Biips software: Bayesian inference with interacting particle systems
BAYES 2015

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

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

Basel, May 2015
Outline

Context

Graphical models and BUGS language

SMC

Biips software

Particle MCMC
Summary

Context

Graphical models and BUGS language

SMC

Biips software

Particle MCMC
Context

\textbf{Biips} = Bayesian inference with interacting particle systems

Bayesian inference

- Sample from a posterior distribution \( p(X|Y) = \frac{p(X,Y)}{p(Y)} = \frac{p(X,Y)}{Z} \)
- 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**

\textbf{Biips} = \textbf{B}ayesian \textbf{i}nference with \textbf{i}nteracting \textbf{p}article \textbf{s}ystems

**Bayesian inference**

- Sample from a posterior distribution \( p(X|Y) = \frac{p(X,Y)}{p(Y)} = \frac{p(X,Y)}{Z} \)
- 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** = Bayesian inference with interacting particle systems

Features

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

Context

Graphical models and BUGS language

SMC

Biips software

Particle MCMC
Graphical models

The graph displays a factorization of the joint distribution:

\[ p(x_{1:3}, y_{1:2}) \]

Directed acyclic graph
Graphical models

The graph displays a factorization of the joint distribution:

\[
p(x_{1:3}, y_{1:2}) = p(x_1) \ p(x_2|x_1) \ p(y_1|x_2) \ p(x_3|x_1, x_2) \ p(y_2|x_2, x_3)
\]
The graph displays a factorization of the joint distribution:

\[
p(x_{1:3}, y_{1:2}) = p(x_1) \ p(x_2 | x_1) \ p(y_1 | x_2) \\
\qquad \times \ p(x_3 | x_1, x_2) \ p(y_2 | x_2, x_3)
\]
Graphical models

The graph displays a factorization of the joint distribution:

\[
p(x_1:3, y_1:2) = p(x_1) \ p(x_2|x_1) \ p(y_1|x_2) \\
p(x_3|x_1, x_2) \ p(y_2|x_2, x_3)
\]
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

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

Linear regression:

```plaintext
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 language

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

Linear regression:

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

\[
\begin{align*}
\mu & \sim \text{dnorm}(0, 1E-6) \\
\beta & \sim \text{dnorm}(0, 1E-6) \\
\tau & \sim \text{dgamma}(0.01, 0.01)
\end{align*}
\]
BUGS language

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

Linear regression:

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

Goal:

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

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

Linear regression:

```r
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

Context

Graphical models and BUGS language

SMC

Biips software

Particle MCMC
Ordering of the graph

Topological sort (with priority to measurement nodes):

Rearrangement of the directed acyclic graph:

The statistical model decomposes as

\[ p(x_1', x_2', y_1, y_2) = p(x_1') p(y_1' | x_1') p(x_2' | x_1', y_1') p(y_2' | x_2') \]
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

Topological sort (with priority to measurement nodes):

\[(X_1, Y_1, Y_3, X_3, X_2, Y_4, Y_2)\]

\[(X'_1, Y'_1, X'_2, Y'_2)\]
Ordering of the graph

Topological sort (with priority to measurement nodes):

\[
(X_1, Y_1, X_3, X_2, Y_4, Y_2) = (X'_1, Y'_1, X'_2, Y'_2)
\]

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

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

where $\text{pa}(x)$ 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 \( t = 1, \ldots, n \)

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

\[ = \pi_{t-1}(x_{1:t-1}|y_{1:t-1}) \frac{p(x_t|pa(x_t))p(y_t|pa(y_t))}{p(y_t|y_{1:t-1})} \]

Two stochastic mechanisms:

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

Standard SMC Algorithm

For $t = 1, \ldots, n$

- For $i = 1, \ldots, N$
  - Sample: $X_{t,t}^{(i)} \sim q_{t}$ and let $X_{t,1:t}^{(i)} = (\tilde{X}_{t-1,1:t-1}^{(i)}, X_{t,t}^{(i)})$
  - Weight: $w_{t}^{(i)} = \frac{\pi(y_{t}|pa(y_{t}))\pi(x_{t,t}^{(i)}|pa(x_{t,t}^{(i)}))}{q_{t}(x_{t,t}^{(i)})}$
  - Normalize: $W_{t}^{(i)} = \frac{w_{t}^{(i)}}{\sum_{j=1}^{N} w_{t}^{(j)}}$

- Resample: $\{X_{t,1:t}^{(i)}, W_{t}^{(i)}\}_{i=1,\ldots,N} \rightarrow \{\tilde{X}_{t,1:t}^{(i)}, \frac{1}{N}\}_{i=1,\ldots,N}$

Outputs

- Weighted particles $(W_{t}^{(i)}, X_{t,1:t}^{(i)})_{i=1,\ldots,N}$ for $t = 1, \ldots, n$
- Estimate of the marginal likelihood $\hat{Z} = \prod_{t=1}^{n} \left( \frac{1}{N} \sum_{i=1}^{N} w_{t}^{(i)} \right)$
SMC algorithm

Marginal distributions

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

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

Smoothing: \( \pi_1(x_1|y_{1:n}), \pi_2(x_2|y_{1:n}), \ldots, \pi_n(x_n|y_{1:n}) \)
Example

Hidden Markov model/State space model

\[ x_0 \sim \mathcal{N}(0, 1) \]
\[ x_t|x_{t-1} \sim \mathcal{N}(x_{t-1}, 1), \quad t = 1, \ldots, 10 \]
\[ y_t|x_t \sim \mathcal{N}(x_t, 1), \quad t = 1, \ldots, 10 \]

▶ Linear Gaussian model
▶ Goal: estimate \( p(x_{1:t}|y_{1:t}) \)
▶ Analytic solution given by Kalman equations
Example: hidden Markov/state space model
Example: hidden Markov/state space model

\( t=1, N=50 \)

Particles mutation

Particle weights

\( \text{ESS}_1 = 22.27 \)
Example: hidden Markov/state space model

$t=1, N=50$

Particles resampling

Particle weights

ESS$_1=22.27$
Example: hidden Markov/state space model
Example: hidden Markov/state space model

Particles resampling

ESS\textsuperscript{2} = 28.78
Example: hidden Markov/state space model

$t=3, N=50$

Particles mutation

Particle weights

$ESS_3 = 25.50$
Example: hidden Markov/state space model
Example: hidden Markov/state space model

$t=4, N=50$
Particles mutation
Particle weights
$\text{ESS}_4=3.39$
Example: hidden Markov/state space model
Example: hidden Markov/state space model

\[ t=5, N=50 \]

Particles mutation

Particle weights

\[ \text{ESS}_5 = 30.92 \]
Example: hidden Markov/state space model

Particles resampling

ESS₅ = 30.92
Example: hidden Markov/state space model

\[ t=6, N=50 \]

Particles mutation
Particle weights
\[ \text{ESS}_6 = 33.01 \]
Example: hidden Markov/state space model
Example: hidden Markov/state space model

\[ t=7, \ N=50 \]

Parties mutation

Particle weights

\[ \text{ESS}_7 = 31.77 \]
Example: hidden Markov/state space model

Particles resampling

ESS$_7$ = 31.77
Example: hidden Markov/state space model

$\text{ESS}_8 = 33.86$

$t=8, N=50$

Particles mutation

Particle weights
Example: hidden Markov/state space model

$t=8, N=50$

Particles resampling

Particle weights

$\text{ESS}_{8} = 33.86$
Example: hidden Markov/state space model

\( t=9, N=50 \)

Particles mutation

Particle weights

ESS\(_9\) = 35.66
Example: hidden Markov/state space model

\[ t=9, \quad N=50 \]

Particles resampling

Particle weights

\[ \text{ESS}_9 = 35.66 \]
Example: hidden Markov/state space model
Limitations and diagnosis of SMC algorithms

For a given $t \leq n$, for each unique value $X'_{n,t}(k)$, $k = 1, \ldots, K_{n,t}$, let $W'_{n,t}(k) = \sum_{i | X_t^{(i)} = X_t'_{n,t}(k)} W_{n}^{(i)}$ 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):

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

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

A. Todeschini
Summary

Context

Graphical models and BUGS language

SMC

Biips software

Particle MCMC
Technical implementation

- Interfaces: Matlab/Octave, R
- Multi-platform: Windows, Linux, Mac OSX
- Free and open source (GPL)
Example: Stochastic kinetic Lotka-Volterra model

- Evolution of two species $X_1(t)$ (prey) and $X_2(t)$ (predator) at time $t$
- Continuous-time Markov jump process described by three reaction equations:

$$
X_1 \xrightarrow{c_1} 2X_1 \quad \text{prey reproduction},
X_1 + X_2 \xrightarrow{c_2} 2X_2 \quad \text{predator reproduction},
X_2 \xrightarrow{c_3} \emptyset \quad \text{predator death}
$$

where $c_1 = 0.5$, $c_2 = 0.0025$ and $c_3 = 0.3$.

$$
\begin{align*}
\Pr(X_1(t+dt) = x_1(t) + 1, X_2(t+dt) = x_2(t)|x_1(t), x_2(t)) &= c_1 x_1(t) dt + o(dt) \\
\Pr(X_1(t+dt) = x_1(t) - 1, X_2(t+dt) = x_2(t) + 1|x_1(t), x_2(t)) &= c_2 x_1(t) x_2(t) dt + o(dt) \\
\Pr(X_1(t+dt) = x_1(t), X_2(t+dt) = x_2(t) - 1|x_1(t), x_2(t)) &= c_3 x_2(t) dt + o(dt)
\end{align*}
$$

[Boys et al., 2008]
Gillespie algorithm

R function to forward simulate from the LV model with Gillespie algorithm

```r
lotka_volterra_gillespie <- function(x, c1, c2, c3, dt) {
  z <- matrix(c(1, -1, 0, 0, 1, -1), nrow = 2, byrow = TRUE)
  t <- 0
  while (TRUE) {
    rate <- c(c1*x[1], c2*x[1]*x[2], c3*x[2])
    sum_rate <- sum(rate);
    # Sample the next event from an exponential distribution
    t <- t - log(runif(1))/sum_rate
    if (t>dt)
      break
    # Sample the type of event
    ind <- which((sum_rate*runif(1)) <= cumsum(rate))[1]
    x <- x + z[,ind]
  }
  return(x)
}
```

[Gillespie, 1977, Golightly and Gillespie, 2013]
Add a custom sampler to the BUGS language

Rbiips

\[
\text{biips\_add\_distribution}(\text{name} = \text{'LV'}, \\
\text{n\_param} = 5, \\
\text{fun\_dim} = \text{lotka\_volterra\_dim}, \\
\text{fun\_sample} = \text{lotka\_volterra\_gillespie})
\]
Example: Stochastic kinetic Lotka-Volterra model

- We observe at some time $t = 1, 2, \ldots, t_{\text{max}}$ the number of preys with some additive noise

\[ Y(t) = X_1(t) + \epsilon(t), \quad \epsilon(t) \sim \mathcal{N}(0, \sigma^2) \]

- Objective: approximate $\Pr(X_1(t), X_2(t) | Y(1), \ldots, Y(t_{\text{max}}))$ at $t = 1, \ldots, t_{\text{max}}$. 
Example: Stochastic kinetic Lotka-Volterra model

```
model
{
  x[,1] ~ LV(x_init, c[1], c[2], c[3], 1)
  y[1] ~ dnorm(x[1,1], 1/sigma^2)
  for (t in 2:t_max)
  {
    x[,t] ~ LV(x[,t-1], c[1], c[2], c[3], 1)
    y[t] ~ dnorm(x[1,t], 1/sigma^2)
  }
}
```
Model compilation

Rbiips

data <- list(t_max = 40, c = c(.5, .0025, .3),
            x_init = c(100, 100), sigma = 10)
model <- biips_model(model_file = 'stoch_kinetic_gill.bug',
                     data = data,
                     sample_data = TRUE)
data <- model$data()
SMC samples

```r
out_smc <- biips_smc_samples(model, variable_names = 'x',
                   n_part = 10000, type = 'fs')

diag_smc <- biips_diagnosis(out_smc)
 summ_smc <- biips_summary(out_smc, probs=c(.025, .975))
 x_s_mean <- summ_smc$x$s$mean
 x_s_quant <- summ_smc$x$s$quant
```

(a) Estimates

(b) Smoothing effective sample size
Kernel density estimates

Rbiips

\[
\text{kernel density estimates} \\
\text{Rbiips} \\
kde_{\text{smc}} \leftarrow \text{biips\_density}(\text{out\_smc})
\]
Probability mass estimates

Rbiips

tab_smc <- biips_table(out_smc)
Summary

Context

Graphical models and BUGS language

SMC

Biips software

Particle MCMC
Particle MCMC

Recent algorithms that use SMC algorithms within a MCMC algorithm

- Particle Independant Metropolis-Hastings (PIMH)
- Particle Marginal Metropolis-Hastings (PMMH)
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 $\theta = (\theta_1, \ldots, \theta_p)$ sampled with a MH proposal
Standard PMMH algorithm

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

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

- Sample $\theta^* \sim \nu(\cdot|\theta^{(k-1)})$
- Run a SMC to approximate $p(x_{1:n}|y_{1:n}, \theta^*)$ with output $(X^*_1, \ldots, X^*_n)$ and $\hat{Z}^* \approx p(y_{1:n}|\theta^*)$
- With probability

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

set $X_{1:n}(k) = X^*_1, \theta(k) = \theta^*$ and $\hat{Z}(k-1) = \hat{Z}^*$, where

$$\ell \sim \text{Discrete}(W^*_1, \ldots, W^*_n)$$

- otherwise, keep previous iteration values

Outputs

- MCMC samples $(X_{1:n}(k), \theta(k))_{k=1,\ldots,n_{\text{iter}}}$
Example: Stochastic kinetic Lotka-Volterra model

```stoch_kinetic_gill.bug

model
{
  logc[1] ~ dunif(-7,2)
  logc[2] ~ dunif(-7,2)
  logc[3] ~ dunif(-7,2)
  c[1] <- exp(logc[1])
  c[2] <- exp(logc[2])
  c[3] <- exp(logc[3])
  ...
}
```

A. Todeschini
Run a PMMH algorithm

```r
# create a pmmh object
obj_pmmh = biips_pmmh_init(model,
param_names = c('logc[1]',
                 'logc[2]',
                 'logc[3]'),
inits = list(-1, -5, -1),
latent_names = 'x')

# adaptation and burn-in iterations
biips_pmmh_update(obj_pmmh, n_iter = 2000, n_part = 100)

# samples
out_pmmh = biips_pmmh_samples(obj_pmmh, n_iter = 20000,
                               n_part = 100, thin = 10)

summ_pmmh = biips_summary(out_pmmh, probs = c(.025, .975))
kde_pmmh = biips_density(out_pmmh)
```
Posterior samples

- **log(c_1)**
- **log(c_2)**
- **log(c_3)**

---

- **Number of samples**

  - **log(c_1)**
  - **log(c_2)**
  - **log(c_3)**

- **Posterior density**

  - **log(c_1)**
  - **log(c_2)**
  - **log(c_3)**
Conclusion

- BUGS language compatible
- Extensibility: custom functions/samplers
- Black-box SMC inference engine
- Interfaces with popular software: Matlab/Octave, R
- Post-processing tools
- And more: backward smoothing algorithm, particle independent Metropolis-Hastings algorithm, sensitivity analysis, some optimal/conditional samplers (Gaussian-Gaussian, beta-Bernoulli, finite discrete)
What is Biips?

Biips is a general software for *Bayesian inference with interacting particle systems*, a.k.a. sequential Monte Carlo (SMC) methods. It aims at popularizing the use of these methods to non-statistician researchers and students, thanks to its automated “black box” inference engine.

It borrows from the @BUGS/@JAGS software, widely used in Bayesian statistics, the statistical modeling with graphical models and the language associated with their descriptions.

Features

- BUGS language compatible
- SMC techniques for filtering and smoothing
- Static parameter estimation using particle MCMC
- Core developed in C++
- R, Matlab/Octave interfaces
- Easy language extensions with custom R and Matlab functions
- Multi-platform: Linux, Windows, Mac
- Free and open source (GPL)

[Todeschini et al., 2014]


THANK YOU

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