State-space models: bootstrap filter and particle MCMC

MVE187-MSA101 “Computational methods for Bayesian statistics”, 2022

Umberto Picchini
`@uPicchini`, picchini@chalmers.se

Chalmers University of Technology and University of Gothenburg
Sweden

Overview

- Today we continue with inference for state-space models.
- We find out why the basic SIS was unsatisfactory and fix it to obtain a better likelihood approximation.
- We also (informally) look at the *filtering distribution*.
- We look at how to perform parameter inference by embedding particle filters into Metropolis-Hastings.

I quickly summarise some of the findings of the previous lecture.

We have the likelihood function

$$p(y_{1:T}|\theta) = p(y_1|\theta) \prod_{t=2}^{T} p(y_t|y_{1:t-1}, \theta)$$

and mentioned that we wanted to approximate the likelihood via importance sampling as

$$p(y_t|y_{1:t-1}) = \int \frac{p(y_t|x_t)p(x_t|x_{t-1})p(x_{0:t-1}|y_{1:t-1})}{h(x_{0:t}|y_{1:t})} h(x_{0:t}|y_{1:t}) dx_{0:t}.$$

Then, it was discussed that, in practice, instead of trying to

approximate the $t+1$-dimensional integral above in one-go, it was easier to design a sequential strategy that recursively approximate the integral.

We had that we can approximate $p(y_t|y_{1:t-1})$ with

$$\hat{p}(y_t|y_{1:t-1}) = \sum_{i=1}^{N} \frac{p(y_t|x_t^i)p(x_t^i|x_{t-1}^i)}{h(x_t^i|x_{0:t-1}^i, y_{1:t})} \tilde{w}_{t-1}^i, \qquad i = 1, ..., N$$

Then I mentioned that, except for the simplest models, it is often difficult to “construct” an importance function $h(\cdot|x_{0:t-1}, y_{0:t})$ that is able to “look ahead” and see the next observation y_t, we simply take

$$h(x_t|x_{0:t-1}, y_{0:t}) = p(x_t|x_{t-1})$$

the transition density, that we always know how to sample from by running the latent model forward (even if we have no idea of which specific distribution it is).

Thanks to this simplification (which has downsides, as we will see) we get

$$\hat{p}(y_t|y_{1:t-1}) = \sum_{i=1}^{N} p(y_t|x_t^i)\tilde{w}_{t-1}^i, \qquad i = 1, ..., N$$

or, by setting $w_t^i = p(y_t|x_t^i)$ and $\tilde{w}_t^i = w_t^i / \sum_{i=1}^{N} w_t^i$,

$$\hat{p}(y_t|y_{1:t-1}) = \sum_{i=1}^{N} w_t^i \tilde{w}_{t-1}^i, \qquad i = 1, ..., N$$

(simplified) Sequential importance sampling

Then we obtained the following sequential algorithm where we "propagate forward" from the transition density:

1. $t = 0$ (initialize) $x_0^i \sim p(x_0)$, assign $\tilde{w}_0^i = 1/N$, $i = 1, ..., N$
2. at the current t assume we have the particles x_t^i
3. From your model *propagate forward* $x_{t+1}^i \sim p(x_{t+1}|x_t^i)$, $i = 1, ..., N$.
4. Compute (unnormalised weights)

$$w_{t+1}^i \propto p(y_{t+1}|x_{t+1}^i) \times \tilde{w}_t^i.$$

5. we can finally approximate (Creal, p. 253 $\longleftarrow$ hyperlink)

$$\hat{p}(y_{t+1}|y_{1:t}) = \sum_{i=1}^{N} w_{t+1}^i \tilde{w}_t^i$$

   and normalise weights $\tilde{w}_{t+1}^i = w_{t+1}^i / \sum_{i=1}^N w_{t+1}^i$

6. set $t := t + 1$ and if $t < T$ go to step 2.

We found out that even for a simple example the approximation was poor, even when using N very large.

Let's see what happened for the usual model

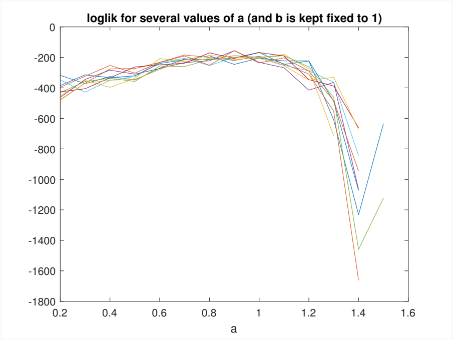
usual example

In `demo_sis.m` we use the following model:

$$\begin{cases} y_t = b \cdot x_t + \epsilon_t^{(1)}, & \epsilon_t^{(1)} \sim_{iid} N(0, 0.3^2) \\ x_t = a \cdot x_{t-1} + \epsilon_t^{(2)}, & \epsilon_t^{(2)} \sim_{iid} N(0, 1) \\ x_0 = 0 \end{cases}$$

and in the previous lecture, just for illustration, we set $b = 1$ constant to the true value that generated data and let a vary on a grid of values.

Ten runs with N=10000



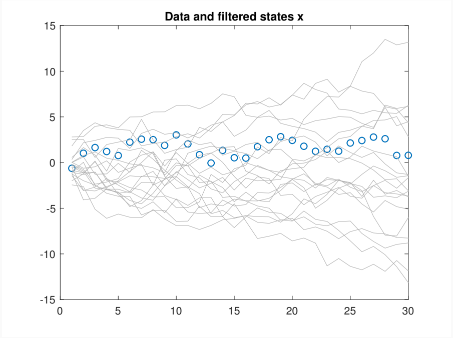
The data-generating value was $a = 1$, but the likelihood approximation does not seem very informative about the optimal value of a.

Notice: sometimes observed data are not informative enough (eg too small dataset). But it is NOT the case here, it is something else.

How do the simulated x values look?

[notice I uploaded a new, slightly modified SIS file `demo_sis_with_states.m`]

I collect the simulated `x` values and they look like (here N=20 just to make things not too messy to read)



Clearly the x values go allover the place and won't get much weight in $p(y_t|x_t) = N(x_t, 0.3^2)$. So the likelihood is badly approximated.

The trick will be to **not allow particles that have low weight** w_t **at time** t **to keep existing at time** $t+1$.

We want to kill those state values (particles) that seem improbable.

An in fact, if particle x_{t-1}^i has low weight w_{t-1}^i, then

$$\tilde{w}_{t-1}^i = w_{t-1}^i / \sum_i w_{t-1}^i$$

will become small and this particle will be forever doomed, see next slide.

It is not unusual that a particle gets an exactly zero weight, since your computer sets to zero a very small yet in principle strictly positive w_t. This is called *numerical underflow*, and that particle is doomed! Since

$$w_t^i \propto \frac{p(y_t|x_t^i)p(x_t^i|x_{t-1}^i)}{h(x_t^i|x_{0:t-1}^i, y_{1:t})}\tilde{w}_{t-1}^i.$$

if for a given i we have $\tilde{w}_{t-1}^i = 0$, then particle i will get zero weight for *all subsequent times*.

Example of underflow

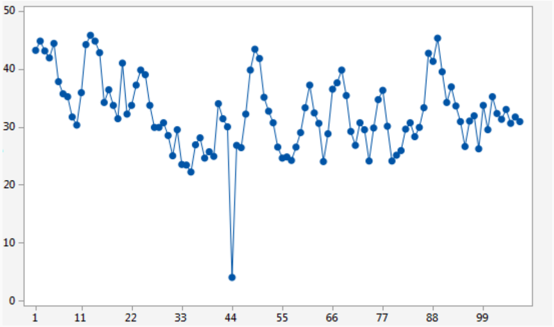
example, say that you want to evaluate particle value $x_t^i = 40$ in $p(y_t|x_t^i) = N(x_t^i; 0, 1)$

```
dnorm(40,0,1)
[1] 0
```

and even if you work on the log-density domain

```
> logw=dnorm(40,0,1,log=TRUE)
> exp(logw)
[1] 0
```

Also if we all of a sudden get an outlier, this could potentially underflow most (all?) weights, say we have this dataset



So, as I said we really need to get rid of unpromising particles and let them die ($\rightarrow$ no propagation forward from those ones).

The key idea is to do resampling with replacement according to normalised weights.

The Resampling idea

A life saving solution is to use *resampling with replacement*.

Say that we are at time t and obtained the particles $x_t^1, ..., x_t^N$ and their unnormalised weights $w_t^1, ..., w_t^N$.

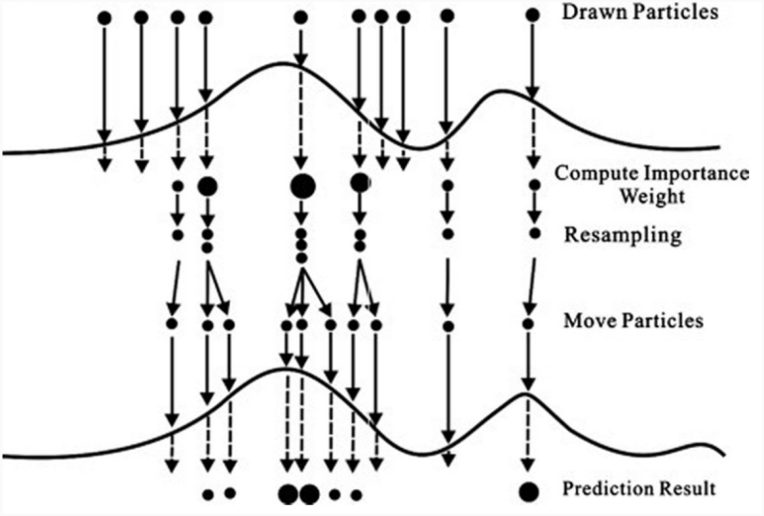
1. (normalization) set $\tilde{w}_t^i := w_t^i / \sum_{i=1}^N w_t^i$, for every $i = 1, ..., N$. Clearly $\tilde{w}_t^i \in (0, 1)$.
   Interpret $\tilde{w}_t^i$ as the probability attached to x_t^i in the **weighted set** $\{x_t^i, \tilde{w}_t^i, i = 1, ..., N\}$.
2. draw N particles with replacement from the weighted set. Call the drawn particles $\{\tilde{x}_t^1, ..., \tilde{x}_t^N\}$ and replace the original ones with $\{\tilde{x}_t^1, ..., \tilde{x}_t^N\}$.
3. After resampling, consider the new particles as all having the same importance, that is give them all $w_t^i = 1/N$.
4. Now propagate forward each of the particles $\tilde{x}_t^i \to x_{t+1}$ as usual by running your model.

The key part was that in the last step we wrote

$$\tilde{x}_t^i \to x_{t+1}.$$

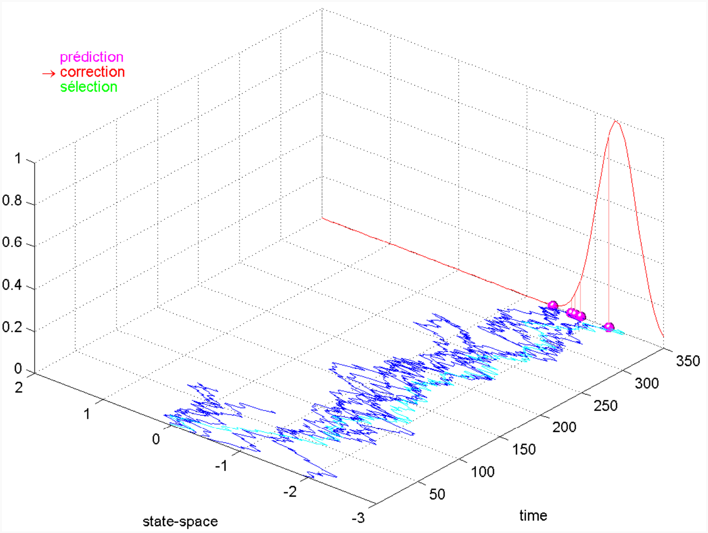
This means that **we propagate forward only the resampled particles**. The others peacefully **die out!**

Propagation→weighting→resampling→propagation of resampled particles→weighting→etc



The next animation illustrates the concept of sequential importance sampling resampling with $N = 5$ particles.

- Cyan: observed trajectory (data)
- dark blue: simulation of the latent process $\{X_t\}$
- pink balls: particles x_t^i
- green balls: selected particles $\tilde{x}_t^i$ from resampling
- red curves: density $p(y_t|x_t)$



prédiction
→ correction
sélection
1
0.8
0.6
0.4
0.2
0
2
1
0
-1
-2
-3
state-space
50
100
150
200
250
300
350
time

Bootstrap filter for likelihood approximation

This is also aptly named *SIS with resampling*:

1. $t = 0$ (initialize) $x_0^i \sim p(x_0)$, assign $\tilde{w}_0^i = 1/N$, for $i = 1, ..., N$
2. at the current t we have the weighted set $\{x_t^i, \tilde{w}_t^i\}_{i=1}^N$.
3. From $\{x_t^i, \tilde{w}_t^i\}_{i=1}^N$ resample particles with replacement N times, to obtain $\{\tilde{x}_t^i\}_{i=1}^N$. Then reset $\tilde{w}_t^i = 1/N$ for $i = 1, ..., N$.
4. For resampled particles: *propagate forward* $x_{t+1}^i \sim p(x_{t+1}|\tilde{x}_t^i)$, $i = 1, ..., N$.
5. Compute (unnormalised weights)

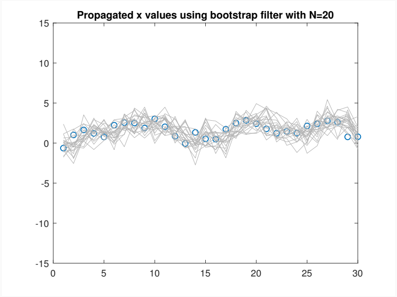
$$w_{t+1}^i \propto p(y_{t+1}|x_{t+1}^i) \times \tilde{w}_t^i = p(y_{t+1}|x_{t+1}^i) \times 1/N.$$

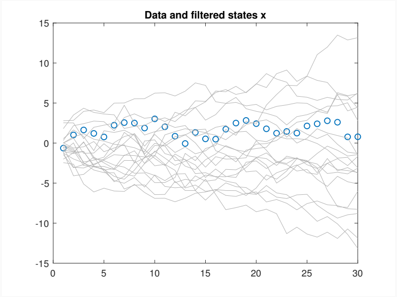
6. approximate the likelihood term as usual

$$\hat{p}(y_{t+1}|y_{1:t}) = \sum_{i=1}^N w_{t+1}^i \tilde{w}_t^i = \sum_{i=1}^N w_{t+1}^i / N$$

The usual example

Let’s look at how the propagated trajectories look, with as little as $N = 20$ with bootstrap filter (left) and SIS (right).





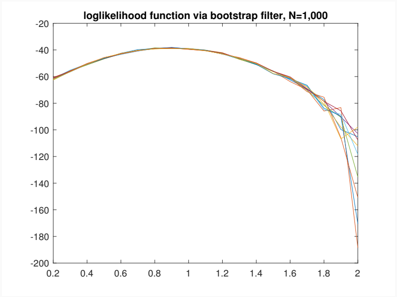
Of course we could use a larger N but visually the plots would look to dense.

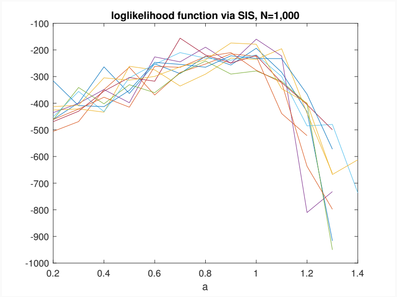
Let’s now look at the loglikelihood approximation.

LogLikelihood approximations for varying a and fixed b

Same as we did with SIS: let a vary and b is kept fixed to truth $b = 1$. Here $N = 1,000$, and we approximate the loglikelihood functions across 10 independent runs.

We use bootstrap filter (left) and SIS (right).





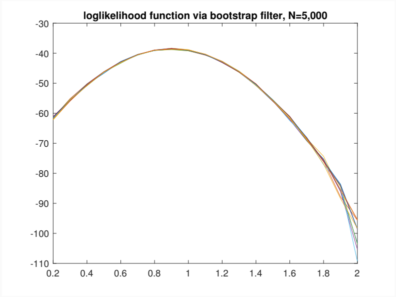
Notice the y-axis are on **very different scales**.

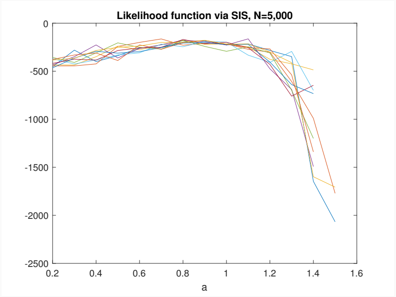
Also notice SIS is unable to return values for $a > 1.4$.

LogLikelihood approximations for varying a and fixed b

Now use $N = 5,000$.

We use bootstrap filter (left) and SIS (right).





Notice the y-axis are on **very different scales**.

We have finally found a criterion that clearly displays a maximizer around $a = 1$ as wanted.

So the bootstrap filter by Gordon et al. (1993)[1] easily provides what we need!

- $\hat{p}(y_t|y_{1:t-1};\theta) = \frac{1}{N}\sum_{i=1}^{N} w_t^i$
- Finally a **likelihood approximation**:

$$\hat{p}(y_{1:T}|\theta) = \hat{p}(y_1)\prod_{t=2}^{T}\hat{p}(y_t|y_{1:t-1};\theta)$$

We could use it for:

- **approximate** maximum likelihood

$$\theta_{mle} = \text{argmax}_\theta \hat{p}(y_{1:T};\theta)$$

  or

- **Bayesian inference** by plugging $\hat{p}(y_{1:T}|\theta)$ inside Metropolis-Hastings.

[1] Gordon, Salmond and Smith. IEEE Proceedings F. 140(2) 1993.

Resampling particles using some software

To resample particles you can make use of built-in routines. Always remember that in this context we wish to sample **with replacement**. Below `normw` denotes the normalised weights $\tilde{w}$ at a given time and `xres` is the vector of resampled particles obtained from the current `x`.

- In R you can use `xres <- sample(x, size = N, replace = TRUE, prob = normw)`
- Or, again in R, I believe the most efficient way is to first sample *indeces* of the particles via `index <- sample.int(N, size = N, replace = TRUE, prob = normw)`

  and then create `xres <- x[,index]` (assuming you created `x` to be a matrix with N columns).

Sampling parameters from the posterior

Now we focus on obtaining parameter inference in a Bayesian way, that is obtain uncertainty quantification by sampling from the posterior distribution.

What we did so far was looking at some plots of the likelihood, but that's unsatisfactory because we kept one parameter fixed and let another one vary: this is too empirical.

Luckily, you have already seen the tool that we need to use. It is MCMC via Metropolis-Hastings.

Let's refresh our memory.

Metropolis-Hastings

We wish to sample from the posterior $\pi(\theta|y_{1:T}) \propto p(y_{1:T}|\theta) \times \pi(\theta)$.

However in practice we can only sample from

$$\hat{\pi}(\theta|y_{1:T}) \propto \hat{p}(y_{1:T}|\theta) \times \pi(\theta),$$

with $\hat{p}(y_{1:T}|\theta)$ approximated via particle filters, eg via bootstrap filter.

Since we embed a particle filter inside an MCMC algorithm, this strategy is often called particle MCMC[2].

For our usual example, $\theta = (a, b)$ and we need to specify priors for a and b.

[2] An introduction is in Dahlin-Schön. The original, way more technical, paper is Andrieu et al. here.

Metropolis-Hastings (“particle MCMC” in this context)

Initialization: Set a starting value $\theta_1 = \theta^*$, eg the mean of the priors of a and b. Set N and R the number of iterations for Metropolis-Hastings. Compute the initial $\hat{p}(y_{1:T}|\theta^*)$. Define a proposal $g(\theta'|\theta)$. Set $r = 1$.

1. Propose a move $\theta^\# \sim g(\theta|\theta^*)$ and run the bootstrap filter using $\theta^\#$ to obtain $\hat{p}(y_{1:T}|\theta^\#)$.
2. Generate a uniform random draw $u \sim U(0, 1)$, and calculate the acceptance probability

$$\alpha = \min\left[1, \frac{\hat{p}(y_{1:T}|\theta^\#)}{\hat{p}(y_{1:T}|\theta^*)} \times \frac{g(\theta^*|\theta^\#)}{g(\theta^\#|\theta^*)} \times \frac{\pi(\theta^\#)}{\pi(\theta^*)}\right].$$

   If $u > \alpha$, set $\theta_{r+1} := \theta_r$ otherwise set $\theta_{r+1} := \theta^\#$, $\theta^* := \theta^\#$ and $\hat{p}(y_{1:T}|\theta^*) := \hat{p}(y_{1:T}|\theta^\#)$. Set $r := r + 1$ and go to step 3.
3. Repeat steps 1–2 as long as $r \leqslant R$.

The resulting sequence $\theta_1, ..., \theta_R$ (possibly after having discarded some initial burnin iterations) is a Markov chain having $\hat{\pi}(\theta | y_{1:T})$ as stationary distribution.

Bayesian inference for a and b

The following is coded in `demo_pmcmc.m`.

I made the following assumptions but this is just for illustration, you are free to change this.

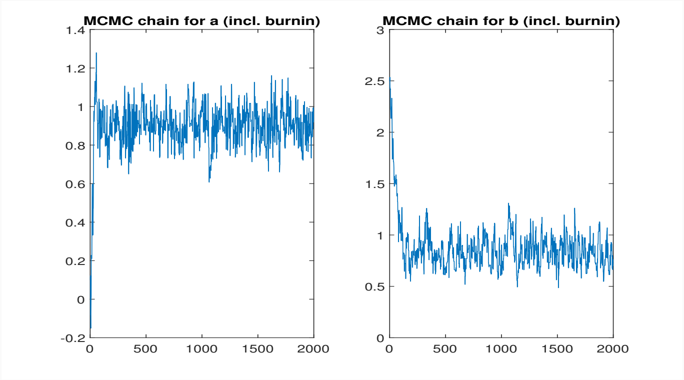
- Joint prior for $\theta = (a, b)$ given by $\pi(\theta) = \pi(a)\pi(b)$ (a and b a-priori independent) with $\pi(a) = N(0.5, 1)$, $\pi(b) = N(1.5, 0.5^2)$.
- $g(\theta|\theta^*) = MVN(\theta^*, \Sigma)$, a multivariate Gaussian with mean θ^* and diagonal covariance matrix

$$\Sigma = \begin{bmatrix} 0.1^2 & 0 \\ 0 & 0.1^2 \end{bmatrix}$$

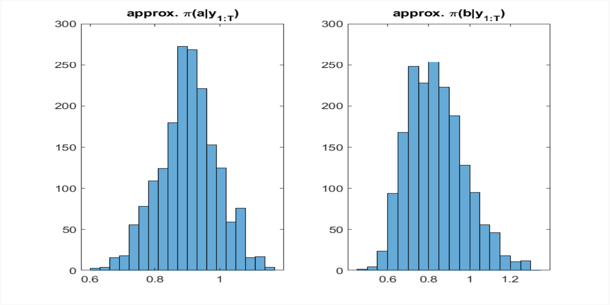
Notice the chosen $g(\cdot|\cdot)$ is **symmetric**, eg $g(\theta^{\#}|\theta^*) = g(\theta^*|\theta^{\#})$, so it simplifies out in α (no need to code it in the ratio).

Recall the used data have been generated with $a = b = 1$ so we hope to recover these values to some extent.

These results used $N = 1,000$ particles, $R = 2000$ MCMC iterations. I used a starting $\theta_1 = (a_1 = 0.1, b_1 = 2.5)$.



The marginal posteriors below are produced by disregarding the first 200 iterations (burnin).



True values $a = b = 1$ are likely in the posterior (which is good) but we also have some uncertainty.

If you didn't know that in reality true values are a=b=1, what you would concluded is that, conditional to observed data, the true value of a should be with high probability somewhere between 0.6 and 1.1, while for b is somewhere between 0.6 and 1.2.

Would you have expected less variability? You can do the following:

- When constructing the experiment, try to get more data (posterior converges to truth as data size grows to infinity).
- For real studies: learn from previous literature and previous experiment (or construct experts), so you can encode a more informative prior that could have some effect on the posterior.
- The results are depending on the quality of the model you design: if the model is inappropriate, inferences will be flawed. But we never know the true model generating the data (except in simulation studies like the one we did).

Tips for Metropolis-Hastings

As usual, best to code things on log-scale for numerical stability.

In `demo_pmcmc.m` I did the following: **instead of coding** (notice I simplified-out the ratio of symmetric proposal densities)

$$\alpha = \min\left[1, \frac{\hat{p}(y_{1:T}|\theta^{\#})}{\hat{p}(y_{1:T}|\theta^{*})} \times \frac{\pi(\theta^{\#})}{\pi(\theta^{*})}\right].$$

If $u > \alpha$, set $\theta_{r+1} := \theta_r$...

I coded

$$\log \alpha = \min\left[0, \log \hat{p}(y_{1:T}|\theta^{\#}) - \log \hat{p}(y_{1:T}|\theta^{*}) + \log \pi(\theta^{\#}) - \log \pi(\theta^{*})\right].$$

If $\log u > \log \alpha$, set $\theta_{r+1} := \theta_r$...etc.

The latter is a completely equivalent but safer approach.

(optional for those interested: more Metropolis-Hastings tips here.)

Canvas uploads

You find on Canvas:

- `demo_sis_with_states.m`: this is a better version of `demo_sis.m`. You can delete the latter. The new version is also able to plot the `x` states, so it is more useful.
- `demo_bootstrap.m` which illustrates the bootstrap filter.
- `demo_pmcmc.m` which illustrates particle MCMC, i.e. produces Bayesian inference via Metropolis-Hastings + bootstrap particle filter.
- `demo_nimbleSMC.R` which illustrates the bootstrap particle filter in R using nimbleSMC, without specifying parameters a and b.
- `demo_nimbleSMC_with_parameters.R` same as above, but here you can easily provide values for parameters a and b.

A possible exercise

- If you did last week's exercise (constructing the SIS filter in R), building the bootstrap filter will be a trivial modification of the SIS filter. Of course you can look at the uploaded Matlab version for inspiration.
- Once the above is done, you may build your own particle MCMC sampler for a and b, for example using the same setup I used, or a different one, and obtain draws from $\hat{\pi}(a|y_{1:T})$ and $\hat{\pi}(b|y_{1:T})$.
- What happens if you start at very unlikely (extreme) values of a and b? Do you observe a lot of rejections? If yes, can you repair this by increasing N? Any intuition why N could have anything to do with MCMC rejections?
- To verify that your custom R code for particle MCMC worked as expected, you may also compare against a version using loglikelihoods obtained via the bootstrap filter as in nimbleSMC (as illustrated in `demo_nimbleSMC_with_parameters.R`), at any given proposed parameter value.