State-space models and particle filters

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

Umberto Picchini
@uPicchini, picchini@chalmers.se

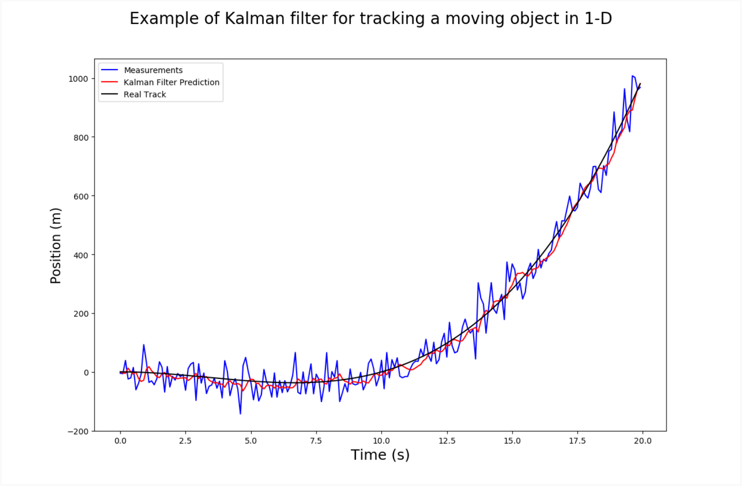
Chalmers University of Technology and University of Gothenburg
Sweden

Overview

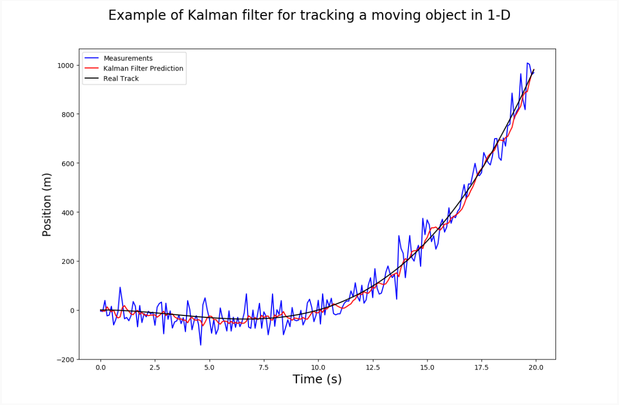
- Today we turn to models with with a “time-structure”: At each time point, the structure of the stochastic model is the same, but variables change over time.
- We consider the case where the model produces observations Y, considered as a “noisy” version of what we are ideally interested in, denoted by X.
- We will look at some theory and algorithmic approaches.

It is often the case that what we observe a phenomenon and record data: say data are denoted $y_1, ..., y_t, ...y_T$.

Say that these data are in blue below:



Imagine hat the data are of type $y_t = x_t + \epsilon_t$, for ϵ a zero-mean random variable.
Think of $(x_t)_{t\geqslant 0}$ as the black line, which is the true yet **unknown** state of a system X at time t.

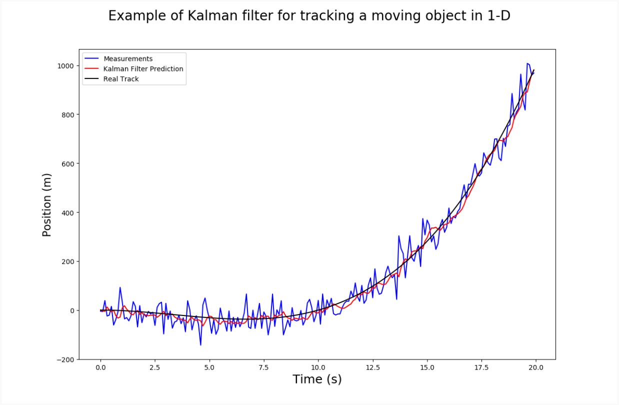


Assume we cannot really observe X exactly, i.e. *the true X is* ***hidden (latent)***

Some terminology

- We have data (blue) that we have **observed** and denote these with the letter y_t at time t.
- y_t is considered as a “noisy” measurement of some **latent** x_t, which we cannot directly observe exactly.
- Notation: when we write $z_{1:t}$ this means the sequence $z_{1:t} = (z_1, ..., z_t)$.

However, despite X being **hidden** (unavailable) we can make use of the actual observations (blue) $y_{1:T} = (y_1, ..., y_T)$ to learn something about X.



In red you see the result of a specific type of **filter**, the Kalman filter. While the filter can only access the data (not X), it manages to filter-out some of the noise and give us back a not too shabby approximation of X.

A very simple example

This is the model implemented in `demo_sis.m` and `demo_nimbleSMC.R`.

Consider for $t = 1, 2, ..., 30$

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

The first equation implies that $p(y_t|x_t) = N(x_t, 0.3^2)$.
The second equation implies that $p(x_t|x_{t-1}) = N(x_{t-1}, 1)$.

But we may also assume unknown parameters such as

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

and be interested in inferring $\theta = (a, b)$ for given data $y_1, ..., y_T$.

Here $p(y_t|x_t; a) = N(a \cdot x_t, 0.3^2)$ and $p(x_t|x_{t-1}; b) = N(b \cdot x_{t-1}, 1)$.

What are we generally interested in?

Things we can learn from data:

- We can learn the distribution of model parameters θ for given data $y_1, \ldots, y_T$;
- We can learn the distribution of the latent x_t given data $y_1, \ldots, y_t$ available up to t, that is

$$p(x_t | y_1, \ldots, y_t; \theta), \qquad \text{this is the filtering distribution}$$

- We can learn the distribution of the latent x_t given all available data $y_1, \ldots, y_T$ available up to final time T

$$p(x_t | y_1, \ldots, y_T; \theta), \qquad \text{this is the smoothing distribution}$$

- We can learn the distribution of the latent x_{t+k} $(k = 1, 2, \ldots)$ given $y_1, \ldots, y_t$, that is

$$p(x_{t+k} | y_1, \ldots, y_t; \theta), \qquad \text{this is the predictive distribution at lag } k$$

There are very many examples where what we want to learn (X) is not directly observable, but a noisy version of X (denoted with Y) is what we can measure.

- Your exact position (X) on a landscape is not possible to obtain, but can be approximated from GPS coordinates (Y).
- The exact number (X) of moose in a given area in a certain month is difficult to get, because animals move around and some leave the area some enter it. So Y is what we measure but it is only an approximation of X.
- The average temperature in Sweden in October 2025 (denote this with $X_{oct,25}$) cannot be predicted exactly. It can only be approximated given *observed* past values in October from previous years (eg by using $Y_{oct,22}, Y_{oct,21}, ..., Y_{oct,18}$).
- etc...

Possible examples are endless!

State-space models

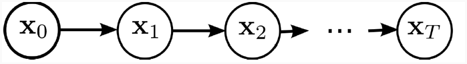
Many important applied problems can be treated by assuming that there is a **Markovian** sequence of *hidden variables* $x_0, x_1, \dots, x_T$.

Markovianity means that, for example

$$p(x_{t+1} \mid x_0, x_1, \dots, x_t) = p(x_{t+1} \mid x_t).$$

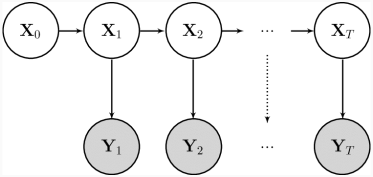
Putting this in words means:
"given the present x_t, the future is independent of the past history $x_0, x_1, \dots, x_{t-1}$".

Which means that every x_{t+1} is exclusively dependent on x_t and on nothing else. In terms of a graph of dependences with can write $x_t \rightarrow x_{t+1}$.



State space models

- But then, we said that we actually measure $y_1, ..., y_T$



- The above means that measurement y_t results as a measurement of x_t **and nothing else**.
- The previous statement implies *conditional independence of observations*: the current measurement y_t given x_t is conditionally independent of the other measurements and state histories

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

or equivalently: given the value of x_t, the variables $y_t, \ldots, y_T$ are independent of variables $y_1, \ldots, y_{t-1}$.

- We will only consider *homogeneous* Markov chains: The variables x_t are of the same type, and the conditional distributions $p(x_t \mid x_{t-1})$ are all the same.
- We will also assume that the $p(y_t \mid x_t)$[1] are the same for all t (and the variables y_t are of the same type).

[1] These are called *emission distributions* in some literature.

State space models

In summary: a **state-space model** is characterised by

1. Markovianity of the latent (hidden) state X;
2. conditionally independent measurements Y.

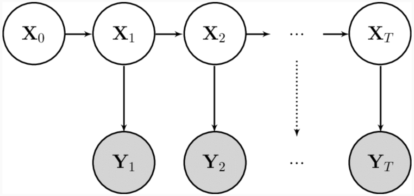
The simple model

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

is a state-space model.

Clearly x_t only depends on x_{t-1}, not x_{t-k} ($k > 1$) or anything else, so it is Markovian.
Then y_t only depends on x_t and since the $\epsilon_t^{(1)}$ are all assumed independent, for given x_t all measurements are (conditionally) independent.

State space models



- Thus, to specify such a state space model we need to specify

$$p(x_0) \qquad p(x_t \mid x_{t-1}) \qquad p(y_t \mid x_t)$$

- The random variables x_t and y_t may be of any type, and may be vectors!
- When x_t are discrete variables with a finite number of possible values, we call the above a *Hidden Markov Model* (HMM).
- If the variables are all (multivariate) normal, and if the dependencies $p(x_t \mid x_{t-1})$ and $p(y_t \mid x_t)$ are linear, we may call the above a *dynamical linear model*.

So many definitions:

- State-space models..
- hidden Markov models...
- dynamical linear model...

Does it really matter to remember these? Not really! Just remember the property of conditional independence of the observations and Markovianity of the latent state.

From now on we only use the term **state-space model**, because it is the one that seems the most persistent in literature.

As I said, we can do many things, parameter estimation, filtering, smoothing,...

However no time to do everything. We start with the important problem of **estimating the likelihood function** for parameters θ.

The likelihood function for SSMs

- In a state-space model (SSM) data are not independent, they are only *conditionally independent*$\rightarrow$ *complication!*:

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

Except for the simplest cases, we generally don't have a closed-form expression for the product above because we do not know how to calculate $p(y_t|y_{1:t-1},\theta)$.

Exceptions are for example linear/Gaussian models where Kalman filtering can be applied (important but we have no time to cover this).

In a SSM the observed process is assumed to depend on the latent Markov process $\{X_t\}$: we can write

$$p(y_{1:T}|\theta) = \int p(y_{1:T}, x_{0:T}|\theta) dx_{0:T} = \int \underbrace{p(y_{1:T}|x_{0:T}, \theta)}_{\text{use cond. indep.}} \times \underbrace{p(x_{0:T}|\theta)}_{\text{use Markovianity}} dx_{0:T}$$

$$= \int \prod_{t=1}^{T} p(y_t|x_t, \theta) \times \left\{ p(x_0|\theta) \prod_{t=1}^{T} p(x_t|x_{t-1}, \theta) \right\} dx_{0:T}$$

Problems

- The expression above is a $(T+1)$-dimensional integral ☹
- For most (nontrivial) models, transition densities $p(x_t|x_{t-1}; \cdot)$ are **unknown** ☹

General Monte Carlo integration

Another equivalent way to write the likelihood function:

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

We can write $p(y_t|y_{1:t-1},\theta)$ as

$$p(y_t|y_{1:t-1},\theta) = \int p(y_t|x_t,\theta)p(x_t|y_{1:t-1},\theta)dx_t = \mathbb{E}(p(y_t|x_t,\theta))$$

Use Monte Carlo integration: generate N draws from $p(x_t|y_{1:t-1},\theta)$, then invoke the law of large numbers.

- produce N independent draws $x_t^i \sim p(x_t|y_{1:t-1},\theta), \quad i = 1,...,N$
- for each x_t^i compute $p(y_t|x_t^i,\theta)$
- and by LLN we have

$$\frac{1}{N}\sum_{i=1}^{N} p(y_t|x_t^i,\theta) \to \mathbb{E}(p(y_t|x_t,\theta)), \qquad N \to \infty$$

- error term is $O(N^{-1/2})$ *regardless the dimension of* x_t ☺

SMC and the bootstrap filter

But how to generate "good" draws (*particles*) $x_t^i \sim p(x_t|y_{1:t-1},\theta)$?

Here "good" means that we want particles such that the values of $p(y_t|x_t^i,\theta)$ are not negligible ("explain" a large fraction of the integrand).

For SSM, *sequential Monte Carlo* (SMC) is the winning strategy.

We will NOT give a thorough introduction to SMC methods. We only use a few notions to solve our parameter inference problem.

[the term *particle filters* can be used interchangeably with SMC.]

Importance sampling

(to simplify reading her I remove the dependence on θ).

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

where $h(\cdot)$ is an arbitrary (positive) density function called "importance density". Choose an $h(\cdot)$ "easy to simulate from".

1. simulate N iid samples: $x^i_{0:t} \sim h(x_{0:t}|y_{1:t}), \quad i = 1,...,N$
2. construct importance weights $w^i_t = \frac{p(y_t|x^i_t)p(x^i_t|x^i_{t-1})p(x^i_{0:t-1}|y_{1:t-1})}{h(x^i_{0:t}|y_{1:t})}$
3. $p(y_t|y_{1:t-1}) = \mathbb{E}(p(y_t|x_t)) \approx \frac{1}{N}\sum_{i=1}^N w^i_t$

Importance Sampling

However generating at each time a “cloud of particles” $x^i_{0:t}$ is not really computationally appealing, and it’s not clear how to do so.

Much better to try to split the problem into a *sequential* mechanism, as t increases.

Sequential Importance Sampling

When $h(\cdot)$ is chosen in an intelligent way, an important property is the one that allows *sequential* update of weights. After some derivation , we have (see p. 121-122 in Särkkä[2] and p. 252 in Creal)

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

where the proportionality symbol means that we will not particularly care of the (unknown) proportionality constant, and $\tilde{w}$ denotes **normalised weights**, i.e.

$$\tilde{w}_{t-1}^i = \frac{w_{t-1}^i}{\sum_{i=1}^N w_{t-1}^i}$$

[2] Särkkä, *Bayesian filtering and smoothing*, available here.

However, for brevity, I immediately go to a simpler version, since for most SSM we do not typically know how to specify a $h(x_t^i|x_{0:t-1}^i, y_{1:t})$ that depends on data $y_{1:t}$.

A simple, popular (but sometimes computationally inefficient) possibility is to take

$$h(x_t^i|x_{0:t-1}^i, y_{1:t}) \equiv p(x_t^i|x_{t-1}^i)$$

this is the transition density!
And even if we do not know its expression look what happens:

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

We now have a very simple expression that we can *always* evaluate for ANY SSM.
We only need to know how to simulate from $p(x_t^i|x_{t-1}^i)$ but this can *always* be done.

Example

Recall the simple model

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

suppose noone told you that $p(x_t|x_{t-1}) = N(a \cdot x_{t-1}, 1)$ (though it is evident, but let's pretend we had no clue). But still, you can write a computer code that runs

```
x[t] = a*x[t-1] + runif(1)
```

and the `x[t]` you generated is certainly distributed as $p(x_t|x_{t-1}) = N(a \cdot x_{t-1}, 1)$.

So you can always simulate, even though you may not know the actual transition density.

(simplified) Sequential importance sampling

We can now write a simplified sequential algorithm where we sample from the transition density (I removed conditioning on θ everywhere to ease reading):

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.

Coding

In terms of coding, in step 5 it makes sense to start multiplying the likelihood terms: say that in step 1 you initialize the likelihood as `lik=1`, then in step 5 you code a recursion

$$\texttt{lik} = \texttt{lik} \times \sum_{i=1}^{N} w_{t+1}^{i} \tilde{w}_{t}^{i}$$

so at the end of T iterations the code returns you the likelihood approximation `lik`$= \hat{p}(y_{1:T}|\theta)$.

Even better (often necessary!) code everything on the log-scale, eg initialize a loglikelihood `loglik=0` and then in step 5 update it as

$$\texttt{loglik} = \texttt{loglik} + \log\Big(\sum_{i=1}^{N} w_{t+1}^{i} \tilde{w}_{t}^{i}\Big)$$

A worked-out experiment

An example of sequential importance sampling (SIS) is coded in the Matlab file I put on Canvas, called `demo_sis.m`.

Why Matlab? Because I wanted to provide you with a starting point to understand how do to things, but I still want you to code it in R to fix things in mind, without passively relying on my code.

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

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

(notice here $x_0 = 0$ deterministically) and data (file `yobs.dat`) is a sequence of length $T = 30$ generated with $a = b = 1$.

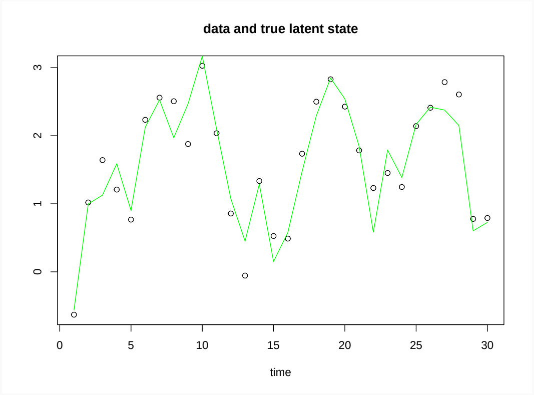


Figure 1: Observed y values (circles) and the true x values (green line).

Notice, consistently with our notation, data start being observed at $t = 1$. However $x_0 = 0$ at $t = 0$ (not depicted in the figure).

I run SIS with $N = 1000$ particles while using $a = b = 1$ and obtain

```
loglik = -78.5547
```

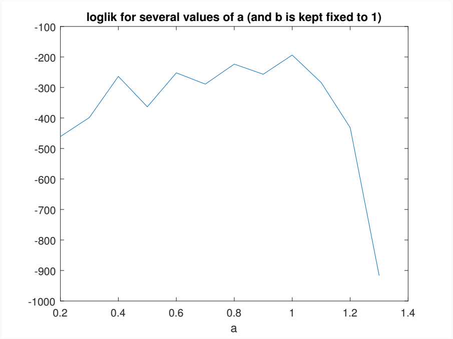
but of course that's just a single run. The approximated loglikelihood is a random variable now (due to Monte Carlo), and if I rerun the filter I obtain something a little different. That's normal!

But is it working?

I compute a bunch of loglikelihoods for $b = 1$ kept fixed and let a change between $a \in [0.2, 2]]$.

What we wish is the loglikelihood to peak around $a = 1$ (truth).

N=1000

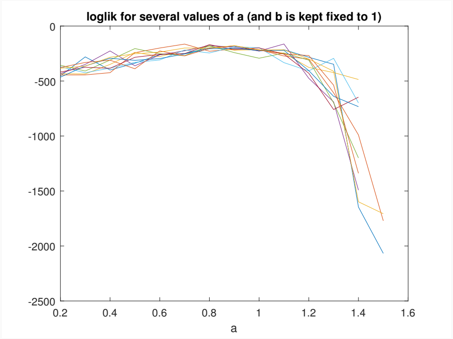


Allright it peaks close to a=1, but if you try multipe times you obtain results not always satisfactory. Also, for $a > 1.3$ everything badly underflows numerically and we get no meaningful result.

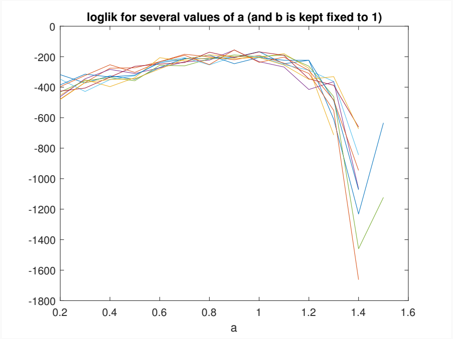
We can do better! (Next lecture).

For now, let's just increase N.

Ten runs with N=5000



Ten runs with N=10000



I mean, this filter barely works and there is a good reason for this (next lecture!). And if it is not great with such a simple model, it turns completely useless with more advanced ones.

There is a very simple modification to make, but a killer one!

An exercise

Look at `demo_sis.m` (*not* `demo_NimbleSMC.R`, that one will be discussed next week). If you can, try to run it, otherwise I hope it is fairly clear to read. Then implement in pure R (no special packages needed) the most important part, the SIS particle filter function that you find from line 50 onward. Write an R script that runs said particle filter with the observations I uploaded on Canvas.

It is good to try to obtain plots similar to those shown in the slides for a=b=1 (remember plots will differ from mine as you cannot reuse my same pseudorandom numbers) and for a grid of several values of a with b fixed to 1.

Show plots.

Try to see what happens for several values of N.

What happens if you set $x_0 = 10$? Any explanation?