Università degli Studi di Padova

Department of Information Engineering
University of Padova

Estimation and Filtering

Markov Chain Monte Carlo and Particle Filter

Marco Mustacchi

1 Markov Chain Monte Carlo

In this first part, a static estimation problem will be addressed by resorting to methods *Markov chain Monte Carlo* (MCMC). Specifically, given the scalar measures model:

$$y_i = f_i(\theta) + w_i, \quad i = 1, ..., N$$

one is asked to reconstruct the random vector $\theta \in \mathbb{R}^m$.

Choice of model

I relied on the scalar state model:

$$\begin{aligned} \dot{x}(t) &= a\,x(t) + b\,t \\ y(t) &= x(t) \end{aligned}$$

where a, b and the initial state $x(0)$ are the unknown real parameters that will have to be reconstructed from noisy measurements of $y(t)$.

From the theory on state models we have that:

$$\begin{aligned} y(t) &= x(t) \\ &= e^{a\,t}x(0) + \int_0^t e^{a(t-\tau)} b\,\tau\, d\tau \\ &= e^{a\,t}x(0) + \frac{b}{a^2}(e^{a\,t} - a\,t - 1) \end{aligned}$$

Measurements y_i are noisy samples of $y(t)$ acquired every $T_s = 1\,ms$[1], so:

$$y_i = y\left(\frac{i}{T_s}\right) + w_i, \quad i = 1, ..., N$$

where $w_i \sim \mathcal{N}(0, \sigma_w^2)$.

I thought of modeling the measurement error with a Gaussian density since it is known to be an extremely good model for modeling measurement errors in a great many practical cases, this also thanks to the Central Limit Theorem (CLT).

By positing $\theta = \begin{bmatrix}\theta_1 & \theta_2 & \theta_3\end{bmatrix}^T := \begin{bmatrix}x(0) & a & b\end{bmatrix}^T$ one therefore has:

$$f_i(\theta) = e^{\theta_2\, i/T_s}, \theta_1 + \frac{\theta_3}{\theta_2^2}(e^{\theta_2\, i/T_s} - \theta_2\, i/T_s - 1)$$

Regarding the number of measurements, I set $N = 1000$ so as to capture the dynamics of the system in a sufficiently large time interval relative to the sampling period, thus allowing us to capture information regarding even the slowest dynamics, which in this case may be those related to θ_3.

Reasons for choosing the model

I chose this model because the functions $f_i(\theta)$ are deeply nonlinear in θ, this is because exponentials, ratios and squares of the θ variables appear in them. The nonlinearity of such functions means that the posterior distribution $p(\theta|y)$ may be non-Gaussian, despite the noise w_i being so; in particular, the nonlinearity θ_2^2 may be a cause of bimodality of the posterior.

The non-Gaussianity of the posterior distribution can very easily lead to integrals that are extremely difficult or even impossible to compute in closed form, necessitating the use of alternative methods to approximate the integral. There are deterministic and stochastic methods to deal with this problem; the **deterministic** ones seen in class are:

- **Conventional numerical methods** (quadrature methods): resort to quadrature formulas to approximate the area of the integral with sums of areas of polygons.

  The main limitation of these approaches is the *dimensionality curtailment*, which limits the use of these methods to spaces of dimension not greater than 2.

- **Laplace asymptotic analytic approximation**: consists of approximating the a posteriori distribution with a Gaussian. These methods can work even in rather large dimensions.

[1] I have chosen a "small" sampling period to optimally capture the dynamics of the system, which can be very fast because of the exponential $e^{\theta_2\, i/T_s}$.

The main limitation is due to the unimodality of the Gaussian: if the real posterior is multimodal, approximation with a Gaussian may be completely unreasonable.

Thus, it is concluded that, in this case, neither method is reasonable: the vector of unknowns is a random vector of dimension 3, which rules out conventional numerical methods; the nonlinearity of $f_i(\theta)$ means that the posterior can be multimodal, which rules out Laplace's asymptotic analytic approximation.

Instead, the **stocastic** methods seen in class are:

- **Monte Carlo Approaches**: Exploits the realizations $x_i, i = 1, ..., n$ *i.i.d.* of the a posteriori distribution $\pi(x)$ to perform the approximation:

  $$E_\pi(g) := \int g(x)\pi(x)dx \simeq \frac{1}{n}\sum_{i=1}^{n} g(x_i)$$

  By the strong law of large numbers you have:

  $$\lim_{n\to+\infty} \frac{1}{n}\sum_{i=1}^{n} g(x_i) \overset{q.c.}{=} E_\pi(g)$$

  Good approximations of the integral require the generation of a large number of independent samples from π. The main limitation of this approach lies precisely in the difficulty of extracting *i.i.d.* samples from *pi* in the case where it is a multivariate, nonstandard distribution. For the extraction of such samples we have seen the method of *Rejection sampling*, which, however, is often difficult to apply in high dimensions (difficulty in choosing the auxiliary distribution, curse of dimensionality).

- **Markov chain Monte Carlo**: This method goes to introduce correlation between samples extracted from the auxiliary density to circumvent the difficulties associated with generating *i.i.d.* samples that we mentioned in the previous method. This allows it to be an extremely effective approach even in the case of multivariate and nonstandard distributions.

Therefore, it is deduced that the most suitable method to solve the static estimation problem in question is precisely Markov chain Monte Carlo (MCMC).

Implementation of MCMC

The first part of the MCMC method consists in creating an irreducible Markov chain whose invariant density is the posterior $p(\theta|y) =: \pi$, to do this I implemented the Metropolis-Hastings algorithm.

A fundamental step of that algorithm consists in choosing the *proposal density* $q(.|.)$, in particular it must be:

- easy to simulate
- easy to evaluate on time
- such that the support of π can be quickly explored.

I opted for a proposal of type *random-walk*, specifically a Gaussian random-walk:

$$c = x_t + \varepsilon$$
$$\varepsilon \sim \mathcal{N}(0, \Sigma)$$

where x_t is the current state of the chain and c is the proposed new state. This proposal is extremely easy to simulate and evaluate punctually, and "by playing around" with the covariance matrix Σ as I will illustrate below I was also able to make it very viable for a quick exploration of π. Moreover, this proposal also makes it possible to greatly simplify the acceptance probability, which in this case takes the form:

$$\alpha(c, x) = \min\left(1, \frac{\pi(c)\cancel{q(x|c)}}{\pi(x)\cancel{q(c|x)}}\right)$$
$$= \min\left(1, \frac{\pi(c)}{\pi(x)}\right)$$

since $q(c|x) = f(|c - x|) = q(x|c)$.

Initially, I chose a constant Σ covariance matrix, i.e., equal from the first to the last iteration of the Markov chain, so as to have (after burn-in) an average acceptance probability of 35%; however, this resulted in a decidedly high number of samples in the burn-in, due to the high distance between the invariant density and the starting point of the Markov chain. It is possible to observe this fact in Figure 1, where the high number of samples "wasted" in the burn-in is clearly visible.

To solve this problem, I increased the variance of the *random-walk* during the burn-in phase, so that the chain could reach the invariant distribution more quickly. By doing so, it was possible to reduce the number of burn-in samples by as much as 3 times[2] (from 3000 to 1000), allowing more samples to be taken from the invariant distribution (2000 more for the same total iterations of the Markov chain). It is possible to see the result in Figure 2.

I had thought of moving the components of the state vector separately, but by implementing this scheme I noticed a worsening in the execution time not compensated by the improvement in the estimates obtained, as both histograms and averages were very similar to the case where I move all components together. The non-necessity of moving one component at a time is due to the "simple" shape of the density π, which is a kind of ellipsoid, as can be seen in Figure 3 (generated using MATLAB's *scatter3* function).

[2]Result obtained using the recommended parameters

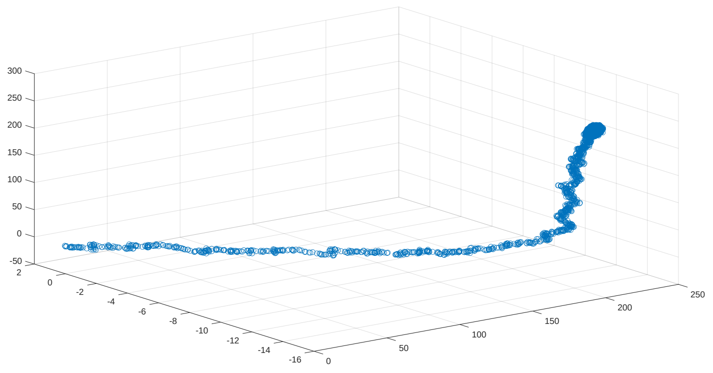

FIGURE 1: Markov chain: burn-in and convergence to the invariant distribution can be observed (parameters are: 200, −15, 250).

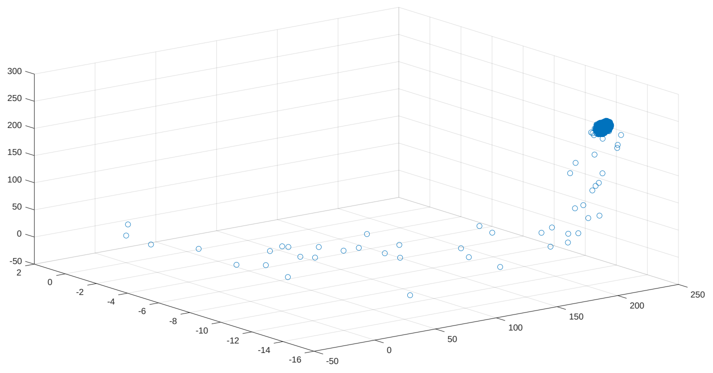

FIGURE 2: Markov chain obtained by increasing the random-walk pitch during the burn-in phase.

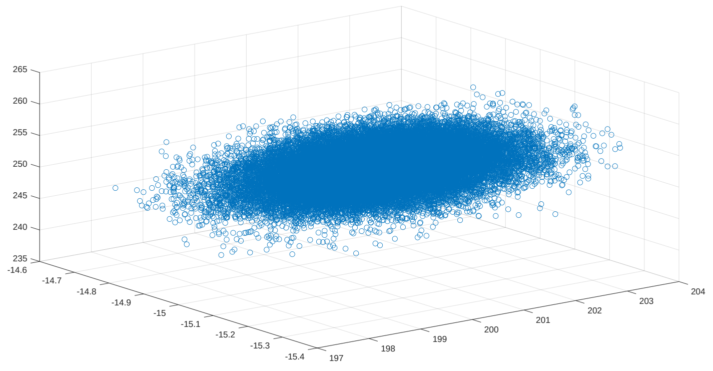

FIGURE 3: Three-dimensional view of the invariant distribution of the Markov chain.

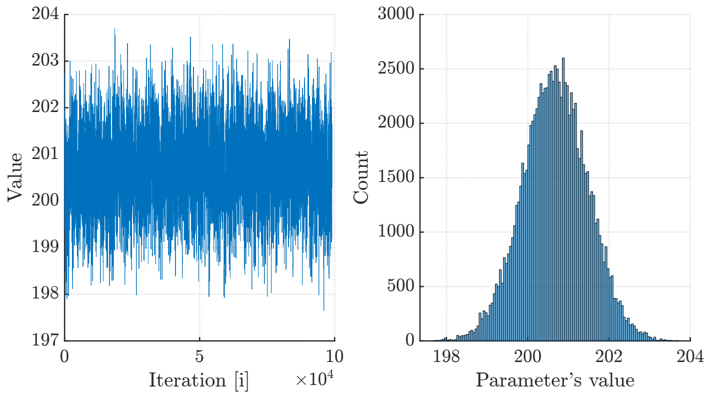

FIGURE 4: Trajectory and histogram of θ_1

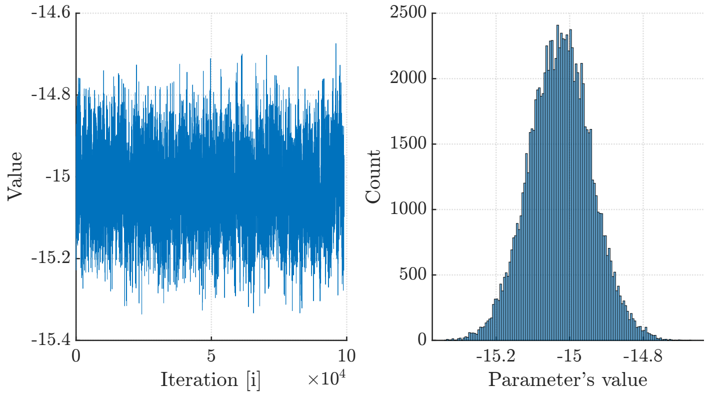

FIGURE 5: Trajectory and histogram of θ_2

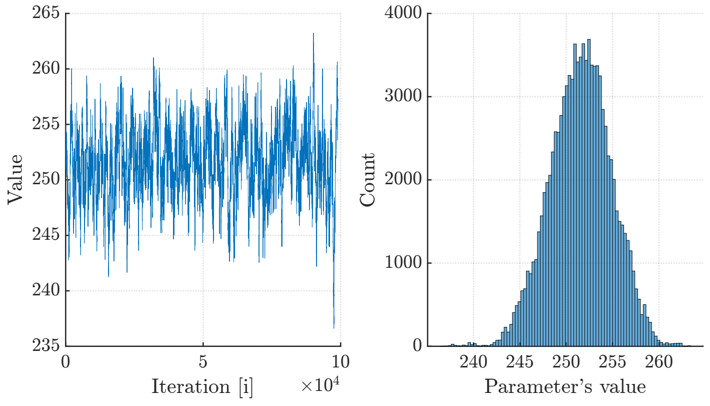

FIGURE 6: Trajectory and histogram of θ_3

In the Figures 4, 5, 6 it is possible to see the good convergence of the Markov chain and to view the histograms inherent in the 3 parameters, whose true values are: $\theta_1 = 200$, $\theta_2 = -15$, $\theta_3 = 250$. The minimum error variance estimate obtained by the Monte Carlo method for the 3 parameters is:

- $\hat{\theta}_1 = 200.2669$
- $\hat{\theta}_2 = -15.0056$
- $\hat{\theta}_3 = 249.8373$

Looking at the histograms shown in the 3 figures, it is also very easy to get an idea about the confidence intervals.

Final remarks

Interestingly, the estimates obtained are very accurate despite the virtually zero information contributed by the prior. By having a priori knowledge about the distribution of the parameters one could obtain even more accurate estimates, or increase the allowable range of variability for the parameters.

In fact, by going to extreme variations in the parameters, I came across cases in which the Markov chain was being "wronged" by likelihood, in that the output obtained with the wrong proposals was very close to the measurements ($\Rightarrow$ likelihood large), this attracted the chain to wrong parts of the space, greatly lengthening its burn-in and eventually leading to completely wrong θ estimates, proposing, for example, positive θ_2 when, since it is the eigenvalue of the state model, it is assumed to be negative (stable). In this case, the wrong estimates are due to the fact that virtually all samples in the chain are burn-in, thus not allowing the extraction of samples from π. Indeed, in these critical situations, in order to allow the chain to explore the invariant distribution and thus obtain reliable estimates, it was necessary to greatly increase the number of iterations.

This unfortunate event could be avoided by going to include in the prior information about the nonpositivity of θ_2, thus preventing the Markov chain from being drawn into "wrong" regions of space.

2 Particle filter

The second part of the paper consists of an estimation problem *online* using particle filters.

Choice of model

For the choice of model state transition equations, I was inspired by an example seen in the course, which testified to the potential of particle filters when dealing with nonlinear state models. That example consisted of a scalar state model, which I proceeded to complicate by including a second state variable. Since I also really liked the example of tracking a robot, I assumed that the two state variables of the model are nothing more than the coordinates in the plane of a robot, which are to be reconstructed through noisy measurements acquired by two sensors placed at the points $\begin{bmatrix}-10 & 0\end{bmatrix}^T$, $\begin{bmatrix}0 & 10\end{bmatrix}^T$, which measure the distance of the robot from them.

The model in question is as follows:

$$\begin{aligned}
x_1(k+1) &= \frac{1}{2}x_1(k) + \frac{25x_1(k)}{x_1^2(k)+1} + x_2(k)\cos(29k) + v_1(k)\\
x_2(k+1) &= \frac{1}{5}x_2(k) + \frac{1}{10}\frac{x_1(k)}{x_2^2(k)+1} + 15\sin(10\cos(2k)) + v_2(k)\\
y_1(k) &= \sqrt{(x_1(k)+10)^2 + x_2^2(k)} + w_1(k)\\
y_2(k) &= \sqrt{x_1^2(k) + (x_2(k)-10)^2} + w_2(k)
\end{aligned}$$

Where v_i and w_i are Gaussian noises with zero mean, flowing with each other and such that $cov[v_1(k), v_2(s)] = \sigma_v^2 I_2 \delta(k-s)$, $cov[w_1(k), w_2(s)] = \sigma_w^2 I_2 \delta(k-s)$, $\sigma_v^2 = 10$, $\sigma_w^2 = 1$.

Reasons for choosing the model

The state model in question is profoundly nonlinear because of the squares that appear in the state transition and output equations; in fact, they cause bimodality in the posterior. The fact that the posterior is bimodal means that the extended Kalman filter cannot be used for state estimation in this model, since the EKF is only capable of propagating Gaussian distributions; in fact, it is not at all reasonable to approximate the bimodal distribution with a Gaussian (unimodal) one. Being faced with such a situation the extended Kalman filter would have to in practice "bet" on one of the two peaks of the distribution, and in case it bets on the wrong peak it would be definitely compromised the tracking of the robot even in the following instants. Clearly this rules out the use of the EKF to address the problem at hand, making it necessary to resort to more powerful tools, such as particle filters. I will now provide an illustration and rationale for the choices I made to implement the parcel filter.

Proposal

I chose a Markovian proposal density so as to minimize the amount to be held in memory during program execution. In fact, thanks to the Markovian hypothesis we have:

$$q(x_k^i | x_{0:k-1}^i, y_{1:k}) = q(x_k^i | x_{k-1}^i, y_k)$$

To further simplify the proposal, and thus further speed up the algorithm, I chose:

$$\begin{aligned}
q(.|.) &= p(x_k^i | x_{k-1}^i)\\
&= \frac{\exp\{-\frac{1}{2\sigma_v^2}(x_k - f(x_{k-1}))^T(x_k - f(x_{k-1}))\}}{\sqrt{(2\pi)^2(\sigma_v^2)^2}}
\end{aligned}$$

where f is the (vector) state update function of the model illustrated above.

This proposal makes it possible to greatly simplify the update of particle weights, which becomes:

$$w_k^i \propto w_{k-1}^i \, p(y_k | x_k^i)$$

Then the new weights are simply obtained by multiplying the previous ones by the likelihood (minus a proportionality constant).

As will be seen below this very simple proposal works decidedly well in this model, and therefore I did not go in search of more complicated proposals.

Regeneration

The choice of regeneration for the particle filter was practically forced, since without regeneration already after about ten iterations there was degeneration, viz:

$$\frac{1}{\sum_{i=1}^{N_S}(w_k^i)^2} =: N_{eff} \simeq 1$$

and this with both $N_S = 10^3$ and $N_S = 10^5$. It is possible to observe this fact in Figure 7.

For the implementation of the resampling algorithm I relied on the pseudo-code given in the handout illustrated during the lectures, choosing $N_T = N_S/2$ as the threshold.

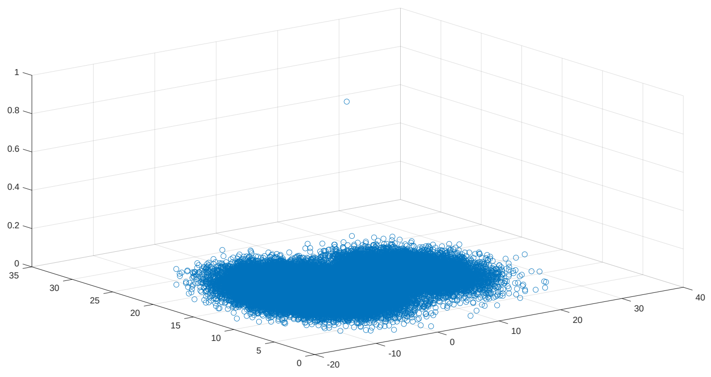

FIGURE 7: Degeneration: only one particle in 10^5 has a relevant weight.

Generation of the first set of particles

To generate the first set of particles, I relied on the prior statistic, defined by mean μ and variance $\sigma^2 I$. Specifically, I generated the first set of particles by extracting N_S realizations from the distribution $\mathcal{N}(\mu, \sigma^2 I)$ using the MATLAB function *randn*.

Results

Figure 8 shows the true-state trajectory and the minimum variance error estimate obtained from the particle filter, for each of the two state variables. Also illustrated is the confidence interval at 95%, which was obtained by summing for each instant k the (normalized) weights of the particles contained in a disk centered in $(\hat{x}_1(k), \hat{x}_2(k))$[3] with radius such that it has sum $\simeq 0.95$.

It is possible to appreciate the goodness of the obtained estimate, to which corresponds (in this example) error variance of $\simeq 5.9475$ for x_1 and $\simeq 4.3335$ for x_2.

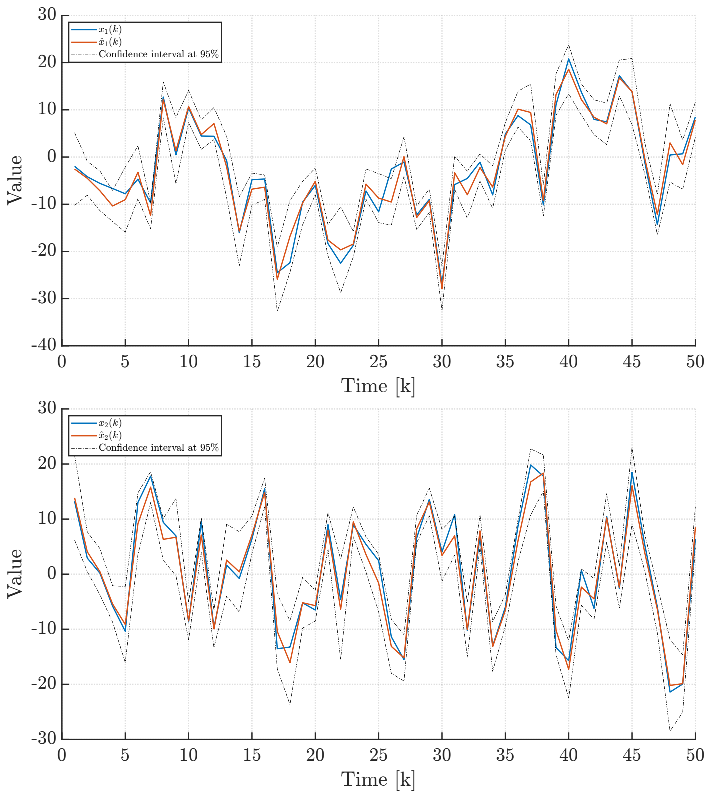

FIGURE 8: Trajectories of the state variables and relative minimum error variance estimation obtained by particle filtering. These results were obtained by entering the recommended values.

[3]Minimum error variance estimate of the robot's position at time k obtained by the Monte Carlo method.