Practicalities in Implementing Adaptive Particle MCMC

Julien Cornebise

Includes tricks learned while working with
A. Doucet (Oxford)
G. Peters (UNSW and UC London)
J. Olsson (Lund University)
E. Moulines (Telecom ParisTech)
O. Cappé (Telecom ParisTech)

ISBA 2012
Kyoto, Japan
June 28th 2012
Early morning: not an advanced research talk:
  - Not (mainly) aimed at compstat experts
  - but at those who need to use them!

Acting word here: **practicalities**
  - more of a tutorial
  - focus on implementation
  - of *existing* cutting-edge algorithms
  - using *known* tricks of SMC/MCMC folklore
  - ... but that are **very easy to overlook**!
Outline

1. Particle MCMC: What & How
   - State-space models
   - PMCMC in a Nutshell
   - Particle Filtering

2. Bottleneck: Resampling step

3. Adaptative MCMC: really worth it, with trick

4. Mirror Mathematical Structure: Object-Oriented
Particle MCMC: What & How
- State-space models
- PMCMC in a Nutshell
- Particle Filtering

Bottleneck: Resampling step

Adaptative MCMC: really worth it, with trick

Mirror Mathematical Structure: Object-Oriented
Hidden Markov Models / State Space Model

\[
\begin{align*}
X_0 &\sim p(x_0|\theta) \quad \text{initial distribution} \\
X_{k+1} &= f_{\theta}(X_k, V_k) \sim p(X_{k+1}|X_k, \theta) \quad \text{prior kernel/dynamic} \\
Y_k &= h_{\theta}(X_k, U_k) \sim p(y_k|X_k, \theta) \quad \text{local likelihood/observation}
\end{align*}
\]

Fundamental model in time series. Spread to robotics, bioinformatics, chemistry, forest science . . .
Hidden Markov Models / State Space Model

\[
\begin{aligned}
    &X_0 \sim p(x_0|\theta) & \text{initial distribution} \\
    &X_{k+1} = f_\theta(X_k, V_k) \sim p(X_{k+1}|X_k, \theta) & \text{prior kernel/dynamic} \\
    &Y_k = h_\theta(X_k, U_k) \sim p(y_k|X_k, \theta) & \text{local likelihood/observation}
\end{aligned}
\]

Fundamental model in time series. Spread to robotics, bioinformatics, chemistry, forest science ...
State-space model as part of hierarchical model, static parameters of interest:

\[ y_{ij} = X_{it_{ij}} (1 + \varepsilon_{ij}), \quad dX_{it} = B_i C_i e^{-C_i t} X_{it} dt + \gamma X_{it} dW_{it} \]
Aim: sample from intricate distribution $\pi(\theta)$.

At initialization $k = 0$,

1. set $\theta(0)$ arbitrarily

At iteration $k \geq 1$

1. sample $\theta_*$ from $q(\cdot|\theta(k - 1))$
2. with probability

$$
\min \left\{ 1, \frac{\pi(\theta_*) q(\theta(k - 1)|\theta_*)}{\pi(\theta(k - 1)) q(\theta_*|\theta(k - 1))} \right\}
$$

accept: set $\theta(k) = \theta_*$, else reject: set $\theta(k) = \theta(k - 1)$.

In Gompertz example – note the absence of trajectory $x_{1:T_i}$, integrated out:

$$
\theta = (A, B, C, \omega_A, \omega_B, \omega_C, \{A^i, B^i, C^i\}_{i=1}^n, \sigma, \gamma)
$$

$$
\pi(\theta) \propto p(\{y_{1:T_i}\}_{i=1}^n|\theta)p(\theta)
$$
Problem: for static parameters of state space models, $p(y_{1:T} | \theta)$ not available – only $p(y_{1:T} | x_{1:T}, \theta)$

Key idea: combine Particle Filter (aka Sequential Monte Carlo) within MCMC, Andrieu, Doucet, Holenstein (2010).

Particle filter
- samples approximately from $p(X_{1:T} | y_{1:T}, \theta)$
- provides approximation of $p(y_{1:T} | \theta)$ needed in MH ratio.

Beauty of PMCMC

Note: no need for accurate approximation, beauty of PMCMC, crude particle approximation suffices for exact MH target.
At iteration $k \geq 1$

1. sample $\theta_*$ from $q(\cdot | \theta(k-1))$

2. use SMC to
   1. Sample from $p^N(X_{1:T}|y_{1:T}, \theta_*)$
   2. obtain $Z_* = \hat{p}(y_{1:T}|\theta_*)$

3. with probability

$$\min \left\{ 1, \frac{Z_*}{Z(k-1)} \frac{q(\theta(k-1)|\theta_*)}{q(\theta_*|\theta(k-1))} \right\}$$

accept: set $\theta(k) = \theta_*$ and $Z(k) = Z_*$, else reject: set $\theta(k) = \theta(k-1)$ and $Z(k) = Z(k-1)$. 
Mutation
Weighting
Selection

State space
Time step

Skip Courtesy of F. Campillo, INRIA
Mutation
Weighting
Selection
→ Mutation
   Weighting
   Selection

State space

Time step

Courtesy of F. Campillo, INRIA
Courtesy of F. Campillo, INRIA
Mutation
Weighting
Selection

State space

Time step

Courtesy of F. Campillo, INRIA
 Courtesy of F. Campillo, INRIA
Courtesy of F. Campillo, INRIA
To beginning

Mutation
Weighting
→ Selection

 Courtesy of F. Campillo, INRIA

State space
Time step
1. Particle MCMC: What & How
   - State-space models
   - PMCMC in a Nutshell
   - Particle Filtering

2. Bottleneck: Resampling step

3. Adaptative MCMC: really worth it, with trick

4. Mirror Mathematical Structure: Object-Oriented
Resample $N$ new particles amongst the $N$ existing ones, with probability $\propto w_i$

- Multinomial, Stratified, Systematic, Residual
- Resampling vs Branching
- Impact on variance, no CLT for Systematic

All ensure that number of sons proportional to weight:

$$\mathbb{E} \left[ \sum_{k=1}^{N} 1 \{ I(k) = i \} \right] = N \frac{w_i}{\sum w_j}$$

Simplest choice implemented: multinomial resampling, possibly on residual
function I = randind_naive(p, N)

cdf = cumsum(p);
cdf(end) = 1;

I = zeros(N, 1);
U = rand(N, 1);

for k = 1:N
    index = 1;
    while (U(k) > cdf(index))
        index = index + 1;
    end
    I(k) = index;
end
function I = randoind_naive(p, N)

cdf = cumsum(p);
cdf(end) = 1;

I = zeros(N, 1);
U = rand(N, 1);

for k = 1:N
    index = 1;
    while (U(k) > cdf(index))
        index = index + 1;
    end
    I(k) = index;
end

function I = randoind_sort(p, N)

cdf = cumsum(p);
cdf(end) = 1;

I = zeros(N, 1);
U = rand(N, 1);
U = sort(U);
index = 1;
for k = 1:N
    while (U(k) > cdf(index))
        index = index + 1;
    end
    I(k) = index;
end
function I = randoind_naive(p, N)

cdf = cumsum(p);
cdf(end) = 1;

I = zeros(N, 1);
U = rand(N, 1);

for k = 1:N
    index = 1;
    while (U(k) > cdf(index))
        index = index + 1;
    end
    I(k) = index;
end

function I = randoind_sort(p, N)

cdf = cumsum(p);
cdf(end) = 1;

I = zeros(N, 1);
U = rand(N, 1);

U = sort(U);
index = 1;

for k = 1:N
    while (U(k) > cdf(index))
        index = index + 1;
    end
    I(k) = index;
end

Algorithmic complexity

$O(N^2)$ vs $O(N \log N)$

Sorted indices, but invariance of MC estimates by permutation of particles:

$$\mathbb{E}_N^N [ f(x_t) | y_{1:t}] := \sum_{i=1}^N f(x_t^i)$$

Even better (stay for discussion!):

$U = U/N + (0:(N-1))'/N$;

Cappé, Moulines, Ryden (2005, Ch. 7)
1. Particle MCMC: What & How
   - State-space models
   - PMCMC in a Nutshell
   - Particle Filtering

2. Bottleneck: Resampling step

3. Adaptative MCMC: really worth it, with trick

4. Mirror Mathematical Structure: Object-Oriented
Adaptive MCMC:
- Especially worthy here, costly proposal
- Implementation reusable with OOP (cf later)

Simplest scheme:

Adaptive Random Walk
- Learn the proposal covariance matrix (Haario et al. 2001)
- Mixture with fixed component to prevent over-learning (Rosenthal and Roberts 2006, Andrieu and Moulines 2008)
Basic trick, do not recompute estimate each time:

\[
m(k) = \frac{1}{k} \sum_{i=1}^{k} \theta(k)
\]

\[
\Sigma(k) = \frac{1}{k} \sum_{i=1}^{k} \left[ \theta(k) - m(k) \right] \left[ \theta(k) - m(k) \right]^T
\]

Instead, **recursive** estimate, only **update** it:

\[
m(k) = m(k-1) + \frac{1}{k} \left[ \theta(k) - m(k-1) \right]
\]

\[
s(k) = s(k-1) + \frac{1}{k} \left[ \theta(k) \cdot \theta(k)^T - s(k-1) \right]
\]

\[
\Sigma(k) = s(k) - m(k) \cdot m(k)^T
\]

Woodbury equality for Cholesky update
• Basic trick, do not recompute estimate each time:

\[ m(k) = \frac{1}{k} \sum_{i=1}^{k} \theta(k) \]

\[ \Sigma(k) = \frac{1}{k} \sum_{i=1}^{k} \left[ \theta(k) - m(k) \right] \left[ \theta(k) - m(k) \right]^T \]

Instead, recursive estimate, only update it:

\[ m(k) = m(k-1) + \frac{1}{k} \left[ \theta(k) - m(k-1) \right] \]

\[ s(k) = s(k-1) + \frac{1}{k} \left[ \theta(k) \cdot \theta(k)^T - s(k-1) \right] \]

\[ \Sigma(k) = s(k) - m(k) \cdot m(k)^T \]

• Woodbury equality for Cholesky update

**Lesser known trick: wait before use**

In theory: can adapt after \( d \) steps – as soon as full rank \( \Sigma(k) \).
In practice: **wait a bit**
Without adaptation
With adaptation

Autocorrelation
Lag

0 10 20 30 40
0.4
0.5
0.6
0.7
0.8
0.9
1

Without adaptation
With adaptation

Autocorrelation
Lag

0 10 20 30 40
0.4
0.5
0.6
0.7
0.8
0.9
1
1 Particle MCMC: What & How
   - State-space models
   - PMCMC in a Nutshell
   - Particle Filtering

2 Bottleneck: Resampling step

3 Adaptative MCMC: really worth it, with trick

4 Mirror Mathematical Structure: Object-Oriented
Why Investing in Quality Software Development?

Quite geeky: *computational* statistics.

**Originally: tables**

Fisher produced **tables**:

1. User picks, opens, uses
2. to do the actual computation

Articles explain how and what those are made for, and bring long-term legacy.
Why Investing in Quality Software Development?

Quite geeky: *computational* statistics.

**Originally: tables**

Fisher produced **tables**:
1. User picks, opens, uses
2. to do the actual computation

**21st century Bayesian: software**

We produce **software**:
1. User picks, installs, uses
2. to do the actual computation

Articles explain how and what those are made for, and bring long-term legacy.

UK’s ESPRC requires **pathway to impact** for grants. Polished usable software package is such a pathway.

Publish (software)-or-perish?  
...and die trying?
Comfort brought by Object-Oriented

- Mirrors mathematical structure
- Reusability
- Modularity: Play Lego with algorithms, not with functions
- Unit tests (test-driven development): debug time plumets, robustify

Not only me:
- ABC-SysBio – Imperial College, Michael Stumpf’s group
- PyMCMC – University of Queensland, Kerrie Mengersen’s group
- JAGS – Martyn Plummer

Good books, easy reads:

- M. Fowler (1999)
- R. Martin (2008)
- J. Kerievsky (2004)
Playing Lego with Algorithms, in Code as in Math

J. Cornebise

Practicalities in implementing Adaptive Particle MCMC 18/20
Practicalities in implementing Adaptive Particle MCMC

SMCObserver
- BeforeRun()
- EndOfTimestep()

C: SMCScorer
C: SMCMarginalLikelihood
C: SMCTimer
C: DynamicSystem

C: Filter
- AttachObserver()
- NotifyBeforeRun()
- NotifyEndOfTimestep()
Playing Lego with Algorithms, in Code as in Math

Practicalities in implementing Adaptive Particle MCMC 18/20
Playing Lego with Algorithms, in Code as in Math

J. Cornebise

Practicalities in implementing Adaptive Particle MCMC 18/20
Imagine a world where:

- new code does not introduce new bug in past code,
- you can thus refine without breaking;
- someone tells you when you can stop coding;
- examples are shipped with the code, and runnable,
- ... for users and for your future self!


Test-Driven Development

xUnit tools: jUnit, mUnit, RUnit...

Run every few seconds

Serve as examples and documentation

Catches bugs when code is still fresh in your mind:

no more pain

Bonus: validate software for use in industry.
Imagine a world where:
- new code does not introduce new bug in past code,
- you can thus refine without breaking;
- someone tells you when you can stop coding;
- examples are shipped with the code, and runnable,
- ... for users and for your future self!


Test-Driven Development

xUnit tools: jUnit, mUnit, RUnit...
- Run every few seconds
- Run in few seconds
- Serve as examples and documentation
- Catches bugs when code is still fresh in your mind: no more pain
- Bonus: validate software for use in industry.
Open questions:
- Adapt number of particles?
- Type of resampling (see discussion)
- Smarter adaptation

Take-home message
- Resampling: lazy is OK, but be smart.
- Adaptation: worth using and waiting.
- Object: invest in comfort for impact!

Thank you for your attention!
Welcoming question/suggestions, now or later by email:

julien@cornebise.com
Discussion: Resampling and Variance

Variance of posterior mean estimate for Stochastic Volatility:

Estimated Variance for fixed number of particles

- Bootstrap
- Residual
- Stratified resampling
- Systematic resampling
- Stratified residual resampling
- Systematic residual resampling

J. Cornebise

Practicalities in implementing Adaptive Particle MCMC
Discussion: Stratified Resampling (1/2)

- Resampling I recommend: **Stratified**
  - Even faster than sorted multinomial
  - Lower resampling variance

```matlab
function I = randoindsort(p, N)
cdf = cumsum(p);
cdf(end) = 1;
I = zeros(N, 1);
U = rand(N, 1);
U = sort(U);
index = 1;
for k = 1:N
    while (U(k) > cdf(index))
        index = index + 1;
    end
    I(k) = index;
end
```

```matlab
function I = randoindstrat(p, N)
cdf = cumsum(p);
cdf(end) = 1;
I = zeros(N, 1);
U = rand(N, 1);
U = U/N + (0:(N-1))/N;
index = 1;
for k = 1:N
    while (U(k) > cdf(index))
        index = index + 1;
    end
    I(k) = index;
end
```

Systematic resampling (not shown): fastest, but no proof of CLT!
Discussion: Stratified Resampling (2/2)

![Graph showing speed-up factors for different resampling methods: Naive, Sorted, Residual Naive, Residual Sorted, Stratified, Systematic. The x-axis represents N (number of samples), and the y-axis represents speed-up factor. The lines for each method show the speed-up factor as N increases.]