Supplementary: A Nearly-Black-Box Online Algorithm for Joint Parameter and State Estimation in Temporal Models

Yusuf B. Erol[†*] **Yi Wu**[†*] **Lei Li**[‡] **Stuart Russell**[†]
[†] EECS, UC Berkeley {yberol,jxwuyi,russell}@eecs.berkeley.edu
[‡] Toutiao Lab lileicc@gmail.com

Bootstrap particle filter as a subcase of APF

Here we will show that when q is a delta function, APF recovers the bootstrap particle filter. The Dirac delta function can be considered as the limit of a Gaussian as the variance goes to zero, $\delta(\theta-\mu)=\lim_{\sigma^2\to 0}\mathcal{N}(\theta;\mu,\sigma^2)$. Therefore, we can view q as an exponential family distribution. Specifically we are dealing with a Gaussian distribution with unknown mean and known variance (zero-variance). Then the moment matching integral required for assumed density filtering reduces to matching the means. If $q_{i-1}=\delta(\theta-\mu_{i-1})$, then

$$\begin{aligned}\mu_i=\int\theta q_i(\theta)d\theta &= \int\theta\hat{p}(\theta)d\theta\\ &= \frac{\int\theta s_i(\theta)\delta(\theta-\mu_{i-1})d\theta}{\int s_i(\theta)\delta(\theta-\mu_{i-1})d\theta}\\ &= \frac{\mu_{i-1}s_i(\mu_{i-1})}{s_i(\mu_{i-1})}=\mu_{i-1}\end{aligned}\quad(5)$$

where the last equality follows from the *sifting property* of the Dirac delta function. The main result here is that for $\mathcal{Q}$ Dirac delta, $\mu_i=\mu_{i-1}$; that is, the APF Update step does not propose new values. Therefore, our proposed algorithm recovers the standard bootstrap particle filter.

Proof for Theorem 1

In this section[1] we will assume an identifiable model where the posterior distribution approaches normality and concentrates in the neighborhood of the posterior mode. Suppose $\hat{\theta}$ is the posterior mode and hence the first-order partial derivatives of $\log p(\theta\mid x_{0:T},y_{0:T})$ vanish at $\hat{\theta}$. Define

$$\hat{I}=-\frac{\partial^2\log p(\theta\mid x_{0:T},y_{0:T})}{\partial\theta\partial\theta^T}\bigg|_{\theta=\hat{\theta}}\quad(6)$$

Applying a second-order Taylor approximation around $\hat{\theta}$ to the posterior density results in

$$\log p(\theta\mid x_{0:T},y_{0:T})\approx\log p(\hat{\theta}\mid x_{0:T})-\frac{1}{2}(\theta-\hat{\theta})^T\hat{I}(\theta-\hat{\theta})$$

Hence;

$$p(\theta\mid x_{0:T},y_{0:T})\propto\exp\left\{-\frac{1}{2}(\theta-\hat{\theta})^T\hat{I}(\theta-\hat{\theta})\right\}\quad(7)$$

which is a p ($\theta\in\mathbb{R}^p$) dimensional Gaussian density with mean $\hat{\theta}$ and covariance $\hat{I}^{-1}$. As the posterior becomes highly concentrated in a neighborhood of the posterior mode, the effect of the prior on the posterior diminishes, which is the Bernstein-von Mises theorem. Then we can rewrite Eq. 7 as

$$p(\theta\mid x_{0:T},y_{0:T})\propto\exp\left\{-\frac{T}{2}\sum_{ij}(\theta_i-\hat{\theta}_i)\hat{J}_{ij}(\theta_j-\hat{\theta}_j)\right\}$$

where $\hat{J}_{ij}=-\partial_i\partial_j\frac{1}{T}\sum_{t=1}^{T}\log s_t(\hat{\theta})$ and $s_t(\theta)=p(x_t\mid x_{t-1},\theta)p(y_t\mid x_t,\theta)$.

The assumed density filter updates for the Gaussian case has been derived in earlier sections. We will reorganize them in a more convenient form.

$$\begin{aligned}\mu_i(t) &= \frac{\int\theta_i s_t(\theta)q_{t-1}(\theta)d\theta}{\int s_t(\theta)q_{t-1}(\theta)d\theta}\\ \Sigma_{ij}(t) &= \frac{\int\theta_i\theta_j s_t(\theta)q_{t-1}(\theta)d\theta}{\int s_t(\theta)q_{t-1}(\theta)d\theta}-\mu_i(t)\mu_j(t)\end{aligned}$$

We will use a simple property of centered Gaussian random variables z, $E[zf(z)]=E(f^{'}(z)).E(z^2)$ which can be proven by applying integration by parts. Then the explicit updates can be written as follows:

$$\begin{aligned}\mu_i(t) &= \mu_i(t-1)\\ &+ \sum_j\Sigma_{ij}(t-1)\times\partial_j\log E_u[s_t(\mu(t-1)+u)]\\ \Sigma_{ij}(t) &= \Sigma_{ij}(t-1)\\ &+\sum_{kl}\Sigma_{ik}(t-1)\Sigma_{lj}(t-1)\partial_k\partial_l\log E_u[s_t(\mu(t-1)+u)]\end{aligned}\quad(8)$$

where u is a zero-mean Gaussian random vector with covariance $\Sigma(t-1)$. We define $V_{kl}=\partial_k\partial_l\log E_u[s_t(\theta+u)]$ and assume that for large times we can replace the difference equation for $\Sigma(t)$ with a differential equation. Then we can rewrite Eq. 8 as

$$\frac{d\Sigma}{dt}=\Sigma V\Sigma\quad(9)$$

[*]The first two authors contributed equally to this work.

[1]Our discussion follows (Opper and Winther 1998) which considers the asymptotic performance of assumed density filtering for independent identically distributed data.

which is solved by

$$\frac{d\Sigma^{-1}}{dt} = -V \tag{10}$$

Integrating both sides

$$\Sigma^{-1}(t) - \Sigma^{-1}(t_0) = -\int_{t_0}^{t} V(\tau)d\tau \tag{11}$$

For large t ; we expect the covariance Σ to be small such that $\log E_u[s_t(\mu+u)] = \log s_t(\mu)$. Assuming that the online dynamics is close to θ^* and dividing both sides of Eq. 11 by t and taking the limit $t \to \infty$, we get

$$\lim_{t\to\infty} \frac{(\Sigma^{-1}(t))_{ij}}{t} = \lim_{t\to\infty} \frac{-\int_{t_0}^{t} \partial_i\partial_j s(\theta^*)}{t} \tag{12}$$

Further assuming ergodicity (i.e., markov process converging to some stationary distribution π), we can replace the time average with the probabilistic average.

$$\lim_{t\to\infty} \frac{(\Sigma^{-1}(t))_{ij}}{t} = \tag{13}$$
$$-\int \pi(x)p(x' \mid x, \theta^*)\partial_i\partial_j \log s(\theta^*)dxdx'$$

If we define the right hand side as $A_{ij} = -\int \pi(x)p(x' \mid x, \theta^*)\partial_i\partial_j \log p(x' \mid x, \theta^*)p(y \mid x', \theta^*)dxdx'$ we have;

$$\lim_{t\to\infty} \Sigma(t) = \frac{A^{-1}}{t} \tag{14}$$

We will also analyze the asymptotic scaling of the estimation error, defined as the deviation between θ^* and $\mu(t)$. Assuming that the estimate μ is close to θ^* and the posterior is sharply concentrated we can neglect the expectation with respect to u in Eq.8. Defining $\mu_i(t) = \theta_i^* + \epsilon_i(t)$, and applying a first order Taylor approximation to Eq. 8 around θ^* we get;

$$\epsilon_i(t+1) - \epsilon_i(t) = \sum_{\ell} \Sigma_{i\ell}\partial_\ell \log P + \sum_{k\ell} \Sigma_{i\ell}\epsilon_k(t)\partial_k\partial_l \log P$$

where $P \equiv p(x' \mid x, \theta^*)p(y \mid x', \theta^*)$. Taking the expectation with respect to the stationary distribution (denoted by an overbar) and using the relationship in Eq. 14 and replacing the difference equation with a differential equation we get an equation of motion for the expected error $e_i = \bar{\epsilon_i}$.

$$\frac{de_i}{dt} + \frac{e_i}{t} = \sum_j \frac{(A^{-1})_{ij}}{t}\overline{\partial_j \log P} \tag{15}$$

As $t \to \infty$ right hand side vanishes and hence the error term decays like $e_i \propto \frac{1}{t}$.

Revisiting Eq. 7, the true posterior covariance matrix is given by $C^{-1} = T\hat{J}$. Due to our ergodicity assumption, $\lim_t \hat{J} = A$. Hence the true posterior density covariance asymptotically converges to A^{-1}/T which is the same limit as the assumed density filter covariance $\Sigma(t)$.

KL divergence between two d-dimensional multivariate Gaussians, $\mathcal{N}(\mu_1, \Sigma_1)$ and $\mathcal{N}(\mu_2, \Sigma_2)$ is given by

$$\frac{1}{2}\left[\log\frac{|\Sigma_2|}{|\Sigma_1|} - d + \text{tr}(\Sigma_2^{-1}\Sigma_1) + (\mu_2 - \mu_1)^T\Sigma_2^{-1}(\mu_2 - \mu_1)\right]$$

We have shown that $\lim_{t\to\infty} C, \Sigma = A^{-1}/t$ and $\mu(t) \to \theta^*$. Due to the identifiability assumption, the posterior mode is also assumed to converge to the parameter θ^*. Applying these findings to the earlier KL-divergence formula, we can see that;

$$\lim_{t\to\infty} D_{KL}(p(\theta \mid x_{0:t}, y_{0:t})||q_t(\theta)) = 0. \tag{16}$$

For the `SIN` model discussed in the experiments section, the true posterior $p(\theta \mid x_{0:t})$ is computed for a grid of parameter values in $O(t)$ time per parameter value. Assumed density filtering is also applied with $\mathcal{Q}$ Gaussian and the true density (solid) vs. assumed density (dashed) is illustrated in Fig. 2. Notice that, ADF is slightly off at earlier stages, however, does indeed catch up with the ground truth with more data.

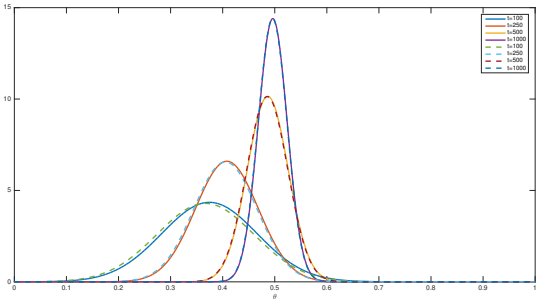


Figure 2: True posterior $p(\theta \mid x_{0:t})$ vs assumed density filter estimate $q_t(\theta)$ (solid vs dashed line respectively). for the `SIN` model.

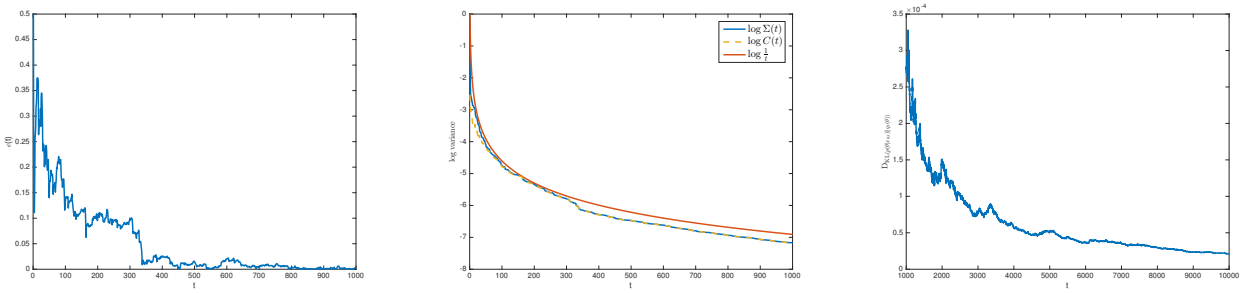
As predicted by Eq. 15, the error term converges to zero as shown in Fig. 3(a). Figure 3(b) illustrates the asymptotic behavior of the true posterior covariance $C(t)$ and assumed density covariance $\Sigma(t)$. Assumed density filter quickly approximates the covariance. Most importantly, as can be seen from the plot, $\log C(t)$ and $\log \Sigma(t)$ is $log(1/t)$ + constant asymptotically, and this agrees with our derivations in Eq. 14. Figure 3(c) confirms our theoretical result of KL divergence converging to zero in the long-sequence limit.

More Details of Experiments

Choice of M in APF: Generally, for continuous parameters, thanks to the Gaussian quadrature rule, a small M is enough (7 in `SIN` and 15 in `BIRD`). For discrete parameters, a larger M is required (100 in `SLAM`). Note that the choice of M does not depend on K or T.

Toy nonlinear model (`SIN`)

The reason for generating 5000 data points is to ensure the true posterior converges to a sharp mode around the true parameter 0.5, as shown in Fig. 2. Due to the sharp posterior, PMMH with a naive proposal or PGibbs without enough particles will mix very slowly.



(a) Error term $\epsilon(t)$ (b) Covariance term $\Sigma(t)$ (c) KL divergence

Figure 3: SIN model with $\theta^* = 0.5$

When given too few data points, the posterior becomes flat and makes the approximation problem easy for the inference algorithms.

The bimodal variant of SIN demonstrates the applicability of APF for models with a multi-modal posterior when using mixtures of Gaussians as the approximate distribution. As shown in Fig.1(b), increasing the number of mixtures will help improve the chance for finding all the possible modes while an inappropriate approximate distribution ($L = 2$) can lead to large bias. Lastly, although APF needs more mixtures to find multiple modes, it still converges much faster than PMCMC in this specific example.

Simultaneous localization and mapping (SLAM)

The 41 actions consist of 21 consecutive right-move actions in the beginning and 20 left-move actions afterwards. Initially the robot locates on the leftmost cell. We color all the odd cells 1 and even cells 0. The robot slipped at time 4 and always observed the true color of the cell. In this case, the true posterior of the colors conditioning on the observations is very sharp and close to the true colors.

Note that randomly generated data may lead to a very flat posterior close to the prior, on which even the plain particle filter may work. While, with our specially designed data, a single particle may succeed in guessing the true color only with probability of 2^{-20}.

Likewise, PMCMC using a proposal resampling the colors of all the cells will fail either. Therefore, in the experiment, we choose a proposal that resamples only a single cell per iteration. In our experiments, as N increases, the KL-divergence decreases very slowly. We suspect that, although better than resampling all the cells, the chosen proposal is still not enough for PMCMC to mix fast. By contrast, APF can quickly converge to the correct posterior even using a very small number of particles (K) as shown in Fig.1(d).

Tracking bird migration (BIRD)

In the dataset of the bird migration problem, there are roughly 10^6 birds observed. The eastern continent of the U.S.A. is partitioned into a 10x10 grid. For each grid cell, the total number of birds is observed on 60 days. We aim to infer the number of birds migrating at different grid locations between two consecutive days.

BIRD can be viewed as complicated HMM model where both the transition and observation likelihood depend on the parameters.

Here is a simplified BLOG program for the BIRD model.

```
//parameters
random Real beta1 ~ UniformReal(0, 10);
random Real beta2 ~ UniformReal(0, 10);
random Real beta3 ~ UniformReal(0, 10);
random Real beta4 ~ UniformReal(0, 10);
//pre-computed features, details omitted
fixed RealMatrix F1(Loc src) = ...
fixed RealMatrix F2(Loc src) = ...
fixed RealMatrix F3(Loc src,Timestep t)=...
fixed RealMatrix F4(Loc src) = ...
//flow probabilities (states)
random RealMatrix probs(Loc src, Timestep t)~
  exp(beta1 * F1(src) + beta2 * F2(src)
    + beta3 * F3(src,t) + beta4 * F4(src));
//number of birds at location loc (state)
random Integer birds(Loc loc, Timestep t) ~
  if t == @0 then initial_value[loc]
  else
  sum({outflow(src,prev(t))[dst] for Loc src});
//outflow from src to other locations (state)
random RealMatrix outflow(Loc src, Timestep t)
  ~ Multinomial(birds(src, t), probs(src, t));
//Noisy Observations
random Integer NoisyObs(Loc loc, Timestep t)~
  if birds(loc, t) == 0 then Poisson(0.01)
  else Poisson(birds(loc, t));
```

Compilation Optimizations in SPEC

In this section, we introduce some optimizations in the implementation of SPEC. These techniques are mostly standard techniques from programming languages and adopted by many modern PPL compilers. All the source code of SPEC will be released after the paper gets accepted.

Memory Efficiency

Memory Pre-allocation: Memory management is a critical issue for particle filter algorithms since it is often necessary to launch a large number of particles in practice. Systems that does not manage the memory well will repeatedly allocate memory at run time. For example, the original inference engine of BLOG will allocate memory for new particles and

erase the memory belonging to the old ones after each iteration. This introduces tremendous overhead since memory allocation is slow.

By contrast, SPEC will analyze the input model and allocate the minimum static memory for computation: if the user specifies to run K particles, and the Markov order of the input model is D, SPEC will allocate static memory for $(D+1)*K$ particles in the target code. When the a new iteration starts, we utilize a rotational array to re-use the memory of previous particles.

Lightweight Memoization: Notice that adopting an expressive language interface (i.e. syntax of BLOG) might lead to time-varying dependencies between random variables. Similar to the compilation of lazy evaluation in programming language community, SPEC memoizes the value for each random variables already sampled. An example compiled code fragment for the `SIN` is shown below.

```
class TParticle { public:
// value of x and y at current timestep
  double val_x, val_y;
// flag of whether x/y is sampled
  int mark_x, mark_y;
} particles[K][DEP]; // particle objects
// getter function of y(t)
double get_y(int t) {
// get the corresponding particle
  TParticle &part = get_particle(t);
  // memoization for y(t)
  if(part.mark_y == cur_time) return part.val_y;
  part.mark_y = cur_time;
// sample y(t), call getter function of x(t)
  part.val_y = sample_normal(
                sin(get_theta() * get_x(t-1)),
                0.5);
  return part.val_x;
}
```

This code fragment demonstrate the basic data structures as well as the memoization framework in the target code. `K` denotes the number of particles. `DEP` is equal to the Markov order of the model plus 1. In the memoization framework, SPEC allocates static memory for all the random variables within a particle data structure and generates a flag for each random variable denoting whether it has been sampled or not. Each random variable also has its *getter* function. Whenever accessing to a random variable, we just call its getter function. In practice, memoization causes negligible overhead.

Memoization also occurs in BLOG. However, every random variable in BLOG has a string "name", and a generic hashmap is used for value storage and retrieval, which brings a great deal of constant factor at run time.

Other Optimizations: SPEC also avoids dynamic memory allocation as much as possible for intermediate computation step. For example, consider a multinomial distribution. When its parameters change, a straightforward implementation to update the parameters is to re-allocate a chunk of memory storing all the new parameters and pass in to the object of the multinomial distribution. However, this dynamic memory allocation operation is also avoided in SPEC by pre-allocating static memory to store the new parameters.

Computational Efficiency

Pointer References: Resampling step is critical for particle filter algorithms since it requires a large number of data copying operations. Note that a single particle might occupy a large amount of memory in real applications. Directly copying data from the old particles to a new ones induce tremendous overhead.

In SPEC, every particle access to its data via an indirect pointer. As a result, redundant memory copying operations are avoided by only copying the pointers referring to the actual particle objects during resampling Note that each particle might need to store multiple pointers when dealing with models with Markov order larger than 1. An example compiled code is shown below.

```
// indirect pointers to the actual particle
TParticles* part_ptrs[K][DEP];
TParticle& get_particle(int t) {// rotational array
  return *part_ptr[cur_part][t % DEP]; }
```

Locality: Besides using pointer references to reduce the amount of moving data, SPEC also enhances program locality to speed up re-sampling. In the compiled code, the index of the arrays stores the indirect pointers are carefully designed to take advantage of memory locality when copying. The fragment of code used in resampling step is shown below.

```
void resample_ptr( int* target,
  TParticle* part_ptr[K][DEP], //storage of  pointers
  TParticle* backup_ptr[K][DEP]) // temporary storage
{ for (int i = 0; i < K; ++i) {
    // pos: index of particle to copy data from
    int& pos = target_ptr[i];
    // move continuous range of data
    std::memcpy(backup_ptr[i], part_ptr[pos],
                sizeof(TParticle*)* DEP);  }
  std::memcpy(ptr_temp_memo, backup_ptr,
              sizeof(TParticle*)* DEP * K);
}
```

In the preceding code fragment, the first dimension of the storage array, `part_ptr`, corresponds to the particle index. While the second dimension corresponds to the current iteration. In this case, when copying the pointers during resampling step, all the memory are continuously located in the memory space, which reduces the constant factor at run time.

References

Opper, M., and Winther, O. 1998. A Bayesian approach to online learning. *On-line Learning in Neural Networks, ed. D. Saad* 363–378.