# 1 Background

Musical recordings are complex data files that describe the intensity and onset time for every keystroke made by the performer. Matching this data to a musical score, removing incorrect notes, anticipating note onsets for automated accompaniment, comparing diverse performances, and discovering the relationship between performer choice and listener enjoyment all require "smoothing" the performance data so as to find low-dimensional structure. Statistical techniques like smoothing splines presume small changes in a derivative. But musical performances do not conform to these assumptions because tempo and dynamic interpretations rely on the juxtaposition of local smoothness with sudden changes and emphases to create listener interest. It is exactly the parts of a performance that are poorly described by statistical smoothers that render a performance interesting. Furthermore, many of these inflections are notated by the composer or are implicit in performance practice developed over centuries of musical expressivity. Consequently, regularization that incorporates domain knowledge leads to better statistical and empirical results [**?** ].

Figure 1 shows (blue dots) the note-by-note tempo of a 2003 recording attributed to Joyce Hatto. Splines with equally spaced knots (orange/dotted) are too smooth, and choosing locations to duplicate knots manually (red/dashed) to coincide with musical phrase endings works better. The solid green line shows a learned musical pattern from a Markov Switching state-space model we developed which can automatically learn tempo emphases (for example, near measure 40), where the performer plays individual notes slightly slower than the prevailing tempo, and automatically discover phrases without purposeful knot duplication. Interestingly, such musical analyses can help to compare performances—it was discovered in 2006 that this particular recording was actually made in 1988 by Eugen Indjic [**?** ].

This application is especially fascinating since it allows for visual, numerical, and aural exploration of the effects of tuning parameter selection on inference. Working with Prof. Chris Raphael on a Bösendorfer CEUS reproducing piano in the IU Jacobs School of Music, we can capture detailed measurements of key and pedal trajectories over time. The information is precise enough to reproduce an accurate replica by artificially "playing" the piano just as was done during the original performance. The piano can also create and respond to MIDI data. However, statistical inferences for these data are difficult. Current procedures are ill-suited for parameter estimation in a computationally efficient manner, as it amounts to Gaussian mixture learning with $K^n$ components. Existing optimization algorithms [**?** **?** ] thus only approximate the global solution.

# 2 The main idea

- We want to model tempo and dynamic decisions.
- We want a musician to understand what the parameters mean.

We will use a switching state-space model as shown in Figure 2. For now, we assume $s$ is a hidden Markov model on four states, denoted $S_1, \ldots, S_4$ with transition probability diagram given by Figure 3.

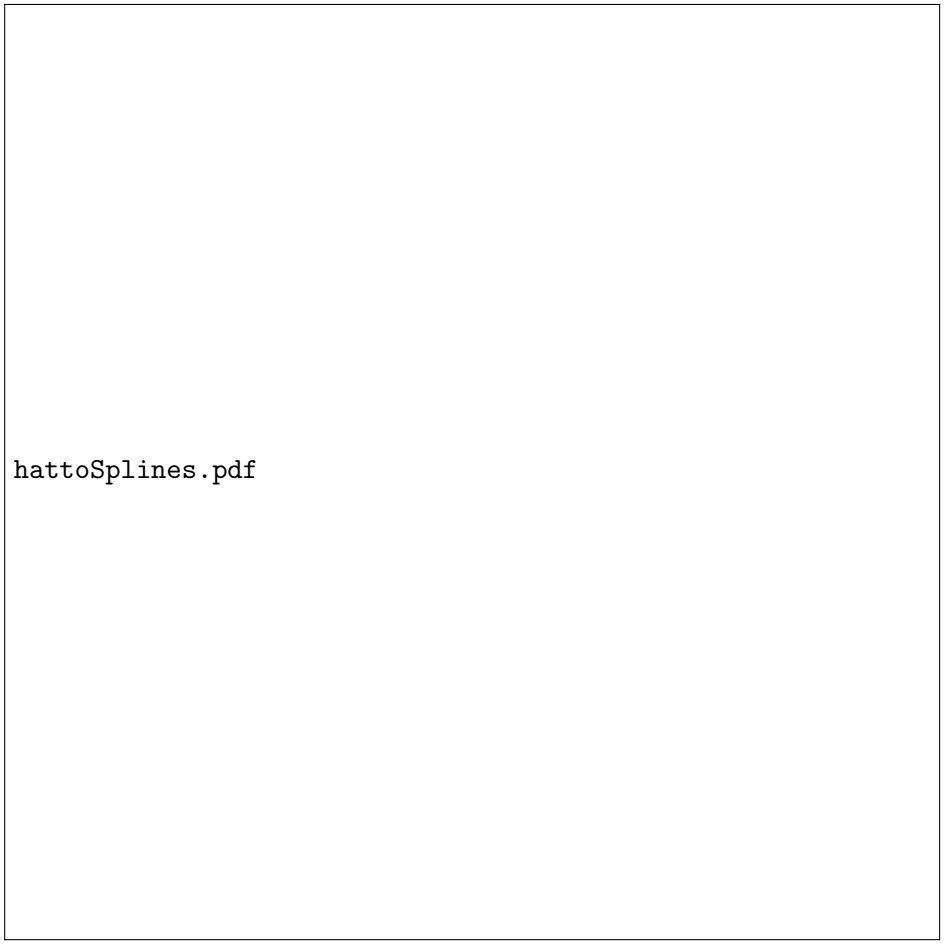


Figure 1: The tempo (beats/minute) of a 2003 recording attributed to Joyce Hatto.

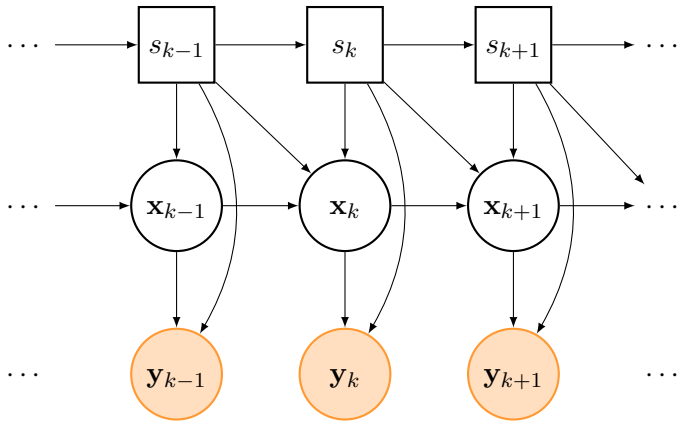


Figure 2: Switching state space model. Filled objects are observed, rectangles are discrete, and circles are continuous.

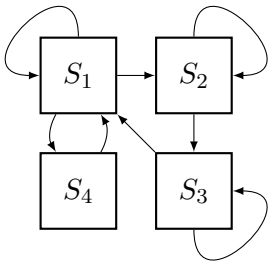


Figure 3: Transition diagram.

Models like this can have many behaviors, but for our case, the general form is:

$$x_t = d(s_t, s_{t-1}) + T(s_t, s_{t-1})x_t + R(s_t, s_{t-1})\eta_t \qquad \eta_t \sim N(0, Q(s_t, s_{t-1})) \tag{1}$$

$$y_t = c(s_t) + Z(s_t)x_t + \epsilon_t \qquad \epsilon_t \sim N(0, G(s_t)). \tag{2}$$

In other words, the hidden markov (switch) state determines which parameter matrices govern the evolution of the system.

The 4 switch states correspond to 4 different behaviors for the performer: (1) constant tempo, (2) speeding up, (3) slowing down, and (4) single note stress. The hidden continuous variable ($x_t$) is taken to be a two component vector with the first component being the "ideal" tempo and the second being the acceleration. Corresponding to these configurations, the parameter matrices are given in Table 1

Finally,

$$Q = \begin{cases} \begin{pmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_4^2 \end{pmatrix} & (s_t,\ s_{t-1}) = (S_4,\ S_1) \\ \begin{pmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{pmatrix} & \text{else.} \end{cases}$$

So for any performance, we want to be able to estimate the following parameters: $\sigma_1^2$, $\sigma_2^2$, $\sigma_4^2$, $\sigma_\epsilon^2$, the probabilities of the transition matrix (there are 4), and vectors $\mu$, $\tau$, and $\varphi$. These last three will be of different lengths depending on the number of times the state is visited. Lastly, we have the initial state distributions

$$x_1 \sim \begin{cases} N\left(\begin{pmatrix} \mu_1 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_1^2 & 0 \\ 0 & 0 \end{pmatrix}\right) & s_1 = S_1 \\ N\left(\begin{pmatrix} \mu_1 \\ \tau_1 \end{pmatrix}, \begin{pmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{pmatrix}\right) & s_1 = S_3. \end{cases}$$

Importantly, this is just one way to write this model.

The `R` function `yupengMats` creates all of these different parameter matrices.

## 3 R documentation

The problem with estimating a model like this is that, because the switch states and the continuous states are both hidden, this becomes an NP-hard problem. In particular, there are $4^N$ possible

| Transition equation | | | | |
|---|---|---|---|---|
| Switch states | | Parameter matrices | | |
| $s_t$ | $s_{t-1}$ | $d$ | $T$ | $R$ |
| $S_1$ | $S_1$ | $0$ | $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ | $0$ |
| $S_2$ | $S_1$ | $\begin{pmatrix} l_t\tau_t \\ \tau_t \end{pmatrix}$ | $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & l_t \\ 0 & l_t \end{pmatrix}$ |
| $S_4$ | $S_1$ | $\begin{pmatrix} 0 \\ \varphi_t \end{pmatrix}$ | $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$ |
| $S_2$ | $S_2$ | $0$ | $\begin{pmatrix} 1 & l_t \\ 0 & 1 \end{pmatrix}$ | $0$ |
| $S_3$ | $S_2$ | $\begin{pmatrix} l_t\tau_t \\ \tau_t \end{pmatrix}$ | $\begin{pmatrix} 1 & l_t \\ 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & l_t \\ 0 & l_t \end{pmatrix}$ |
| $S_1$ | $S_3$ | $\begin{pmatrix} \mu_t \\ 0 \end{pmatrix}$ | $0$ | $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ |
| $S_3$ | $S_3$ | $0$ | $\begin{pmatrix} 1 & l_t \\ 0 & 1 \end{pmatrix}$ | $0$ |
| $S_1$ | $S_4$ | $0$ | $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ | $0$ |

| Measurement equation | | | |
|---|---|---|---|
| Switch states | Parameter matrices | | |
| $s_t$ | $c$ | $Z$ | $G$ |
| $S_4$ | $0$ | $\begin{pmatrix} 1 & 1 \end{pmatrix}$ | $\sigma_\epsilon^2$ |
| else | $0$ | $\begin{pmatrix} 1 & 0 \end{pmatrix}$ | $\sigma_\epsilon^2$ |

Table 1: Parameter matrices of the switching state space model.

paths through the switch variables, so evaluating the likelihood at all of them is intractable. Thus, I implemented a particular approximation. The Beam Search (Algorithm 1), finds (greedily) evaluates the most likely path through the switch states. Another name for the algorithm is Discrete Particle Filter (`dpf`). Once we have those, `getLogLike` returns the negative loglikelihood of the data associated with that path. So for any configuration of parameters, we would form the matrices (`yupengMats`) then find the best path (`dpf`) then evaluate the likelihood of that path (`getLogLike`). We can then optimize over parameters using any variety of numerical optimization technique. However, when I have tried this, I always get infinite likelihood.

For this model, the dpf is more easily specified if we make the measurement equation depend only on the current state and not the previous state. For this reason, the code uses 16 states rather than 4. One can always change a Markov model in this way.

# 4 Python

A friend of mine wrote a python library for switching state space models (mattjj/pyslds). It may be easier to use this rather than the code I wrote. It is Bayesian rather than frequentist. If you're familiar with python, feel free to give it a whirl.

**Algorithm 1** Beam search

**Input:** Initial parameters of the matrices. Integer beam width $B$.
**for** $i = 1$ **to** $N$ **do**
    (`dpf` performs 1-step of the following);
    For each current path, calculate the 1-step likelihood for moving to each potential switch (`kf1step`)
    Multiply the likelihood by the probability of transitioning to that switch state
    Multiply by the previous path weights $w$
    If $\|w\|_0 > B$, resample the weights (`resampleSubOptimal`) to get $B$ non-zero weights which add to 1.
    Keep only those paths corresponding to the non-zero weights
**end for**
Return $B$ paths through the switch space along with their weights.