Switching Linear Dynamical Systems

Information Theory and Inference

Bacilieri, Barbiero, Bordin, Pitteri - GROUP9

Intuition

Switching state space models

The idea behind switching state space models is quite intuitive and can be summed up as

“The macroscopic behavior of a system depends on hidden states…

One can be seen when driving a vehicle. The macroscopic motion is essentially governed by four basic behaviors (accelerate, brake, turn left, turn right)

Base Model

The layers

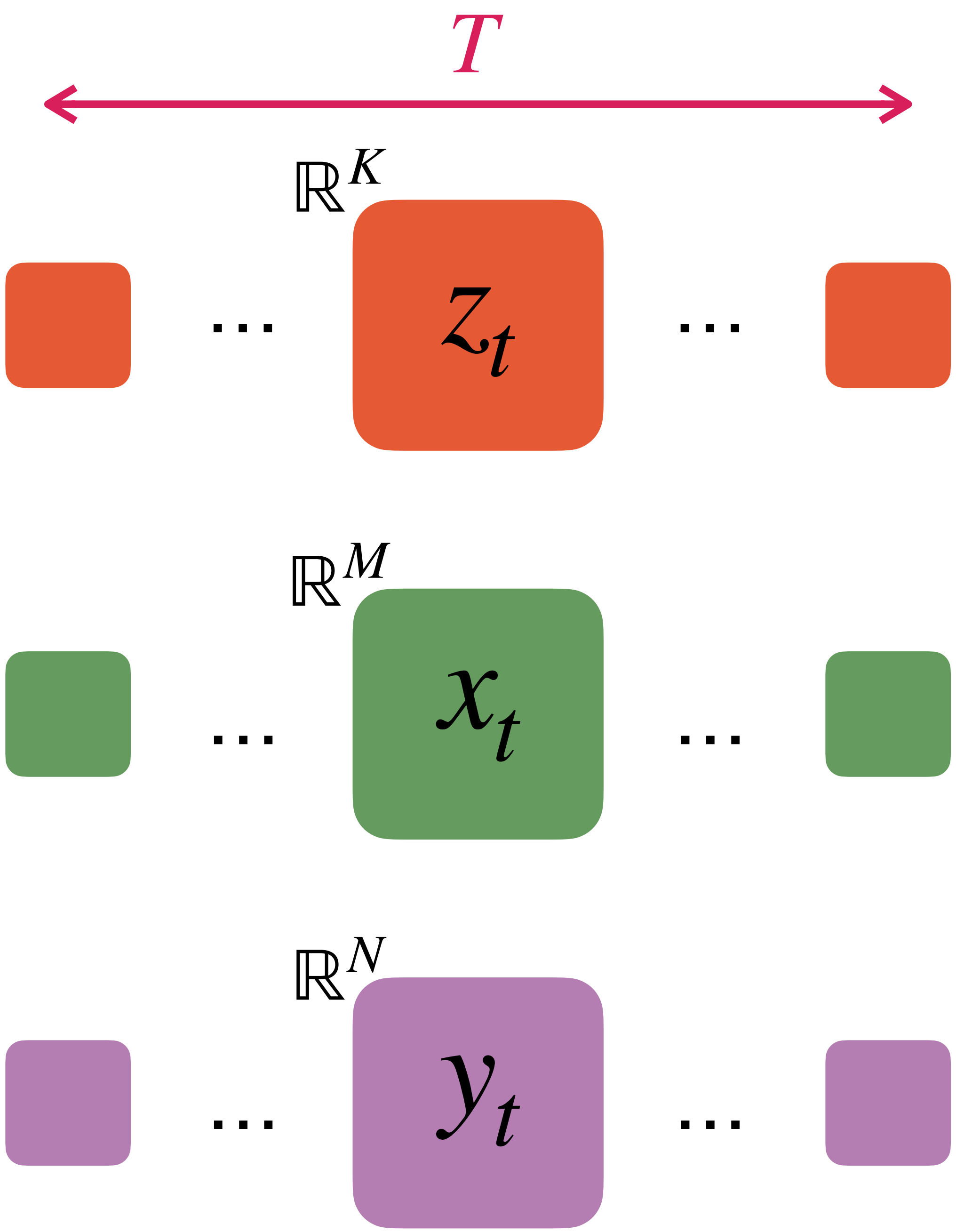
The model is composed of:

K discrete latent states

M continuous latent states

N visible states

Finally, the time step is denoted with $t = 1,\ldots,T$



Base Model

The dynamics

The latent discrete dynamic is Markovian

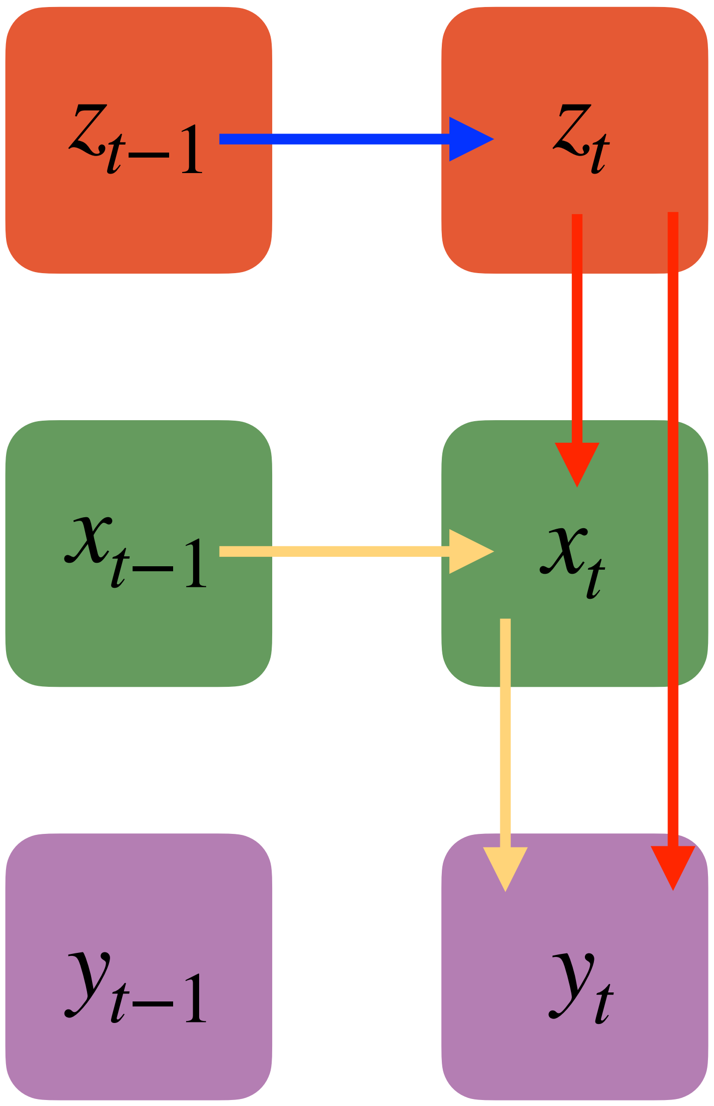
$z_t \sim \pi_{z_{t-1}} \in \mathbb{R}^{K \times K}$

And it is propagated through the other layers as

$x_t = A_{z_t} x_{t-1} + b_{z_t} + v_t$

$y_t = C_{z_t} x_t + d_{z_t} + w_t$

Where ν_t, w_t are Gaussian noises



Base Model

Hidden continuous layer

Remember how

$x_t = A_{z_t} x_{t-1} + b_{z_t} + v_t$ with $v_t \sim \mathcal{N}(0, Q_{z_{t+1}})$

It follows that

$A_k \in \mathbb{R}^{M \times M}$

$b_k \in \mathbb{R}^{M}$

$Q_k \in \mathbb{R}^{M \times M}$

In principle different for any latent state $k = 1, \ldots, K$

Base Model

Visible layer

In a similar way

$y_t = C_{z_t} x_t + d_{z_t} + w_t$ with $w_t \sim \mathcal{N}(0, S_{z_{t+1}})$

It follows that

$C_k \in \mathbb{R}^{N \times M}$

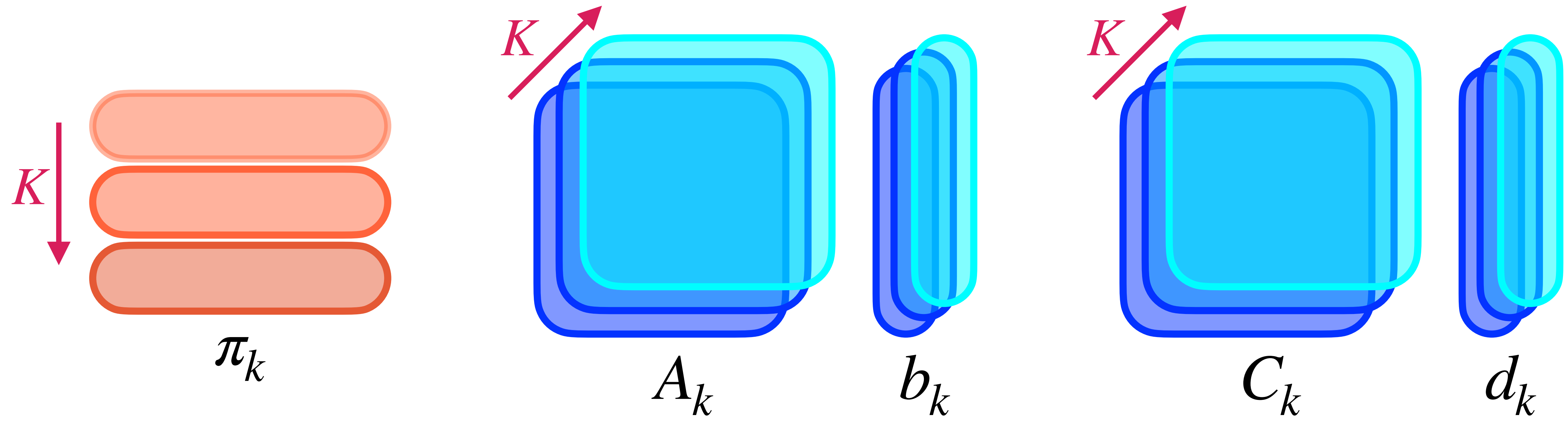
$d_k \in \mathbb{R}^{N}$

$Q_k \in \mathbb{R}^{N \times N}$

Also different for any latent state $k = 1, \ldots, K$

Base Model

Recap



$$\vartheta = [(\underline{\pi_k}, \underline{A_k, Q_k, b_k}, \underline{C_k, S_k, d_k})]_{k=1}^{K}$$

Simplified Model

SMH my head

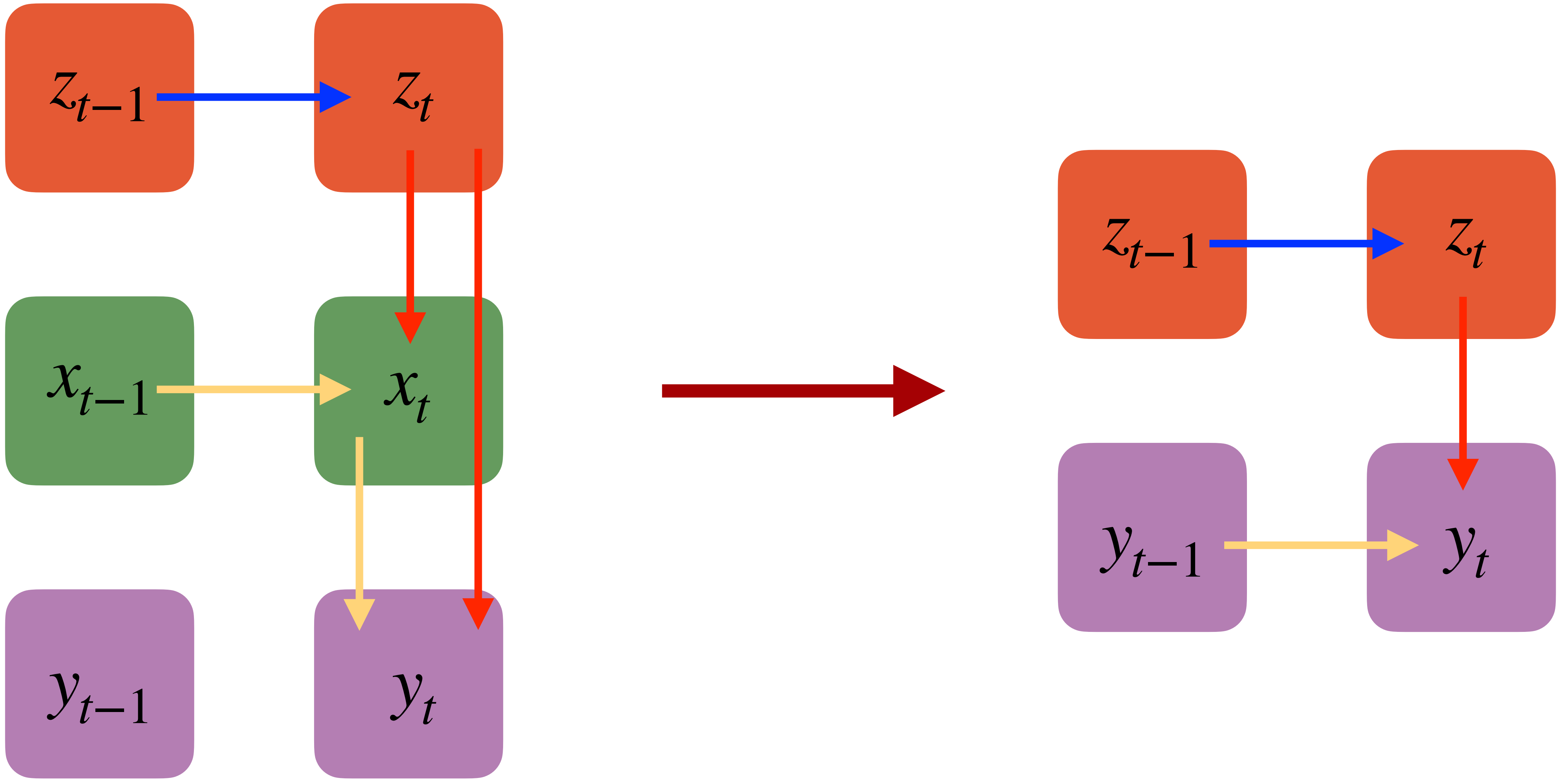
Despite having developed the model from a theoretical standpoint with three layers in practice we only used two of them, the two main reasons were

1 - Explainability

2 - Link to problem complexity

Final Model

Simplified model

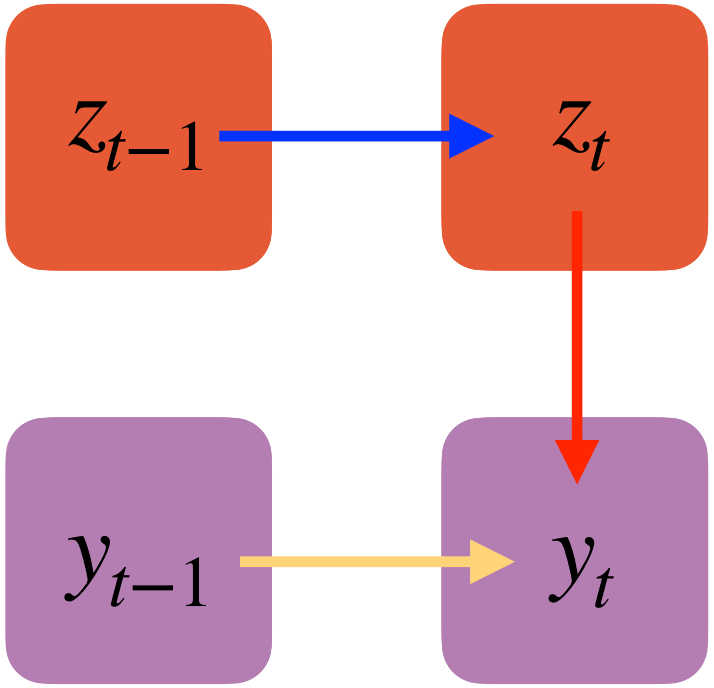


Final Model

Simplified model

$$z_t \sim \pi_{z_{t-1}}$$

$$y_t = A_{z_t} y_{t-1} + b_{z_t} + v_t$$



Numerical Implementation

JAGS vs Stan

Numerical Implementation

Stan - The inference problem

The first approach was setting up an appropriate Hamiltonian Monte Carlo model with Rstan.

Once the model was set, we needed a way to actually retrieve the latent states z. This was performed in two steps

1 - Likelihood estimation with the *Forward Algorithm*

2 - MAP z sequence retrieval with the *Viterbi Algorithm*

Numerical Implementation

Forward algorithm

The forward algorithm has been implemented for both sampling efficiency and a Stan requirement, since discrete parameters are not allowed

The final goal is to compute the joint likelihood

$$p(x_{1:T}, y_{1:T})$$

We can start from

$$\gamma_t(k) = p(z_t = k, x_{1:t}, y_{1:t})$$

Numerical Implementation

Forward algorithm

From

$$\gamma_t(k) = p(z_t = k, x_{1:t}, y_{1:t})$$

We can sum over the hidden states z at $t-1$ and then apply the chain rule to retrieve a recurrent expression

$$\gamma_t(k) = p(y_t \,|\, z_t = k, x_t) p(x_t \,|\, z_t = k, x_{t-1}) \sum_{j=1}^{K} \pi_{jk} \gamma_{t-1}(j)$$

Numerical Implementation

Forward algorithm

Finally

$$\mathscr{L}_k(y_t) = p(y_t \mid z_t = k, x_t) = \mathcal{N}(C_k x_t + d_k, S_k)$$

$$\mathscr{L}_k(x_t) = p(x_t \mid z_t = k, x_{t-1}) = \mathcal{N}(A_k x_{t-1}, b_k, Q_k)$$

And so

$$\gamma_t(k) = \mathscr{L}_k(y_t)\mathscr{L}_k(x_t) \sum_j \pi_{jk} \gamma_{t-1}(j)$$

Numerical Implementation

Forward algorithm

$$\gamma_t(k) = p(z_t = k, x_{1:t}, y_{1:t}) = \mathscr{L}_k(y_t)\mathscr{L}_k(x_t)\Sigma_j \pi_{jk}\gamma_{t-1}(j)$$

For the first step

$$\gamma_1(k) = p(z_t = k, x_1, y_1) = \mathscr{L}_k(y_1)p(x_1 \,|\, z_1 = k)p(z_1 = k)$$

Where

$p(x_1 \,|\, z_1 = k)$ is a multivariate Gaussian

$p(z_1 = k)$ is a uniform distribution

Numerical Implementation

Viterbi algorithm

Most of the steps are similar to the forward algorithm so we won't get in too much detail

$$\eta_t(k) = \arg\max p(z_{1:t-1}, z_t = k, x_{1:t}, y_{1:t})$$

We can retrieve the recursive relation

$$\eta_t(k) = \mathscr{L}_k(y_t)\mathscr{L}_k(x_t)\ \mathrm{argmax}_j[\pi_{jk}\eta_{t-1}(j)]$$

With initialization

$$\eta_1(k) = p(z_1 = k, x_1, y_1)$$

Numerical Implementation

Extracting information

Putting everything together we have

$$p(x_{1:T}, y_{1:T}) = \sum_{k=1}^{K} p(z_T = k, x_{1:T}, y_{1:T}) = \sum_{k=1}^{K} \gamma_T(k)$$

$$\hat{z}_{1:T} = \arg\max_{z_{1:T}} p(z_{1:T}, x_{1:T}, y_{1:T}) = \arg\max_{k} \eta_T(k)$$

Numerical Implementation

JAGS

In JAGS most of the issues that plagued the Stan implementation were, fortunately, absent and as a result the model was easier to code since:

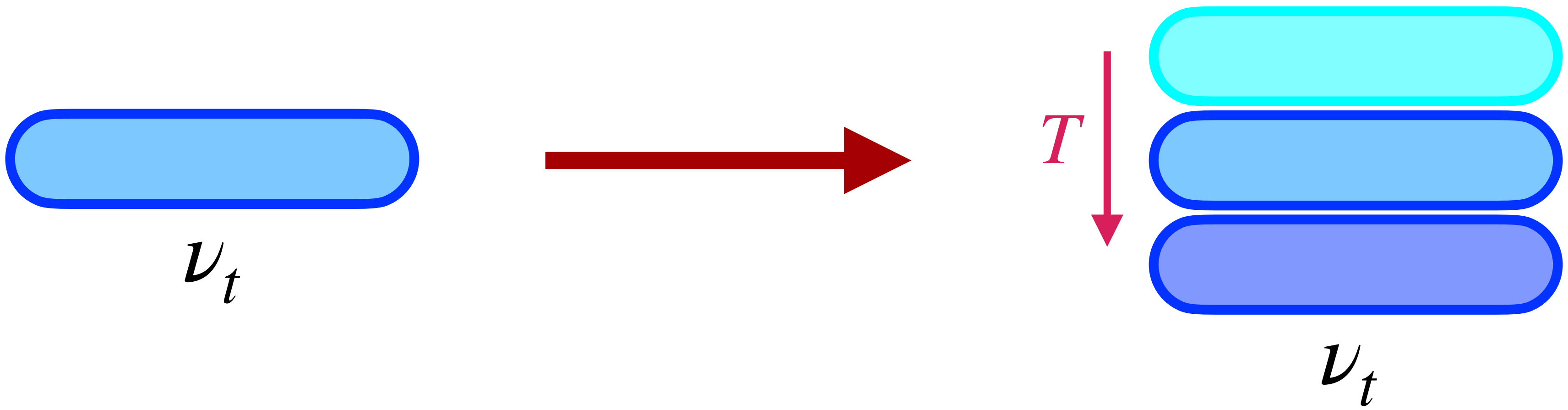
1 - It is enough to specify the stochastic dependance between variables to directly retrieve posterior distributions for all parameters

2 - The Viterbi algorithm wasn't needed thanks to the native support for discrete parameters

Numerical Implementation

JAGS

On the other hand, JAGS does not allow redefining variables, resulting in the need for much more complex, multi-dimensional arrays.



Moreover, some functions are not vectorized

Generated data

Dummy Random Walk

The first dataset we generated to test the system was a polar random walk with two states, also known in the literature as "Markov windshield wiper".

state 1: "move clockwise with step θ"

state 2: "move counter clockwise with step θ"

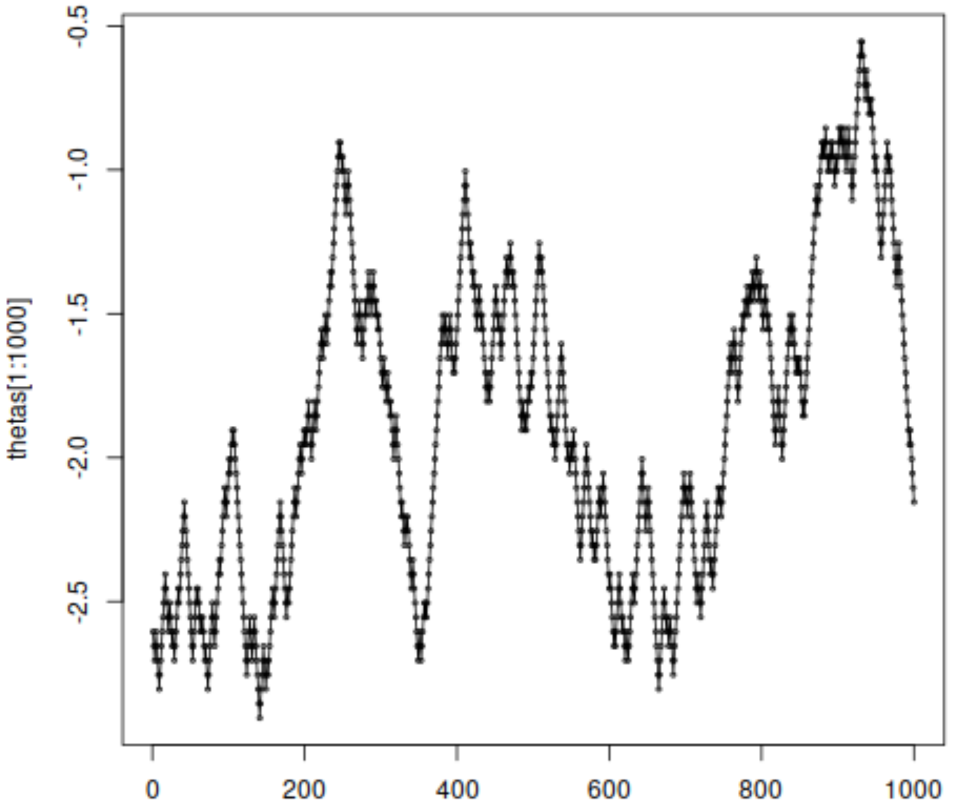
As for the transition matrix it simply was

$$\pi = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$$

Generated data

Results

The dataset was then saved as (x, y) coordinates



Recursive Model

Refinement

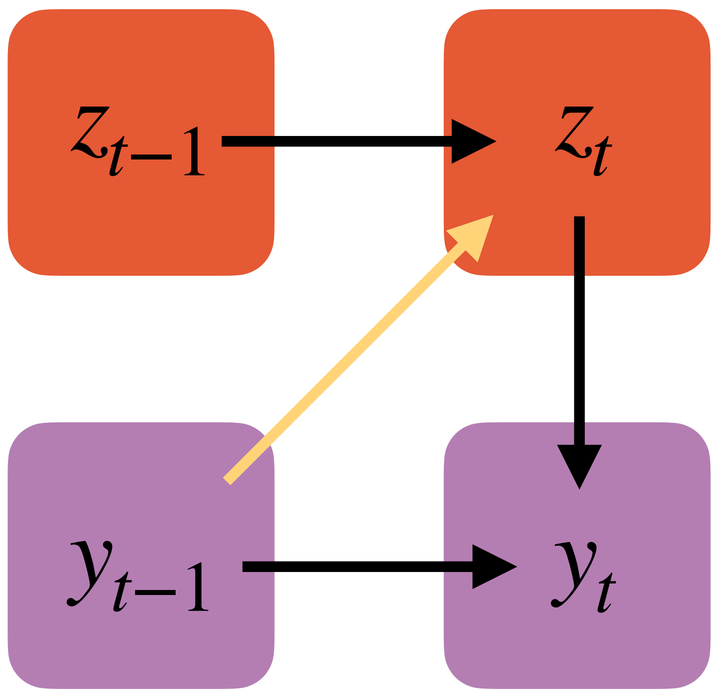
One way to make the model more flexible is to modify the Markov process by introducing a conditional dependence on the previous state

$z_t \sim \pi_{SB}(\nu_t) \quad$ with $\quad \nu_t = R_{z_{t-1}} y_{t-1} + r_{z_{t-1}}$

To have consistent dimensionality

$R_k \in \mathbb{R}^{K-1 \times M}$

$r_k \in \mathbb{R}^{K-1}$



Recursive Model

Stick Breaking

The stick breaking distribution is defined as

$$\pi_{SB}(\nu) = [\pi_{SB}^{(1)}(\nu), \ldots, \pi_{SB}^{(K)}(\nu)]$$

$$\pi_{SB}^{(K)}(\nu) = \begin{cases} \sigma(\nu_k)\prod_{j=1}^{k}(1-\sigma(\nu_j)) & \text{if } k \leq K-1, \\ \prod_{j=1}^{K-1}(1-\sigma(\nu_j)) & \text{if } k = K, \end{cases}$$

Where $\sigma(x)$ is the logistic function

$$\sigma(x) = \frac{e^x}{1+e^x}$$

Recursive Model

Stick Breaking

Both the Forward and Viterbi algorithms can be adapted to accomodate the Sick Breaking update rule, we will also need additional priors.

$$(R_k, r_k) \sim \mathrm{MN}(M_r, \Sigma_r, \Omega_r)$$

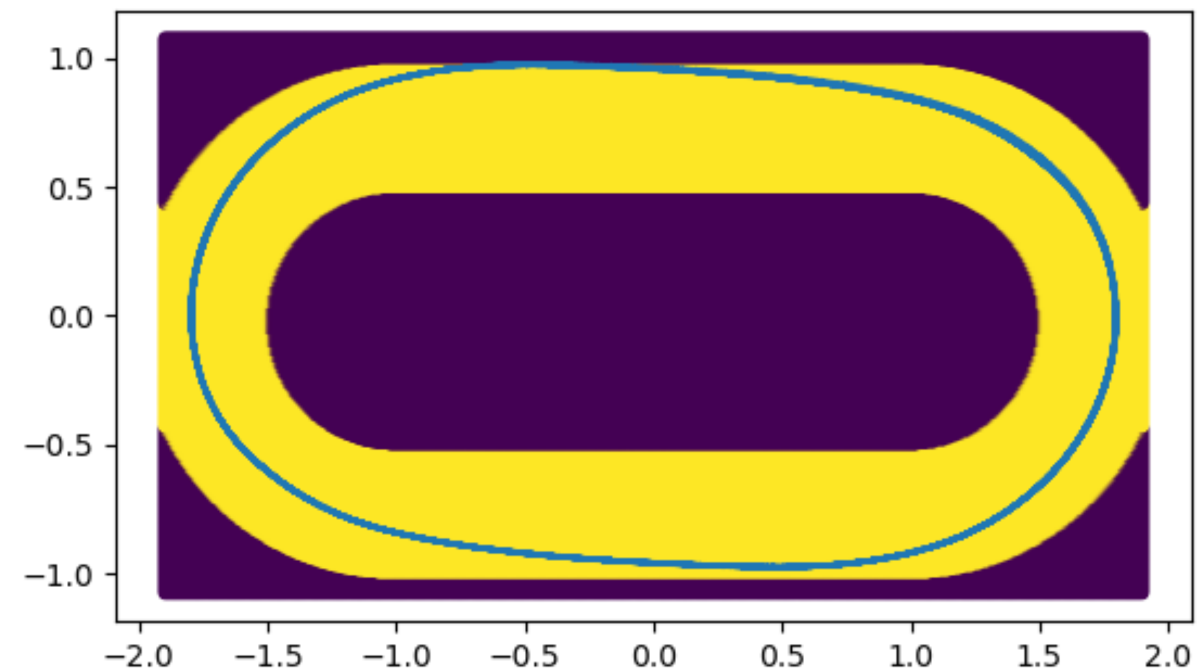
Racing circuit datasets

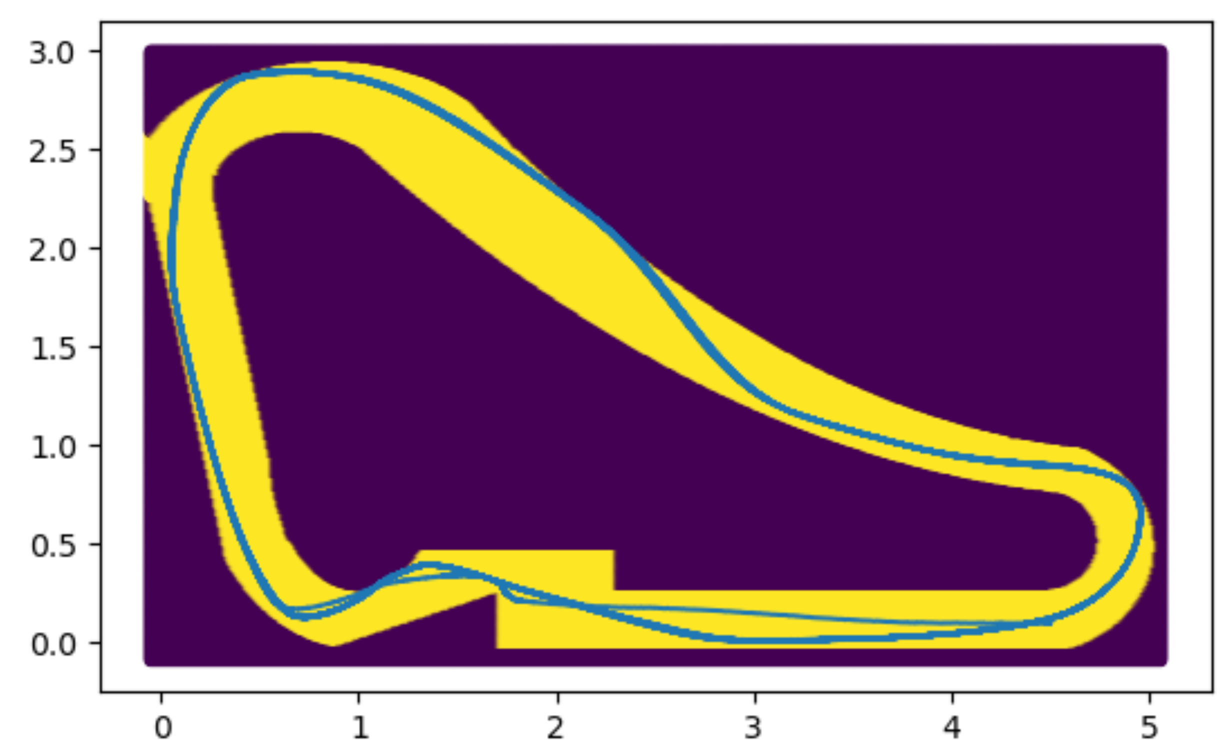
NASCAR and Monza circuits

A simple code has been implemented to simulate the driving behavior around a racing track.

First a simpler NASCAR style circuit, then a heavily romanticized rendition of the Monza circuit*

*which notably lacks its only decent corner, the Ascari chicane. [editor note]

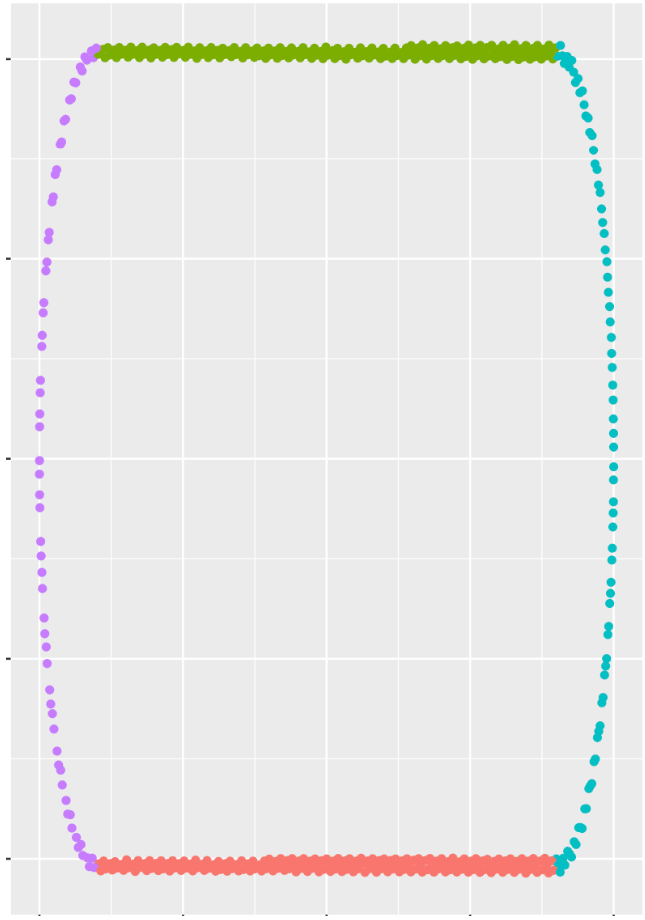




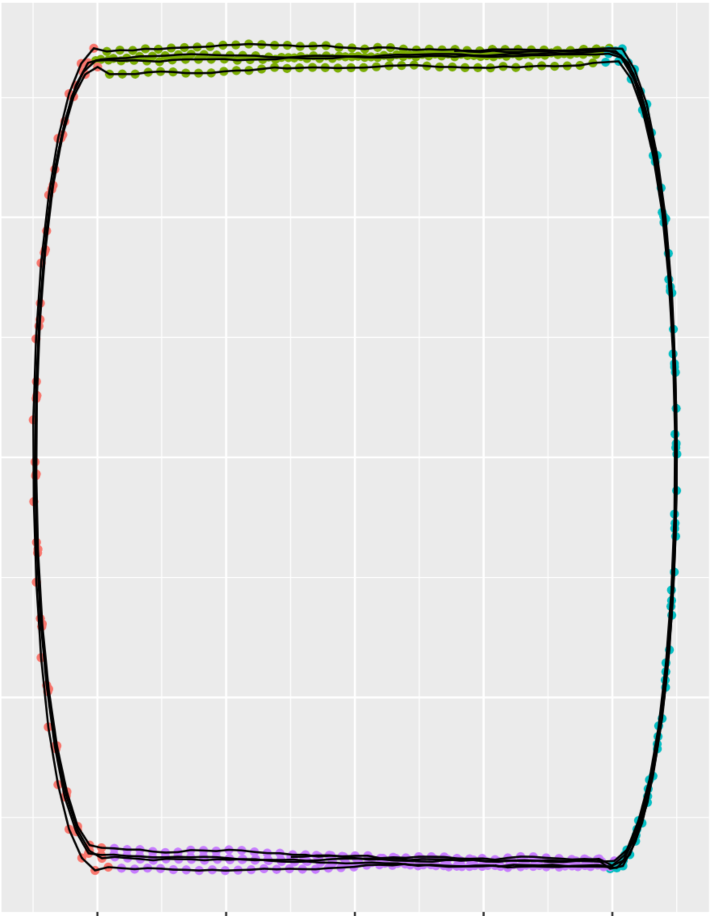
Results

NASCAR - JAGS

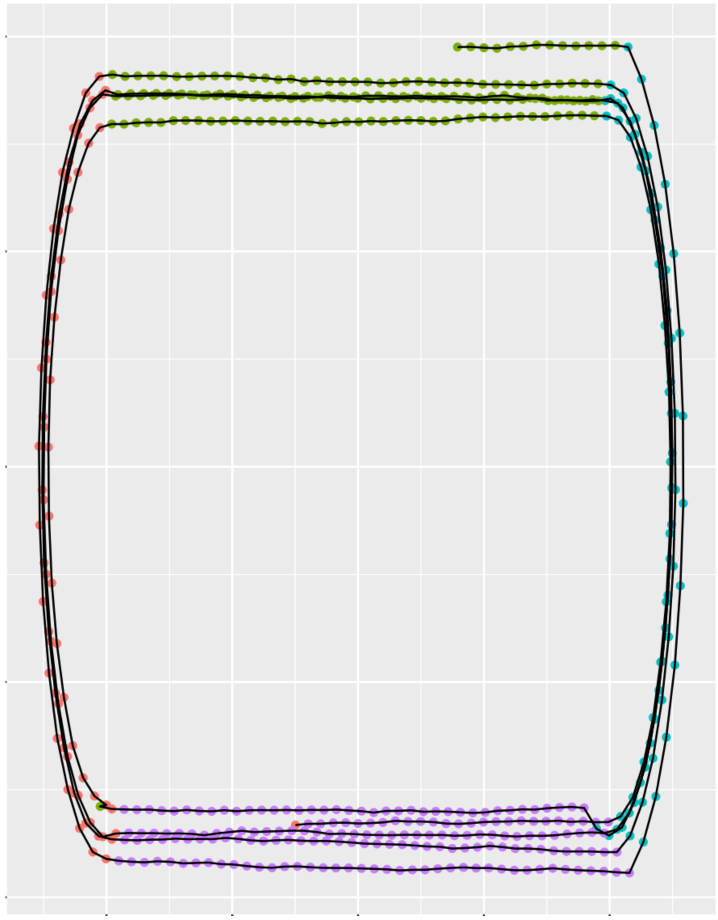
Final simulations have been run with 20000 burn-in steps and then generated 10 sequences of reconstructed data



Original



Reconstructed 1

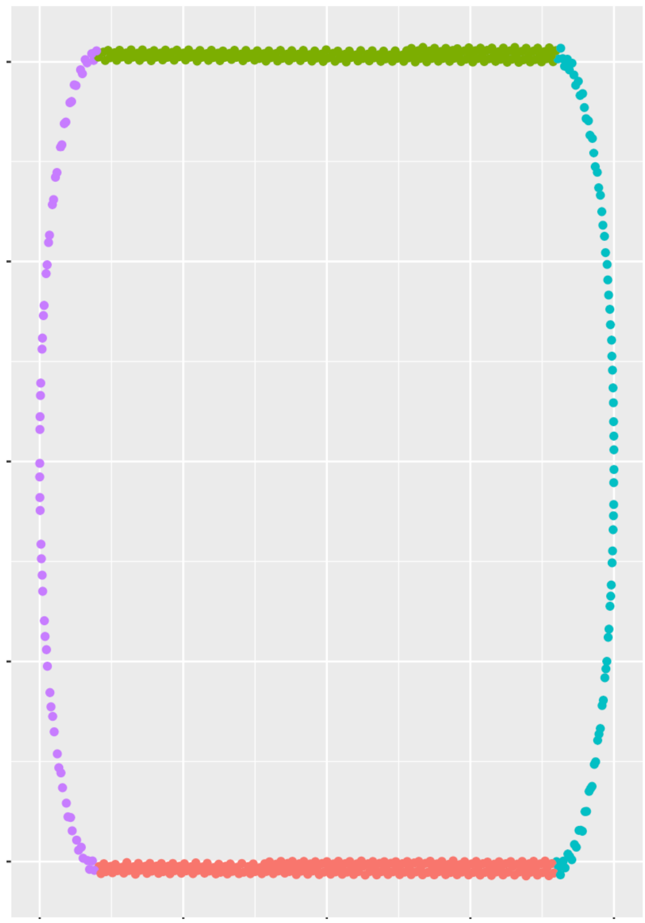


Reconstructed 2

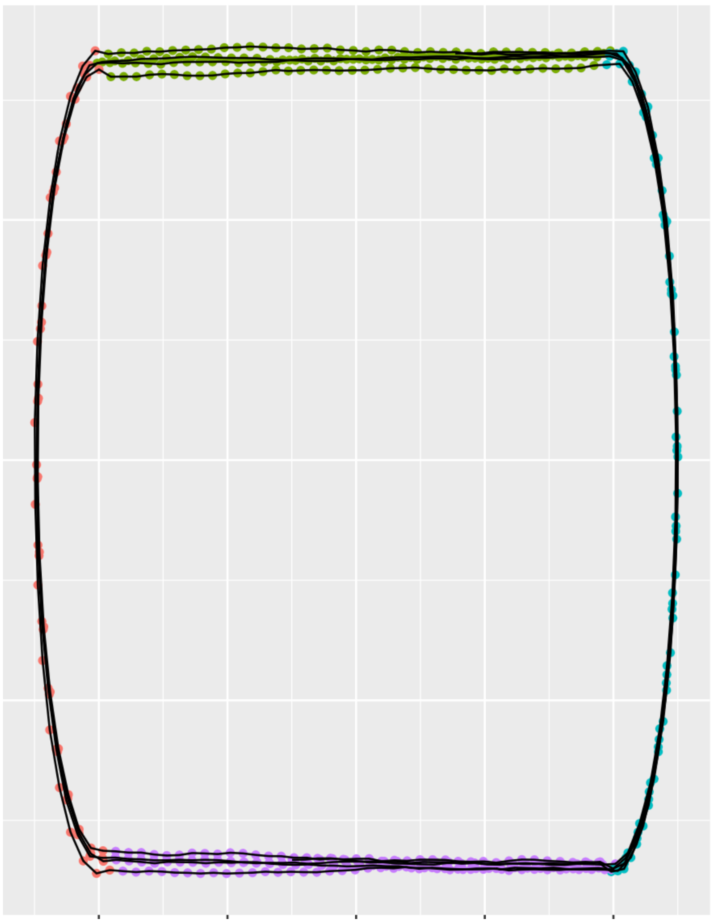
Results

Monza - JAGS

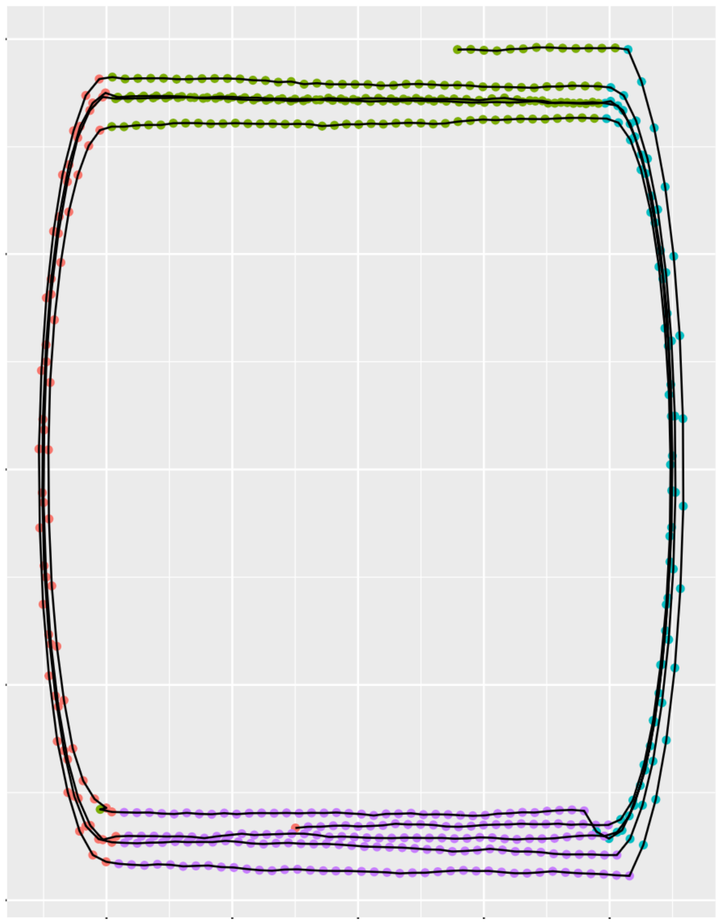
In this case the added complexity of the model tends to push the boundaries of the system, as a matter of fact stability is hard to achieve



Original



Reconstructed 1



Reconstructed 2

$$\gamma_t(k) = \Sigma_{j=1}^{K} p(z_t = k, z_{t-1} = j, x_{1:T}, y_{1:T}) =$$

$$= p(y_t \mid z_t = k, x_t) p(x_t \mid z_t = k, x_{t-1}) \cdot \Sigma_j \pi_{jk} p(z_{t-1} = j, x_{1:t-1}, y_{1:t-1}) =$$