Interacting Particle Systems with Application.
Springer.

Doucet, A., de Freitas, N., and Gordon, N., editors (2001).
Sequential Monte Carlo Methods in Practice.
Springer-Verlag.

Doucet, A. and Johansen, A. (2010).
A tutorial on particle filtering and smoothing: Fifteen years later.
In Crisan, D. and Rozovsky, B., editors, *Oxford Handbook of Nonlinear Filtering*. Oxford University Press.

Johansen, A. (2009).
SMCTC: Sequential monte carlo in C++.
Journal of Statistical Software, 30:1–41.

Bibliography II

Lunn, D., Jackson, C., Best, N., Thomas, A., and Spiegelhalter, D. (2012).
*The **BUGS** Book: A Practical Introduction to Bayesian Analysis*.
CRC Press/ Chapman and Hall.

Lunn, D., Thomas, A., Best, N., and Spiegelhalter, D. (2000).
WinBUGS - a Bayesian modelling framework: Concepts, structure and extensibility.
Statistics and Computing, 10:325–337.

Mansinghka, V. K., Selsam, D., and Perov, Y. N. (2014).
Venture: A higher-order probabilistic programming platform with programmable inference.
Technical report, arXiv:1404.0099.

Murray, L. (2013).
Bayesian state-space modelling on high-performance hardware using **LibBi**.
Technical report, CSIRO.
Arxiv:1306.3277.

Plummer, M. (2003).
JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling.
In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing*.

Plummer, M. (2012).
***JAGS** Version 3.3.0 user manual*.

Bibliography III

Stan Development Team (2013).
Stan: A C++ library for probability and sampling, version 2.1.

Wood, F., van de Meent, J. W., and Mansinghka, V. (2014).
A new approach to probabilistic programming inference.
In *Proceedings of the 17th International conference on Artificial Intelligence and Statistics*.

THANK YOU

http://alea.bordeaux.inria.fr/biips