A Tutorial on Hidden Markov Models using Stan

Luis Damiano (Universidad Nacional de Rosario), Brian Peterson (University of Washington), Michael Weylandt (Rice University)

2017-12-15

Contents

This case study documents the implementation in Stan (Carpenter et al. 2017) of the Hidden Markov Model (HMM) for unsupervised learning (Baum and Petrie 1966; Baum and Eagon 1967; Baum and Sell 1968; Baum et al. 1970; Baum 1972). Additionally, we present the adaptations needed for the Input-Output Hidden Markov Model (IOHMM). IOHMM is an architecture proposed by Bengio and Frasconi (1995) to map input sequences, sometimes called the control signal, to output sequences. Compared to HMM, it aims at being especially effective at learning long term memory, that is when input-output sequences span long points. In all cases, we provide a fully Bayesian estimation of the model parameters and inference on hidden quantities, namely filtered and smoothed posterior distribution of the hidden states, and jointly most probable state path.

A Tutorial on Hidden Markov Models using Stan is distributed under the Creative Commons Attribution 4.0 International Public License. Accompanying code is distributed under the GNU General Public License v3.0. See the README file for details. All files are available in the stancon18 GitHub repository.

---

1 The Hidden Markov Model

Real-world processes produce observable outputs characterized as signals. These can be discrete or continuous in nature, can be pure or embed uncertainty about the measurements and the explanatory model, come from a stationary or non-stationary source, among many other variations. These signals are modeled to allow for both theoretical descriptions and practical applications. The model itself can be deterministic or stochastic, in which case the signal is characterized as a parametric random process whose parameters can be estimated in a well-defined manner.

Autocorrelation, a key feature in most signals, can be modeled in countless forms. While certainly pertinent to this purpose, high-order Markov chains can prove inconvenient when the range of the correlation among the observations is long. A more parsimonious approach assumes that the observed sequence is a noisy observation of an underlying hidden process represented as a first-order Markov chain. In other terms, long-range dependencies between observations are mediated via latent variables. It is important to note that the Markov property is only assumed for the hidden states, and the observations are assumed conditionally independent given these latent states.

1.1 Model specification

HMM involve two interconnected models. The state model consists of a discrete-time, discrete-state, first-order Markov chain $z_t \in \{1, \dots, K\}$ that transitions according to $p(z_t|z_{t-1})$. In turns, the observation model is governed by $p(\mathbf{y}_t|z_t)$, where $\mathbf{y}_t$ are the observations, emissions or output.[1] The corresponding joint distribution is

$$p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T}) = p(\mathbf{z}_{1:T})p(\mathbf{y}_{1:T}|\mathbf{z}_{1:T}) = \left[p(z_1)\prod_{t=2}^{T} p(z_t|z_{t-1})\right]\left[\prod_{t=1}^{T} p(\mathbf{y}_t|z_t)\right].$$

This is a specific instance of the state space model family in which the latent variables are discrete. Each single time slice corresponds to a mixture distribution with component densities given by $p(\mathbf{y}_t|z_t)$, thus HMM may be interpreted as an extension of a mixture model in which the choice of component for each observation is not selected independently but depends on the choice of component for the previous observation. In the case of a simple mixture model for an independent and identically distributed sample, the parameters of the transition matrix inside the i-th column are the same, so that the conditional distribution $p(z_t|z_{t-1})$ is independent of z_{t-1}.

When the output is discrete, the observation model commonly takes the form of an observation matrix

$$p(\mathbf{y}_t|z_t = k, \theta) = \text{categorical}(\mathbf{y}_t|\theta_k)$$

Alternatively, if the output is continuous, the observation model is frequently a conditional Gaussian

$$p(\mathbf{y}_t|z_t = k, \theta) = \mathcal{N}(\mathbf{y}_t|\mu_k, \mathbf{\Sigma}_k).$$

The latter is equivalent to a Gaussian mixture model with cluster membership ruled by Markovian dynamics, also known as Markov Switching Models (MSM). In this context, multiple sequential observations tend to share the same location until they suddenly jump into a new cluster.

The non-stochastic quantities of the model are the length of the observed sequence T and the number of hidden states K. The observed sequence $\mathbf{y}_t$ is a stochastic known quantity. The parameters of the models are $\theta = (\pi_1, \theta_h, \theta_o)$, where π_1 is the initial state distribution, θ_h are the parameters of the hidden model and θ_o are the parameters of the state-conditional density function $p(\mathbf{y}_t|z_t)$. The form of θ_h and θ_o depends on the specification of each model. In the case under study, state transition is characterized by the $K \times K$ sized transition matrix with simplex rows $\mathbf{A} = \{a_{ij}\}$ with $a_{ij} = p(z_t = j|z_{t-1} = i)$.

The following Stan code illustrates the case of continuous observations where emissions are modeled as sampled from the Gaussian distribution with parameters μ_k and σ_k for $k \in \{1, \dots, K\}$. Adaptation for categorical observations should follow the guidelines outlined in the manual (Stan Development Team 2017c, sec. 10.6).

[1] The output can be univariate or multivariate depending on the choice of model specification, in which case an observation at a given time index t is a scalar y_t or a vector $\mathbf{y}_t$ respectively. Although we introduce definitions and properties along an example based on a univariate series, we keep the bold notation to remind that the equations are valid in a multivariate context as well.

```
data {
  int<lower=1> T;                 // number of observations (length)
  int<lower=1> K;                 // number of hidden states
  real y[T];                      // observations
}

parameters {
  // Discrete state model
  simplex[K] pi1;                 // initial state probabilities
  simplex[K] A[K];                // transition probabilities
                                  // A[i][j] = p(z_t = j | z_{t-1} = i)

  // Continuous observation model
  ordered[K] mu;                  // observation means
  real<lower=0> sigma[K];         // observation standard deviations
}
```

1.2 The generative model

We write a routine in the R programming language for our generative model. Broadly speaking, this involves three steps:

1. The generation of parameters according to the priors $\theta^{(0)} \sim p(\theta)$.
2. The generation of the hidden path $\mathbf{z}_{1:T}^{(0)}$ according to the transition model parameters.
3. The generation of the observed quantities based on the sampling distribution $\mathbf{y}_t^{(0)} \sim p(\mathbf{y}_t|\mathbf{z}_{1:T}^{(0)}, \theta^{(0)})$.

We break down the description of our code in these three steps.

```r
runif_simplex <- function(T) {
  x <- -log(runif(T))
  x / sum(x)
}

hmm_generate <- function(K, T) {
  # 1. Parameters
  pi1   <- runif_simplex(K)
  A     <- t(replicate(K, runif_simplex(K)))
  mu    <- sort(rnorm(K, 10 * 1:K, 1))
  sigma <- abs(rnorm(K))

  # 2. Hidden path
  z <- vector("numeric", T)

  z[1] <- sample(1:K, size = 1, prob = pi1)
  for (t in 2:T)
    z[t] <- sample(1:K, size = 1, prob = A[z[t - 1], ])

  # 3. Observations
  y <- vector("numeric", T)
  for (t in 1:T)
    y[t] <- rnorm(1, mu[z[t]], sigma[z[t]])

  list(y = y, z = z,
       theta = list(pi1 = pi1, A = A,
```

```
                    mu = mu, sigma = sigma))
}
```

1.2.1 Generating parameters from the priors

The parameters to be generated include the K-sized initial state distribution vector π_1 and the $K \times K$ transition matrix $\mathbf{A}$. There are $(K-1)(K+1)$ free parameters as the vector and each row of the matrix are simplexes.

We set up uniform priors for π_1 and $\mathbf{A}$, a weakly informative Gaussian for the location parameter μ_k and a weakly informative half-Gaussian that ensures positivity for the scale parameters σ_k. An ordinal constraint is imposed on the location parameter to restrict the exploration of the symmetric, degenerate mixture posterior surface to a single ordering of the parameters, thus solving the non-identifiability issues inherent to the model density (Betancourt 2017). In the simulation routine, the location parameters are adjusted to ensure that the observations are well-separated. We refer the reader to the Prior Choice Recommendations wiki article for very useful practical guidelines that are both computationally and statistically meaningful. Given the fixed quantity K, we draw one sample from the prior distributions $\theta^{(0)} \sim p(\theta)$.

1.2.2 Generating the hidden path

The initial hidden state is drawn from a multinomial distribution with one trial and event probabilities given by the initial state probability vector $\pi_1^{(0)}$. Given the fixed quantity T, the transition probabilities for each of the following steps $t \in \{2, \ldots, T\}$ are generated from a multinomial distribution with one trial and event probabilities given by the i-th row of the transition matrix $\mathbf{A}_1^{(0)}$, where i is the state at the previous time step $z_{t-1}^{(0)} = i$. The hidden states are subsequently sampled based on these transition probabilities.

1.2.3 Generating data from the sampling distribution

The observations conditioned on the hidden states are drawn from a univariate Gaussian density with parameters $\mu_k^{(0)}$ and $\sigma_k^{(0)}$.

1.3 Characteristics

One of the most powerful properties of HMM is the ability to exhibit some degree of invariance to local warping of the time axis. Allowing for compression or stretching of the time, the model accommodates for variations in speed. By specification of the latent model, the density function of the duration τ in state i is given by

$$p_i(\tau) = (A_{ii})^{\tau}(1 - A_{ii}) \propto \exp(-\tau \ln A_{ii}),$$

which represents the probability that a sequence spends precisely τ steps in state i. The expected duration conditional on starting in that state is

$$\bar{\tau}_i = \sum_{\tau=1}^{\infty} \tau p_i(\tau) = \frac{1}{1 - A_{ii}}.$$

The density is an exponentially decaying function of τ, thus longer durations are less probable than shorter ones. In applications where this proves unrealistic, the diagonal coefficients of the transition matrix $A_{ii}\ \forall\ i$ may be set to zero and each state i is explicitly associated with a probability distribution of possible duration times $p(\tau|i)$ (Rabiner 1990).

1.4 Inference

There are several quantities of interest that can be inferred via different algorithms. Our code contains the implementation of the most relevant methods for unsupervised data: forward, forward-backward and Viterbi decoding algorithms. We acknowledge the authors of the Stan Manual for the thorough illustrations and code snippets, some of which served as a starting point for our own code. As estimation is treated later, we assume that model parameters θ are known.

Table 1: Summary of the hidden quantities and their corresponding inference algorithm. † Time complexity is reduced to $O(KT)$ for inference on a left-to-right (upper triangular) transition matrix.

Name	Hidden Quantity	Availability at	Algorithm	Complexity	
Filtering	$p(z_t\|\mathbf{y}_{1:t})$	t (online)	Forward	$O(K^2T)$	$O(KT)$†
Smoothing	$p(z_t\|\mathbf{y}_{1:T})$	T (offline)	Forward-backward	$O(K^2T)$	$O(KT)$†
Fixed lag smoothing	$p(z_{t-\ell}\|\mathbf{y}_{1:t})$, $\ell \geq 1$	$t+\ell$ (lagged)	Forward-backward	$O(K^2T)$	$O(KT)$†
State prediction	$p(z_{t+h}\|\mathbf{y}_{1:t})$, $h \geq 1$	t			
Observation prediction	$p(y_{t+h}\|\mathbf{y}_{1:t})$, $h \geq 1$	t			
MAP Estimation	$\text{argmax}_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T}\|\mathbf{y}_{1:T})$	T	Viterbi decoding	$O(K^2T)$	
Log likelihood	$p(\mathbf{y}_{1:T})$	T	Forward	$O(K^2T)$	$O(KT)$†

1.4.1 Filtering

A filter infers the posterior distribution of the hidden states at a given step t based on all the information available up to that point $p(z_t|\mathbf{y}_{1:t})$. It achieves better noise reduction than simply estimating the hidden state based on the current estimate $p(z_t|\mathbf{y}_t)$. The filtering process can be run online, or recursively, as new data streams in.

Filtered marginals can be computed recursively by means of the forward algorithm (Baum and Eagon 1967). Let $\psi_t(j) = p(\mathbf{y}_t|z_t = j)$ be the local evidence at step t and $\Psi(i,j) = p(z_t = j|z_{t-1} = i)$ be the transition probability. First, the one-step-ahead predictive density is computed

$$p(z_t = j|\mathbf{y}_{1:t-1}) = \sum_i \Psi(i,j)p(z_{t-1} = i|\mathbf{y}_{1:t-1}).$$

Acting as prior information, this quantity is updated with observed data at the step t using the Bayes rule,

$$\begin{aligned}\alpha_t(j) &\triangleq p(z_t = j|\mathbf{y}_{1:t}) \\ &= p(z_t = j|\mathbf{y}_t, \mathbf{y}_{1:t-1}) \\ &= Z_t^{-1}\psi_t(j)p(z_t = j|\mathbf{y}_{1:t-1})\end{aligned}$$

where the normalization constant is given by

$$Z_t \triangleq p(\mathbf{y}_t|\mathbf{y}_{1:t-1}) = \sum_{l=1}^{K} p(\mathbf{y}_t|z_t = l)p(z_t = l|\mathbf{y}_{1:t-1}) = \sum_{l=1}^{K} \psi_t(l)p(z_t = l|\mathbf{y}_{1:t-1}).$$

This predict-update cycle results in the filtered belief states at step t. As this algorithm only requires the evaluation of the quantities $\psi_t(j)$ for each value of z_t for every t and fixed $\mathbf{y}_t$, the posterior distribution of

the latent states is independent of the form of the observation density or indeed of whether the observed variables are continuous or discrete (Jordan 2003). In other words, $\alpha_t(j)$ does not depend on the complete form of the density function $p(\mathbf{y}|\mathbf{z})$ but only on the point values $p(\mathbf{y}_t|z_t = j)$ for every $\mathbf{y}_t$ and z_t.

Let α_t be a K-sized vector with the filtered belief states at step t, $\psi_t(j)$ be the K-sized vector of local evidence at step t, $\mathbf{\Psi}$ be the transition matrix and $\mathbf{u} \odot \mathbf{v}$ be the Hadamard product, representing element-wise vector multiplication. Then, the Bayesian updating procedure can be expressed in matrix notation as

$$\alpha_t \propto \psi_t \odot (\mathbf{\Psi}^T \alpha_{t-1}).$$

In addition to computing the hidden states, the algorithm yields the log likelihood

$$\mathcal{L} = \log p(\mathbf{y}_{1:T}|\theta) = \sum_{t=1}^{T} \log p(\mathbf{y}_t|\mathbf{y}_{1:t-1}) = \sum_{t=1}^{T} \log Z_t.$$

```
transformed parameters {
  vector[K] logalpha[T];

  { // Forward algorithm log p(z_t = j | y_{1:t})
    real accumulator[K];

    logalpha[1] = log(pi1) + normal_lpdf(y[1] | mu, sigma);

    for (t in 2:T) {
      for (j in 1:K) { // j = current (t)
        for (i in 1:K) { // i = previous (t-1)
                         // Murphy (2012) p. 609 eq. 17.48
                         // belief state      + transition prob + local evidence at t
          accumulator[i] = logalpha[t-1, i] + log(A[i, j]) + normal_lpdf(y[t] | mu[j], sigma[j]);
        }
        logalpha[t, j] = log_sum_exp(accumulator);
      }
    }
  } // Forward
}
```

The Stan code makes evident that the time complexity of the algorithm is $O(K^2T)$: there are $K \times K$ iterations within each of the T iterations of the outer loop. Brute-forcing through all possible hidden states K^T would prove prohibitive for realistic problems as time complexity increases exponentially with sequence length $O(K^T T)$.

The implementation is representative of the matrix notation in Murphy (2012 eq. 17.48). The `accumulator` variable carries the element-wise operations for all possible previous states which are later combined as indicated by the matrix multiplication.

Since log domain should be preferred to avoid numerical underflow, multiplications are translated into sums of logs. Furthermore, we use Stan's implementation of the linear sums on the log scale to prevent underflow and overflow in the exponentiation (Stan Development Team 2017c, 141). In consequence, `logalpha` represents the forward quantity in log scale and needs to be exponentially normalized for interpretability.

```
generated quantities {
  vector[K] alpha[T];

  { // Forward algortihm
    for (t in 1:T)
      alpha[t] = softmax(logalpha[t]);
```

```
  } // Forward
}
```

Since the unnormalized forward probability is sufficient to compute the posterior log density and estimate the parameters, it should be part of either the `model` or the `transformed parameters` blocks. We chose the latter to keep track of the estimates. We expand on estimation afterward.

1.4.2 Smoothing

A smoother infers the posterior distribution of the hidden states at a given state based on all the observations or evidence $p(z_t|\mathbf{y}_{1:T})$. Although noise and uncertainty are significantly reduced as a result of conditioning on past and future data, the smoothing process can only be run offline.

Inference can be done by means of the forward-backward algorithm, which also plays an important role in the Baum-Welch algorithm for learning model parameters (Baum and Eagon 1967; Baum et al. 1970). Let $\gamma_t(j)$ be the desired smoothed posterior marginal,

$$\gamma_t(j) \triangleq p(z_t = j|\mathbf{y}_{1:T}),$$

$\alpha_t(j)$ be the filtered belief state at the step t as defined previously, and $\beta_t(j)$ be the conditional likelihood of future evidence given that the hidden state at step t is j,

$$\beta_t(j) \triangleq p(\mathbf{y}_{t+1:T}|z_t = j).$$

Then, the chain of smoothed marginals can be segregated into the past and the future components by conditioning on the belief state z_t,

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{p(\mathbf{y}_{1:T})} \propto \alpha_t(j)\beta_t(j).$$

The future component can be computed recursively from right to left:

$$\begin{aligned}
\beta_{t-1}(i) &= p(\mathbf{y}_{t:T}|z_{t-1} = i) \\
&= \sum_{j=1}^{K} p(z_t = j, \mathbf{y}_t, \mathbf{y}_{t+1:T}|z_{t-1} = i) \\
&= \sum_{j=1}^{K} p(\mathbf{y}_{t+1:T}|z_t = j)p(z_t = j, \mathbf{y}_t|z_{t-1} = i) \\
&= \sum_{j=1}^{K} p(\mathbf{y}_{t+1:T}|z_t = j)p(\mathbf{y}_t|z_t = j)p(z_t = j|z_{t-1} = i) \\
&= \sum_{j=1}^{K} \beta_t(j)\psi_t(j)\Psi(i, j)
\end{aligned}$$

Let β_t be a K-sized vector with the conditional likelihood of future evidence given the hidden state at step t. Then, the backward procedure can be expressed in matrix notation as

$$\beta_{t-1} \propto \mathbf{\Psi}(\psi_t \odot \beta_t).$$

At the last step, the base case is given by

$$\beta_T(i) = p(\mathbf{y}_{T+1:T}|z_T = i) = p(\varnothing|z_t = i) = 1.$$

Intuitively, the forward-backward algorithm passes information from left to right and then from right to left, combining them at each node. A straightforward implementation of the algorithm runs in $O(K^2T)$ time because of the $K \times K$ matrix multiplication at each step. Nonetheless the frequent description as two subsequent passes, these procedures are not inherently sequential and share no information. As a result, they could be implemented in parallel.

There is a significant reduction if the transition matrix is sparse. Inference for a left-to-right (upper triangular) transition matrix, a model where the state index increases or stays the same as time passes, runs in $O(TK)$ time (Bakis 1976; Jelinek 1976). Additional assumptions about the form of the transition matrix may ease complexity further, for example decreasing the time to $O(TK \log K)$ if $\psi(i, j) \propto \exp(-\sigma^2|\mathbf{z}_i - \mathbf{z}_j|)$. Finally, ad-hoc data pre-processing strategies may help control complexity, for example by pruning nodes with low conditional probability of occurrence.

```
functions {
  vector normalize(vector x) {
    return x / sum(x);
  }
}

generated quantities {
  vector[K] alpha[T];

  vector[K] logbeta[T];
  vector[K] loggamma[T];

  vector[K] beta[T];
  vector[K] gamma[T];

  { // Forward algortihm
    for (t in 1:T)
      alpha[t] = softmax(logalpha[t]);
  } // Forward

  { // Backward algorithm log p(y_{t+1:T} | z_t = j)
    real accumulator[K];

    for (j in 1:K)
      logbeta[T, j] = 1;

    for (tforward in 0:(T-2)) {
      int t;
      t = T - tforward;

      for (j in 1:K) {    // j = previous (t-1)
        for (i in 1:K) {  // i = next (t)
                          // Murphy (2012) Eq. 17.58
                          // backwards t    + transition prob + local evidence at t
          accumulator[i] = logbeta[t, i] + log(A[j, i]) + normal_lpdf(y[t] | mu[i], sigma[i]);
          }
        logbeta[t-1, j] = log_sum_exp(accumulator);
      }
```

```
    }

    for (t in 1:T)
      beta[t] = softmax(logbeta[t]);
  } // Backward

  { // forward-backward algorithm log p(z_t = j | y_{1:T})
    for(t in 1:T) {
        loggamma[t] = alpha[t] .* beta[t];
    }

    for(t in 1:T)
      gamma[t] = normalize(loggamma[t]);
  } // forward-backward
}
```

The reader should not be deceived by the similarity to the code shown in the filtering section. Note that the indices in the log transition matrix are inverted and the evidence is now computed for the next state. We need to invert the time index as backward ranges are not available in Stan.

1.4.2.1 Inference recapitulation

The forward-backward algorithm was designed to exploit via recursion the conditional independencies in the HMM. First, the posterior marginal probability of the latent states at a given time step is broken down into two quantities: the past and the future components. Second, taking advantage of the Markov properties, each of the two are further broken down into simpler pieces via conditioning and marginalizing, thus creating an efficient predict-update cycle.

This strategy makes otherwise unfeasible calculations possible. Consider for example a time series with $T = 100$ observations and $K = 5$ hidden states. Summing the joint probability over all possible state sequences would involve $2 \times 100 \times 5^{100} \approx 10^{72}$ computations, while the forward and backward passes only take $3,000$ each. Moreover, one pass may be avoided depending on the goal of the data analysis. Summing the forward probabilities at the last time step is enough to compute the likelihood, and the backwards recursion would be only needed if the posterior probabilities of the states were also required.

1.4.3 MAP: Viterbi

It is also of interest to compute the most probable state sequence or path,

$$\mathbf{z}^* = \underset{\mathbf{z}_{1:T}}{\operatorname{argmax}}\, p(\mathbf{z}_{1:T}|\mathbf{y}_{1:T}).$$

The jointly most probable sequence of states can be inferred by means of maximum a posterior (MAP) estimation. It is not necessarily the same as the sequence of marginally most probable states given by the maximizer of the posterior marginals (MPM),

$$\hat{\mathbf{z}} = (\underset{z_1}{\operatorname{argmax}}\, p(z_1|\mathbf{y}_{1:T}),\ \ldots,\ \underset{z_T}{\operatorname{argmax}}\, p(z_T|\mathbf{y}_{1:T})),$$

which maximizes the expected number of correct individual states.

The MAP estimate is always globally consistent: while locally a state may be most probable at a given step, the Viterbi or max-sum algorithm decodes the most likely single plausible path (Viterbi 1967). Furthermore, the MPM sequence may have zero joint probability if it includes two successive states that, while being individually the most probable, are connected in the transition matrix by a zero. On the other hand, MPM

can be considered more robust since the state at each step is estimated by averaging over its neighbors rather than conditioning on a specific value of them.

The Viterbi applies the max-sum algorithm in a forward fashion plus a traceback procedure to recover the most probable path. In simple terms, once the most probable state z_t is estimated, the procedure conditions the previous states on it. Let $\delta_t(j)$ be the probability of arriving to the state j at step t given the most probable path was taken,

$$\delta_t(j) \triangleq \max_{z_1,\ldots,z_{t-1}} p(\mathbf{z}_{1:t-1}, z_t = j|\mathbf{y}_{1:t}).$$

The most probable path to state j at step t consists of the most probable path to some other state i at point $t-1$, followed by a transition from i to j,

$$\delta_t(j) = \max_i \delta_{t-1}(i)\ \psi(i,j)\ \psi_t(j).$$

Additionally, the most likely previous state on the most probable path to j at step t is given by

$$a_t(j) = \operatorname{argmax}_i \delta_{t-1}(i)\ \psi(i,j)\ \psi_t(j).$$

By initializing with $\delta_1 = \pi_j \phi_1(j)$ and terminating with the most probable final state $z_T^* = \operatorname{argmax}_i \delta_T(i)$, the most probable sequence of states is estimated using the traceback,

$$z_t^* = a_{t+1}(z_{t+1}^*).$$

It is advisable to work in the log domain to avoid numerical underflow,

$$\delta_t(j) \triangleq \max_{\mathbf{z}_{1:t-1}} \log p(\mathbf{z}_{1:t-1}, z_t = j|\mathbf{y}_{1:t}) = \max_i \log \delta_{t-1}(i) + \log \psi(i,j) + \log \psi_t(j).$$

As with the backward-forward algorithm, the time complexity of Viterbi is $O(K^2T)$ and the space complexity is $O(KT)$. If the transition matrix has the form $\psi(i,j) \propto \exp(-\sigma^2||\mathbf{z}_i - \mathbf{z}_j||^2)$, implementation runs in $O(TK)$ time.

```
generated quantities {
  int<lower=1, upper=K> zstar[T];
  real logp_zstar;

  { // Viterbi algorithm
    int bpointer[T, K]; // backpointer to the most likely previous state on the most probable path
    real delta[T, K];   // max prob for the sequence up to t
                        // that ends with an emission from state k

    for (j in 1:K)
      delta[1, K] = normal_lpdf(y[1] | mu[j], sigma[j]);

    for (t in 2:T) {
      for (j in 1:K) { // j = current (t)
        delta[t, j] = negative_infinity();
        for (i in 1:K) { // i = previous (t-1)
          real logp;
          logp = delta[t-1, i] + log(A[i, j]) + normal_lpdf(y[t] | mu[j], sigma[j]);
          if (logp > delta[t, j]) {
            bpointer[t, j] = i;
```

```
            delta[t, j] = logp;
          }
        }
      }
    }

    logp_zstar = max(delta[T]);

    for (j in 1:K)
      if (delta[T, j] == logp_zstar)
        zstar[T] = j;

    for (t in 1:(T - 1)) {
      zstar[T - t] = bpointer[T - t + 1, zstar[T - t + 1]];
    }
  }
}
```

The variable `delta` is a straightforward implementation of the corresponding equation. `bpointer` is the traceback needed to compute the most probable sequence of states after the most probably final state `zstar` is computed.

1.5 Parameter estimation

The model likelihood can be derived from the definition of the quantity $\gamma_t(j)$: given that its sum over all possible values of the latent variable must equal one, the log likelihood at time index t becomes

$$\mathcal{L}_t = \sum_{i=1}^{K} \alpha_t(i)\beta_t(i).$$

The last step T has two convenient characteristics. First, the recurrent nature of the forward probability implies that the last iteration retains the information of all the intermediate state probabilities. Second, the base case for the backwards quantity is $\beta_T(i) = 1$. Consequently, the log likelihood reduces to

$$\mathcal{L}_T \propto \sum_{i=1}^{K} \alpha_T(i).$$

```
model {
  target += log_sum_exp(logalpha[T]); // Note: update based only on last logalpha
}
```

As we expect high multimodality in the posterior density, we use a clustering algorithm to feed the sampler with initialization values for the location and scale parameters. Although K-means is not truthful to the real model because it does not consider the time-dependent nature of the data, it provides an educated guess.

```
hmm_init <- function(K, y) {
  clasif <- kmeans(y, K)
  init.mu <- by(y, clasif$cluster, mean)
  init.sigma <- by(y, clasif$cluster, sd)
  init.order <- order(init.mu)

  list(
    mu = init.mu[init.order],
```

```
    sigma = init.sigma[init.order]
  )
}

hmm_fit <- function(K, y) {
  rstan_options(auto_write = TRUE)
  options(mc.cores = parallel::detectCores())

  stan.model = 'stan/hmm_gaussian.stan'
  stan.data = list(
    T = length(y),
    K = K,
    y = y
  )

  stan(file = stan.model,
       data = stan.data, verbose = T,
       iter = 400, warmup = 200,
       thin = 1, chains = 1,
       cores = 1, seed = 900,
       init = function(){hmm_init(K, y)})
}
```

1.6 A walk-through

We draw one sample of length $T = 500$ from a data generating process with $K = 3$ latent states. We fit the model using the Stan code and the initialization methodology introduced previously.

```
set.seed(900)
K        <- 3
T_length <- 500
dataset  <- hmm_generate(K, T_length)
fit      <- hmm_fit(K, dataset$y)
```

```
##
## TRANSLATING MODEL 'hmm_gaussian' FROM Stan CODE TO C++ CODE NOW.
## successful in parsing the Stan model 'hmm_gaussian'.
##
## CHECKING DATA AND PREPROCESSING FOR MODEL 'hmm_gaussian' NOW.
##
## COMPILING MODEL 'hmm_gaussian' NOW.
##
## STARTING SAMPLER FOR MODEL 'hmm_gaussian' NOW.
##
## SAMPLING FOR MODEL 'hmm_gaussian' NOW (CHAIN 1).
##
## Gradient evaluation took 0.002 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 20 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:   1 / 400 [  0%]  (Warmup)
## Iteration:  40 / 400 [ 10%]  (Warmup)
```

```
## Iteration:  80 / 400 [ 20%]  (Warmup)
## Iteration: 120 / 400 [ 30%]  (Warmup)
## Iteration: 160 / 400 [ 40%]  (Warmup)
## Iteration: 200 / 400 [ 50%]  (Warmup)
## Iteration: 201 / 400 [ 50%]  (Sampling)
## Iteration: 240 / 400 [ 60%]  (Sampling)
## Iteration: 280 / 400 [ 70%]  (Sampling)
## Iteration: 320 / 400 [ 80%]  (Sampling)
## Iteration: 360 / 400 [ 90%]  (Sampling)
## Iteration: 400 / 400 [100%]  (Sampling)
##
##  Elapsed Time: 26.925 seconds (Warm-up)
##                3.612 seconds (Sampling)
##                30.537 seconds (Total)
```

The estimates are extremely efficient as expected when dealing with generated data. The Markov Chain are well behaved as diagnosed by the low Monte Carlo standard error, the high effective sample size and the near-one shrink factor of Gelman and Rubin (1992). Although not shown, further diagnostics confirm satisfactory mixing, convergence and the absence of divergences. Point estimates and posterior intervals are provided by rstan's `summary` function.

```
knitr::kable(cbind(
             unlist(list(dataset$theta$pi1,
                         t(dataset$theta$A),
                         dataset$theta$mu,
                         dataset$theta$sigma)),
             summary(fit,
                  pars = c('pi1', 'A', 'mu', 'sigma'),
                  probs = c(0.10, 0.50, 0.90))$summary),
             col.names = c("True", "Mean", "MCSE", "SE",
                           "$q_{10\\%}$", "$q_{50\\%}$", "$q_{90\\%}$",
                           "ESS", "$\\hat{R}$"),
      digits = 2, align = "r",
      caption = "Estimated parameters and hidden quantities.
                 *MCSE = Monte Carlo Standard Error,
                 SE = Standard Error,
                 ESS = Effective Sample Size*.")
```

Table 2: Estimated parameters and hidden quantities. *MCSE = Monte Carlo Standard Error, SE = Standard Error, ESS = Effective Sample Size.*

	True	Mean	MCSE	SE	$q_{10\%}$	$q_{50\%}$	$q_{90\%}$	ESS	$\hat{R}$
pi1[1]	0.14	0.35	0.02	0.24	0.05	0.31	0.69	200.00	1.00
pi1[2]	0.38	0.29	0.02	0.22	0.04	0.26	0.61	200.00	1.01
pi1[3]	0.47	0.36	0.02	0.22	0.08	0.35	0.66	200.00	1.00
A[1,1]	0.03	0.02	0.00	0.02	0.00	0.02	0.05	198.89	1.00
A[1,2]	0.54	0.53	0.00	0.05	0.47	0.53	0.59	200.00	1.00
A[1,3]	0.43	0.45	0.00	0.05	0.38	0.45	0.50	200.00	1.00
A[2,1]	0.56	0.57	0.00	0.04	0.52	0.57	0.63	200.00	1.00
A[2,2]	0.31	0.30	0.00	0.04	0.24	0.30	0.35	200.00	1.00
A[2,3]	0.13	0.13	0.00	0.03	0.10	0.13	0.17	200.00	1.00
A[3,1]	0.20	0.17	0.00	0.05	0.12	0.17	0.24	200.00	1.00
A[3,2]	0.72	0.79	0.00	0.05	0.72	0.80	0.85	123.20	1.00
A[3,3]	0.07	0.03	0.00	0.02	0.01	0.03	0.07	78.75	1.00

	True	Mean	MCSE	SE	$q_{10\%}$	$q_{50\%}$	$q_{90\%}$	ESS	$\hat{R}$
mu[1]	8.94	9.16	0.01	0.14	8.98	9.15	9.34	200.00	1.00
mu[2]	18.73	18.64	0.03	0.29	18.29	18.63	18.99	107.77	1.00
mu[3]	29.23	29.52	0.01	0.17	29.30	29.52	29.76	200.00	1.00
sigma[1]	0.19	1.80	0.01	0.12	1.65	1.79	1.97	200.00	1.01
sigma[2]	3.65	3.98	0.02	0.24	3.67	3.97	4.30	100.08	1.01
sigma[3]	1.69	1.75	0.01	0.15	1.57	1.74	1.94	200.00	1.00

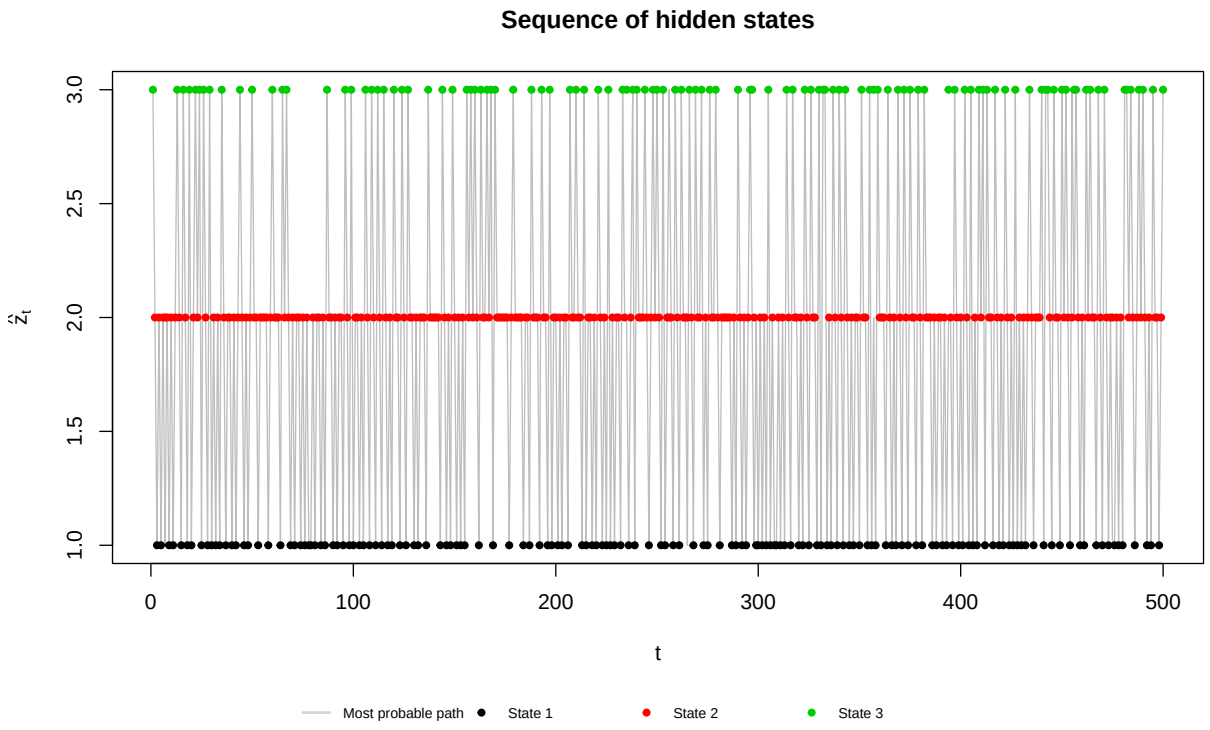
We extract the samples for some quantities of interest, namely the filtered probabilities vector α_t, the smoothed probability vector γ_t and the most probable hidden path $\mathbf{z}^*$. As an informal assessment that our software recover the hidden states correctly, we observe that the filtered probability, the smoothed probability and the most likely path are all reasonable accurate to the true values. As expected, the MAP estimate recovers the simulated hidden path with no label switching due because of the ordinal constraints.

```
alpha <- extract(fit, pars = 'alpha')[[1]]
gamma <- extract(fit, pars = 'gamma')[[1]]
```

```
alpha_med  <- apply(alpha, c(2, 3), function(x) { quantile(x, c(0.50)) })
alpha_hard <- apply(alpha_med, 1, which.max)

table(true = dataset$z, estimated = alpha_hard)
```

```
##     estimated
## true   1   2   3
##    1 155   0   0
##    2   6 226   1
##    3   0   5 107
```

```
zstar <- extract(fit, pars = 'zstar')[[1]]
plot_statepath(zstar, dataset$z)
```

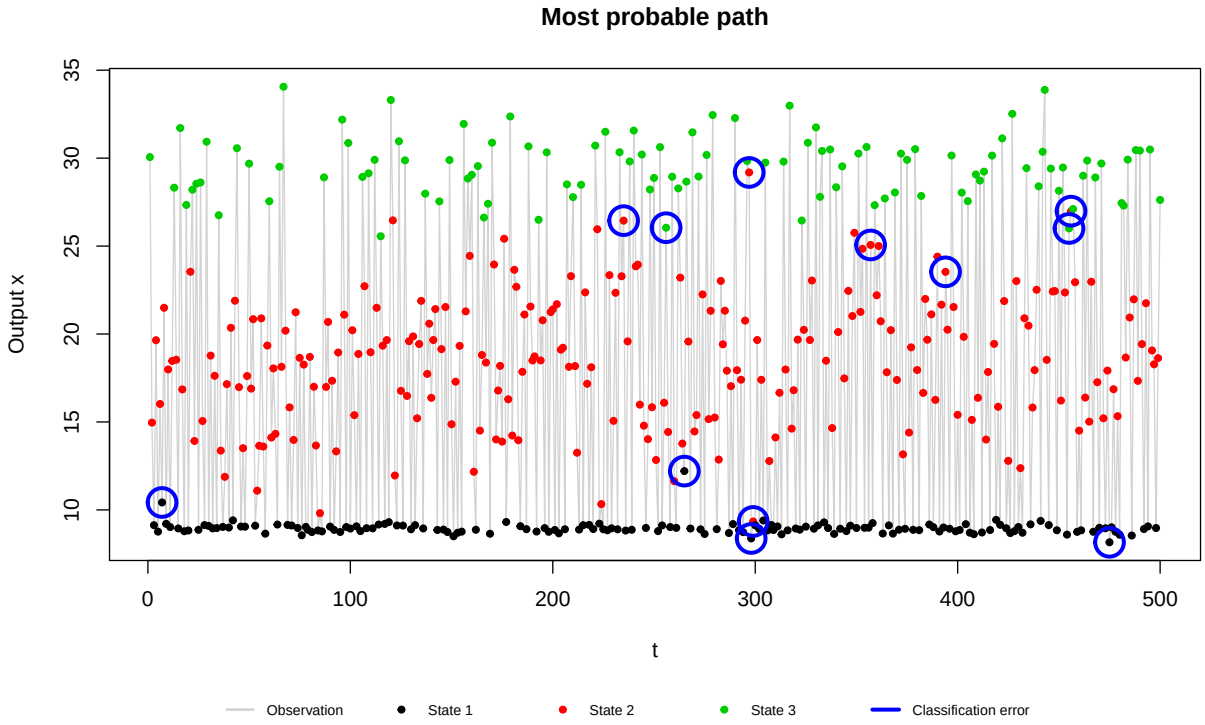



```
table(true = dataset$z, estimated = apply(zstar, 2, median))
```

```
##     estimated
## true   1   2   3
##    1 154   1   0
##    2   4 227   2
##    3   0   5 107
```

Finally, we plot the observed series colored according to the jointly most likely state. We identify an insignificant quantity of misclassifications product of the stochastic nature of our software.

```
plot_outputvit(x = dataset$y,
               z = dataset$z,
               zstar = zstar,
               main = "Most probable path")
```



2 The Input-Output Hidden Markov Model

The IOHMM is an architecture proposed by Bengio and Frasconi (1995) to map input sequences, sometimes called the control signal, to output sequences. It is a probabilistic framework that can deal with general sequence processing tasks such as production, classification and prediction. The main difference with Hidden Markov Models (HMM), which are part of the unsupervised learning paradigm, is the capability to learn the output sequence itself instead of the distribution of the output sequence.

2.1 Definitions

As with HMM, IOHMM involves two interconnected models,

$$z_t = f(z_{t-1}, \mathbf{u}_t)$$
$$\mathbf{y}_t = g(z_t, \mathbf{u}_t).$$

The first line corresponds to the state model, which consists of discrete-time, discrete-state hidden states $z_t \in \{1, \ldots, K\}$ whose transition depends on the previous hidden state z_{t-1} and the input vector $\mathbf{u}_t \in \mathbb{R}^M$. Additionally, the observation model is governed by $g(z_t, \mathbf{u}_t)$, where $\mathbf{y}_t \in \mathbb{R}^R$ is the vector of observations, emissions or output. The corresponding joint distribution is

$$p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T} | \mathbf{u}_t).$$

In the proposed parameterization with continuous inputs and outputs, the state model involves a multinomial regression whose parameters depend on the previous state taking the value i,

$$p(z_t|\mathbf{y}_t, \mathbf{u}_t, z_{t-1} = i) = \text{softmax}^{-1}(\mathbf{u}_t \mathbf{w}_i),$$

and the observation model is built upon a linear regression with Gaussian error and parameters depending on the current state taking the value j,

$$p(\mathbf{y}_t|\mathbf{u}_t, z_t = j) = \mathcal{N}(\mathbf{u}_t \mathbf{b}_j, \mathbf{\Sigma}_j)$$

IOHMM adapts the logic of HMM to allow for input and output vectors, retaining its fully probabilistic quality. Hidden states are assumed to follow a multinomial distribution that depends on the input sequence. The transition probabilities $\Psi_t(i, j) = p(z_t = j|z_{t-1} = i, \mathbf{u}_t)$, which govern the state dynamics, are driven by the control signal as well.

As for the output sequence, the local evidence at time t now becomes $\psi_t(j) = p(\mathbf{y}_t|z_t = j, \eta_t)$, where $\eta_t = E\langle \mathbf{y}_t|z_t, \mathbf{u}_t \rangle$ can be interpreted as the expected location parameter for the probability distribution of the emission $\mathbf{y}_t$ conditional on the input vector $\mathbf{u}_t$ and the hidden state z_t.

The actual form of the emission density $p(\mathbf{y}_t, \eta_t)$ can be discrete or continuous. In case of sequence classification or symbolic mutually exclusive emissions, it is possible to set up the multinomial distribution by running the softmax function over the estimated outputs of all possible states. In this case, we approximate continuous observations with the Gaussian density, the target is estimated as a linear combination of these outputs.

The adaptation of the data and parameters blocks is straightforward: we add the number of input variables `M`, the array of input vectors `u`, the regressors `b` and the residual standard deviation `sigma`.

```
data {
  int<lower=1> T;                   // number of observations (length)
  int<lower=1> K;                   // number of hidden states
  int<lower=1> M;                   // size of the input vector

  real y[T];                        // output (scalar so far)
  vector[M] u[T];                   // input vectors
}

parameters {
  // Discrete state model
  simplex[K] pi1;                   // initial state probabilities
  vector[M] w[K];                   // state regressors

  // Continuous observation model
  vector[M] b[K];                   // mean regressors
  real<lower=0> sigma[K];             // residual standard deviations
}
```

2.2 Inference

2.2.1 Filtering

The filtered marginals are computed recursively by adjusting the forward algorithm to consider the input sequence,

$$
\begin{aligned}
\alpha_t(j) &\triangleq p(z_t = j|\mathbf{y}_{1:t}, \mathbf{u}_{1:t}) \\
&= \sum_{i=1}^{K} p(z_t = j|z_{t-1} = i, \mathbf{y}_t, \mathbf{u}_t) p(z_{t-1} = i|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t-1}) \\
&= \sum_{i=1}^{K} p(\mathbf{y}_t|z_t = j, \mathbf{u}_t) p(z_t = j|z_{t-1} = i, \mathbf{u}_t) p(z_{t-1} = i|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t-1}) \\
&= \psi_t(j) \sum_{i=1}^{K} \Psi_t(i, j) \alpha_{t-1}(i).
\end{aligned}
$$

The implementation in Stan requires one modification: the time-dependent transition probability matrix is now computed as the linear combination of the input variables and the parameters of the multinomial regression that drives the latent process.

```
transformed parameters {
  vector[K] logalpha[T];

  vector[K] unA[T];
  vector[K] A[T];

  vector[K] logoblik[T];

  { // Transition probability matrix p(z_t = j | z_{t-1} = i, u)
    unA[1] = pi1; // Filler
    A[1] = pi1; // Filler
    for (t in 2:T) {
      for (j in 1:K) { // j = current (t)
        unA[t][j] = u[t]' * w[j];
      }
      A[t] = softmax(unA[t]);
    }
  }

  { // Evidence (observation likelihood)
    for(t in 1:T) {
      for(j in 1:K) {
        logoblik[t][j] = normal_lpdf(y[t] | mu[j], sigma[j]);
      }
    }
  }

  { // Forward algorithm log p(z_t = j | y_{1:t})
    real accumulator[K];

    for(j in 1:K)
      logalpha[1][j] = log(pi1[j]) + logoblik[1][j];

    for (t in 2:T) {
      for (j in 1:K) { // j = current (t)
        for (i in 1:K) { // i = previous (t-1)
                         // Murphy (2012) Eq. 17.48
                         // belief state + transition prob + local evidence at t
```

```
          accumulator[i] = logalpha[t-1, i] + log(A[t][i]) + logoblik[t][j];
        }
        logalpha[t, j] = log_sum_exp(accumulator);
      }
    }
  } // Forward
}
```

2.2.2 Smoothing

A smoother infers the posterior distribution of the hidden states at a given step based on all the observations or evidence,

$$\begin{aligned}\gamma_t(j) &\triangleq p(z_t = j|\mathbf{y}_{1:T}, \mathbf{u}_{1:T}) \\ &\propto \alpha_t(j)\beta_t(j),\end{aligned}$$

where

$$\beta_{t-1}(i) \triangleq p(\mathbf{y}_{t:T}|z_{t-1} = i, \mathbf{u}_{t:T}).$$

Similarly, inference about the smoothed posterior marginal requires the adaptation of the forward-backward algorithm to consider the input sequence in both components $\alpha_t(j)$ and $\beta_t(j)$. The latter now becomes

$$\begin{aligned}\beta_{t-1}(i) &\triangleq p(\mathbf{y}_{t:T}|z_{t-1} = i, \mathbf{u}_{t:T}) \\ &= \sum_{j=1}^{K} \psi_t(j)\Psi_t(i,j)\beta_t(j).\end{aligned}$$

Once adjusted the transition probability matrix, the Stan code for the forward-backward algorithm need no further modification.

2.2.3 MAP: Viterbi

The Stan code for the Viterbi decoding algorithm need no further modification.

2.3 Parameter estimation

The parameters of the models are $\theta = (\pi_1, \theta_h, \theta_o)$, where π_1 is the initial state distribution, θ_h are the parameters of the hidden model and θ_o are the parameters of the state-conditional density function $p(\mathbf{y}_t|z_t = j, \mathbf{u}_t)$. State transition is characterized by a multinomial regression with parameters $\mathbf{w}_k$ for $k \in \{1, \ldots, K\}$, while emissions are modeled by a linear regression with Gaussian error and parameters $\mathbf{b}_k$ and $\mathbf{\Sigma}_k$ for $k \in \{1, \ldots, K\}$.

Estimation can be run under both the maximum likelihood and Bayesian frameworks. Although it is a straightforward procedure when the data is fully observed, in practice the latent states $\mathbf{z}_{1:T}$ are hidden. The most common approach is the application of the EM algorithm to find either the maximum likelihood or the maximum a posteriori estimates. Bengio and Frasconi (1995) proposes a straightforward modification of the

EM algorithm. The application of sigmoidal functions, for example the logistic or softmax transforms for the hidden transition model, requires numeric optimization via gradient ascent or similar methods for the M step. In this work, we exploit Stan's capabilities to produce a sampler that explores the posterior density of the model parameters.

2.4 A walk-though

2.4.1 Numerical stability for the softmax function

A digression! The softmax function, or normalized exponential function, can suffer from over or underflow in the exponentials. A naive implementation may fail:

```
x <- 10^(1:5)
exp(x) / sum(exp(x))
```

```
## [1]   0   0 NaN NaN NaN
```

A well-known, safer implementation exploits the fact that softmax is location invariant, ie $softmax(\mathbf{y}) = softmax(\mathbf{y} + c)$ for any constant c. Subtracting the maximum value produces a new vector with non-positive entries, ruling out overflows, and at least one zero element, guaranteeing at least one significant term in the denominator.

```
logsumexp <- function(x) {
  y = max(x)
  y + log(sum(exp(x - y)))
}

softmax <- function(x) {
  exp(x - logsumexp(x))
}

softmax(x)
```

```
## [1] 0 0 0 0 1
```

This is already taken care in Stan (Stan Development Team 2017c, 474).

2.4.2 Back to the walk-though

We first adapt the R routine for our new generative model. The arguments are the sequence length T, the number of discrete hidden states K, the input matrix $\mathbf{u}$, the initial state distribution vector π_1, a matrix with the parameters of the multinomial regression that rules the hidden states dynamics $\mathbf{w}$, the name of a function drawing samples from the observation distribution and its arguments.

The initial hidden state is drawn from a multinomial distribution with one trial and event probabilities given by the initial state probability vector π_1. The latent states for each of the following steps $t \in \{2, \dots, T\}$ are generated from a multinomial regression with vector parameters $\mathbf{w}_k$, one set per possible hidden state $k \in \{1, \dots, K\}$, and covariates $\mathbf{u}_t$. The hidden states are subsequently sampled based on these transition probabilities.

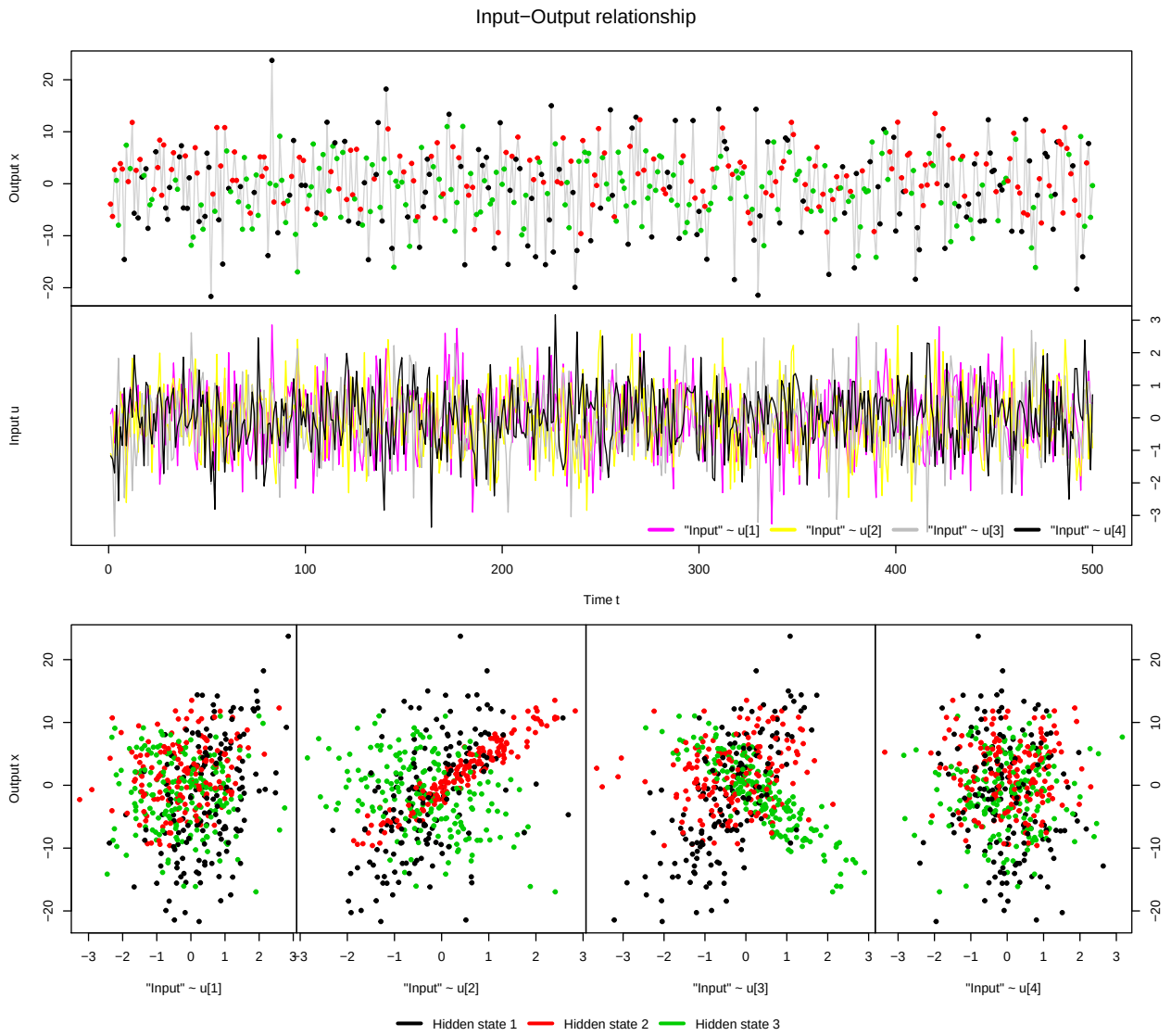
The observation at each step may generate from a Gaussian with parameters μ_k and σ_k, one set per possible hidden state.

```
iohmm_generate <- function(T) {
  # 1. Parameters
  K <- 3
```

```
  M <- 4
  u <- matrix(rnorm(T * M), nrow = T, ncol = M, byrow = TRUE)
  w <- matrix(
    c(1.2, 0.5, 0.3, 0.1, 0.5, 1.2, 0.3, 0.1, 0.5, 0.1, 1.2, 0.1),
    nrow = K, ncol = M, byrow = TRUE)
  b <- matrix(
    c(5.0, 6.0, 7.0, 0.5, 1.0, 5.0, 0.1, -0.5, 0.1, -1.0, -5.0, 0.2),
    nrow = K, ncol = M, byrow = TRUE)
  sigma <- c(0.2, 1.0, 2.5)
  pi1 <- c(0.4, 0.2, 0.4)

  p.mat <- matrix(0, nrow = T, ncol = K)
  p.mat[1, ] <- pi1

  # 2. Hidden path
  z <- vector("numeric", T)
  z[1] <- sample(x = 1:K, size = 1, replace = FALSE, prob = pi1)
  for (t in 2:T) {
    p.mat[t, ] <- softmax(sapply(1:K, function(j) {u[t, ] %*% w[j, ]}))
    z[t] <- sample(x = 1:K, size = 1, replace = FALSE, prob = p.mat[t, ])
  }

  # 3. Observations
  y <- vector("numeric", T)
  for (t in 1:T) {
    y[t] <- rnorm(1, u[t, ] %*% b[z[t], ], sigma[z[t]])
  }

  list(
    u = u,
    z = z,
    y = y,
    theta = list(pi1 = pi1, w = w,
                 b = b, sigma = sigma, p.mat = p.mat)
  )
}
```

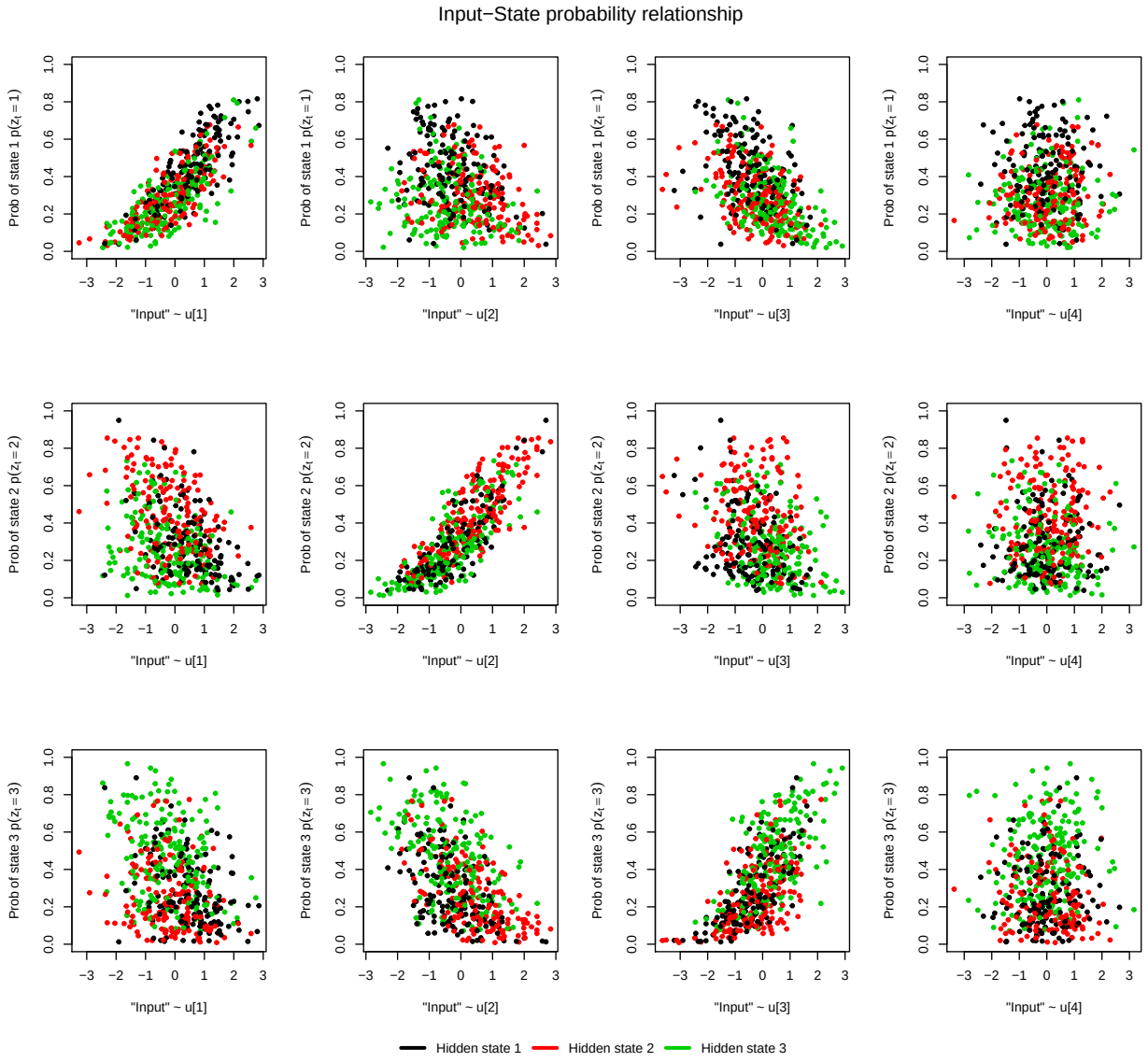
We draw one sample of length $T = 500$ from a data generating process with $K = 3$ latent states and run an exploratory analysis of the observed quantities.

```
set.seed(900)
K <- 3
T_length <- 500
dataset  <- iohmm_generate(T_length)

plot_inputoutput(x = dataset$y, u = dataset$u, z = dataset$z)
```



We observe how the chosen values for the parameters affect the generated data. For example, the relationship between the third input $\mathbf{u}_3$ and the output $\mathbf{y}_t$ is positive, indifferent and negative for the hidden states $K = 1, 2, 3$ respectively. The true slopes are 7, 0.1 and -5.

```
plot_inputprob(u = dataset$u, p.mat = dataset$theta$p.mat, z = dataset$z)
```



We then analyse the relationship between the input and the state probabilities, which are usually hidden in applications with real data. The pairs $\{\mathbf{u}_1, p(z_t = 1)\}$, $\{\mathbf{u}_2, p(z_t = 2)\}$ and $\{\mathbf{u}_3, p(z_t = 3)\}$ show the strongest relationships because of values of true regression parameters: those inputs take the largest weight in each state, namely $w_{11} = 1.2$, $w_{22} = 1.2$ and $w_{33} = 1.2$.

We run the software to draw samples from the posterior density of model parameters and other hidden quantities.

```
iohmm_fit <- function(K, u, y) {
  rstan_options(auto_write = TRUE)
  options(mc.cores = parallel::detectCores())

  stan.model = 'stan/iohmm_reg.stan'
  stan.data = list(
    T = nrow(u),
    K = K,
    M = ncol(u),
    u = as.array(u),
```

```
    y = y
  )

  stan(file = stan.model,
       data = stan.data, verbose = T,
       iter = 400, warmup = 200,
       thin = 1, chains = 1,
       cores = 1, seed = 900)
}

fit <- iohmm_fit(K, dataset$u, dataset$y)
```

```
##
## TRANSLATING MODEL 'iohmm_reg' FROM Stan CODE TO C++ CODE NOW.
## successful in parsing the Stan model 'iohmm_reg'.
##
## CHECKING DATA AND PREPROCESSING FOR MODEL 'iohmm_reg' NOW.
##
## COMPILING MODEL 'iohmm_reg' NOW.
##
## STARTING SAMPLER FOR MODEL 'iohmm_reg' NOW.
##
## SAMPLING FOR MODEL 'iohmm_reg' NOW (CHAIN 1).
##
## Gradient evaluation took 0.002 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 20 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:   1 / 400 [  0%]  (Warmup)
## Iteration:  40 / 400 [ 10%]  (Warmup)
## Iteration:  80 / 400 [ 20%]  (Warmup)
## Iteration: 120 / 400 [ 30%]  (Warmup)
## Iteration: 160 / 400 [ 40%]  (Warmup)
## Iteration: 200 / 400 [ 50%]  (Warmup)
## Iteration: 201 / 400 [ 50%]  (Sampling)
## Iteration: 240 / 400 [ 60%]  (Sampling)
## Iteration: 280 / 400 [ 70%]  (Sampling)
## Iteration: 320 / 400 [ 80%]  (Sampling)
## Iteration: 360 / 400 [ 90%]  (Sampling)
## Iteration: 400 / 400 [100%]  (Sampling)
##
##  Elapsed Time: 111.748 seconds (Warm-up)
##                76.025 seconds (Sampling)
##                187.773 seconds (Total)
```

We rely on several diagnostic statistics and plots provided by rstan (Stan Development Team 2017a) and shinystan (Stan Development Team 2017b) to assess mixing, convergence and the absence of divergences. Label switching hinders the comparison between true and observed parameters.

```
knitr::kable(cbind(
             unlist(list(dataset$theta$pi1,
                         t(dataset$theta$w),
                         dataset$theta$b,
                         dataset$theta$sigma)),
```

```
        summary(fit,
             pars = c('pi1', 'w', 'b', 'sigma'),
             probs = c(0.10, 0.50, 0.90))$summary),
        col.names = c("True", "Mean", "MCSE", "SE",
                      "$q_{10\\%}$", "$q_{50\\%}$", "$q_{90\\%}$",
                      "ESS", "$\\hat{R}$"),
  digits = 2, align = "r",
  caption = "Estimated parameters and hidden quantities.
           *MCSE = Monte Carlo Standard Error,
           SE = Standard Error,
           ESS = Effective Sample Size*.")
```

Table 3: Estimated parameters and hidden quantities. *MCSE = Monte Carlo Standard Error, SE = Standard Error, ESS = Effective Sample Size.*

	True	Mean	MCSE	SE	$q_{10\%}$	$q_{50\%}$	$q_{90\%}$	ESS	$\hat{R}$
pi1[1]	0.4	0.27	0.01	0.20	0.04	0.24	0.56	200.00	1.00
pi1[2]	0.2	0.25	0.01	0.19	0.05	0.20	0.56	200.00	1.00
pi1[3]	0.4	0.48	0.02	0.22	0.19	0.48	0.76	200.00	1.00
w[1,1]	1.2	-0.33	0.25	2.66	-3.81	-0.56	3.15	113.42	1.00
w[1,2]	0.5	-0.06	0.24	3.03	-4.43	0.06	3.73	158.28	1.00
w[1,3]	0.3	0.06	0.20	2.78	-3.48	0.08	3.54	184.68	1.00
w[1,4]	0.1	-0.23	0.26	3.09	-4.04	-0.16	3.94	140.91	1.00
w[2,1]	0.5	-0.34	0.25	2.67	-3.76	-0.53	3.22	113.70	1.00
w[2,2]	1.2	-0.15	0.24	3.03	-4.39	-0.05	3.67	158.29	1.00
w[2,3]	0.3	0.06	0.20	2.76	-3.54	0.13	3.70	185.29	1.00
w[2,4]	0.1	-0.10	0.26	3.10	-3.98	-0.03	4.19	141.38	1.00
w[3,1]	0.5	-0.54	0.25	2.66	-3.92	-0.68	2.96	115.20	1.00
w[3,2]	0.1	-0.10	0.24	3.03	-4.43	-0.07	3.78	158.11	1.00
w[3,3]	1.2	0.05	0.20	2.76	-3.48	0.19	3.60	187.16	1.00
w[3,4]	0.1	-0.17	0.26	3.08	-3.99	-0.07	3.96	141.51	1.00
b[1,1]	5.0	-0.05	0.01	0.19	-0.31	-0.05	0.18	200.00	1.01
b[1,2]	1.0	-1.08	0.01	0.18	-1.32	-1.09	-0.84	200.00	1.00
b[1,3]	0.1	-4.93	0.01	0.20	-5.17	-4.94	-4.67	200.00	1.00
b[1,4]	6.0	0.15	0.02	0.21	-0.13	0.15	0.44	200.00	0.99
b[2,1]	5.0	4.98	0.00	0.02	4.96	4.98	5.01	200.00	1.00
b[2,2]	-1.0	5.99	0.00	0.02	5.96	5.99	6.01	200.00	1.00
b[2,3]	7.0	7.00	0.00	0.02	6.98	7.00	7.02	200.00	1.00
b[2,4]	0.1	0.49	0.00	0.02	0.47	0.49	0.52	200.00	1.00
b[3,1]	-5.0	0.98	0.01	0.10	0.85	0.98	1.12	200.00	1.00
b[3,2]	0.5	5.00	0.01	0.09	4.88	5.00	5.12	200.00	1.00
b[3,3]	-0.5	0.09	0.01	0.08	-0.02	0.09	0.20	200.00	1.00
b[3,4]	0.2	-0.61	0.01	0.09	-0.73	-0.61	-0.50	200.00	1.00
sigma[1]	0.2	2.59	0.01	0.14	2.42	2.59	2.78	200.00	1.00
sigma[2]	1.0	0.20	0.00	0.01	0.18	0.20	0.22	200.00	1.00
sigma[3]	2.5	1.07	0.00	0.07	0.99	1.06	1.16	200.00	1.00

While mixing and convergence is extremely efficient, as expected when dealing with generated data, we note that the regression parameters for the latent states are the worst performers. The smaller effective size translates into higher Monte Carlo standard error and broader posterior intervals. One possible reason is that softmax is invariant to change in location, thus the parameters of a multinomial regression do not have

a natural center and become harder to estimate.

We assess that our software recover the hidden states correctly. Due to label switching, the states generated under the labels 1 through 3 were recovered in a different order. In consequence, we decide to relabel the observations based on the best fit. This would not prove to be a problem with real data as the hidden states are never observed.

```
# Relabeling (ugly hack edition) ----------------------------------------
alpha <- extract(fit, pars = 'alpha')[[1]]
dataset$zrelab <- rep(0, T_length)

hard <- sapply(1:T_length, function(t, med) {
  which.max(med[t, ])
}, med = apply(alpha, c(2, 3),
                    function(x) {
                      quantile(x, c(0.50)) }))

tab <- table(hard = hard, original = dataset$z)

for (k in 1:K) {
  dataset$zrelab[which(dataset$z == k)] <- which.max(tab[, k])
}

table(new = dataset$zrelab, original = dataset$z)
```

```
##    original
## new   1   2   3
##   1   0   0 183
##   2 152   0   0
##   3   0 165   0
```

The confusion matrix makes evident that, under ideal conditions, the sampler works as intended.

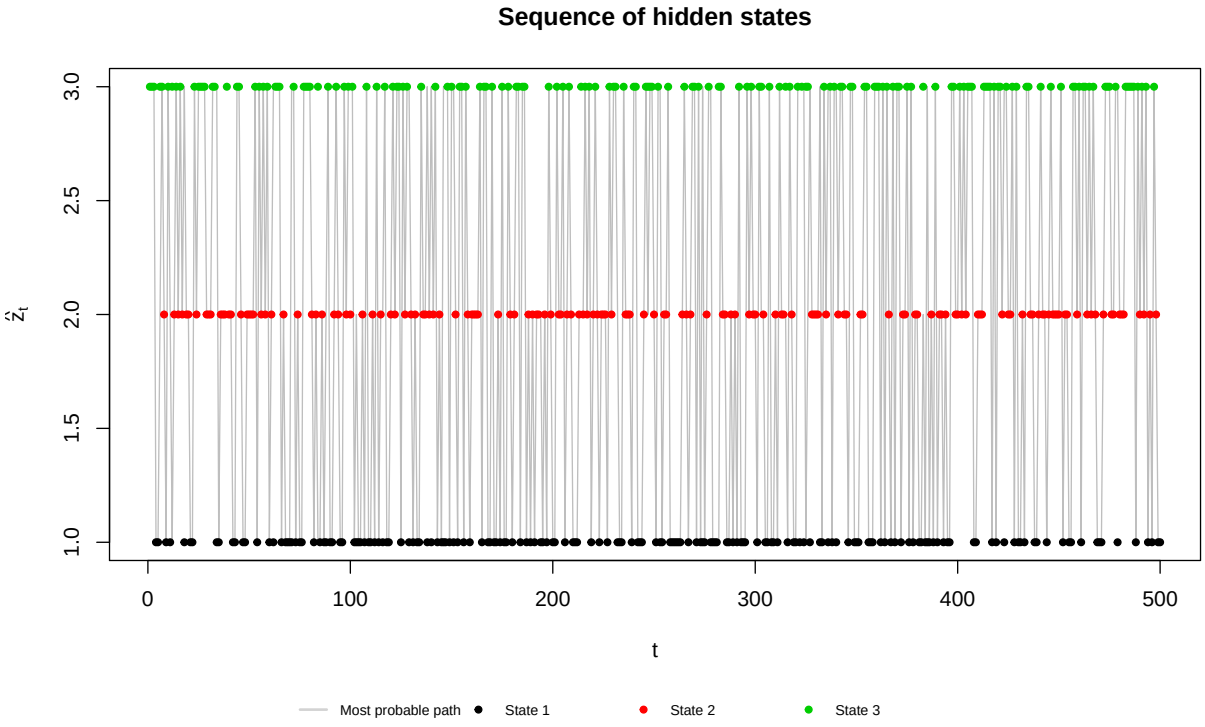
```
knitr::kable(table(
        estimated = apply(round(apply(alpha, c(2, 3),
                                      function(x) {
                                        quantile(x, c(0.50)) })), 1, which.max),
        real = dataset$zrelab),
      col.names = c("Real 1", "Real 2", "Real 3"),
      row.names = TRUE,
      digits = 2, align = "r", caption = "Hard classification.")
```

Table 4: Hard classification.

	Real 1	Real 2	Real 3
1	155	1	6
2	6	151	7
3	22	0	152

Similarly, the Viterbi algorithm recovers the expected most probably hidden state.

```
zstar <- extract(fit, pars = 'zstar')[[1]]
plot_statepath(zstar, dataset$zrelab)
```




```
table(true = dataset$z, estimated = apply(zstar, 2, median))
```

```
##     estimated
## true   1   2   3
##    1   0 151   1
##    2   4   8 153
##    3 155   6  22
```

3 Acknowledgements

We acknowledge the members of the Stan Development Team for being so passionate about Bayesian statistics and technically unbeatable. Special mentions to Aaron Goodman, Ben Bales and Bob Carpenter for their active participation in the discussions held in Stan forums for HMM with constraints and HMMM. Although not strictly related, the discussion remains very valuable for the current tutorial.

We acknowledge the Google Summer Of Code (GSOC) program for funding. This tutorial builds on top of our project: Bayesian Hierarchical Hidden Markov Models applied to financial time series.

---

4 Original Computing Environment

```
## Warning in readLines(file.path(Sys.getenv("HOME"), ".R/Makevars")):
## incomplete final line found on 'C:\Users\Bebop/.R/Makevars'
```

```
## CXXFLAGS=-O3 -Wno-unused-variable -Wno-unused-function
```

```
## Session info -------------------------------------------------------------
```

```
##  setting  value
##  version  R version 3.4.1 (2017-06-30)
##  system   x86_64, mingw32
##  ui       RTerm
##  language (EN)
##  collate  Spanish_Argentina.1252
##  tz       America/Buenos_Aires
##  date     2018-01-09
```

```
## Packages -------------------------------------------------------------------
```

```
##  package      * version   date       source
##  BH             1.65.0-1  2017-08-24 CRAN (R 3.4.1)
##  colorspace     1.3-2     2016-12-14 CRAN (R 3.4.1)
##  dichromat      2.0-0     2013-01-24 CRAN (R 3.4.1)
##  digest         0.6.12    2017-01-27 CRAN (R 3.4.1)
##  ggplot2      * 2.2.1     2016-12-30 CRAN (R 3.4.1)
##  graphics     * 3.4.1     2017-06-30 local
##  grDevices    * 3.4.1     2017-06-30 local
##  grid           3.4.1     2017-06-30 local
##  gridExtra      2.3       2017-09-09 CRAN (R 3.4.1)
##  gtable         0.2.0     2016-02-26 CRAN (R 3.4.1)
##  inline         0.3.14    2015-04-13 CRAN (R 3.4.1)
##  labeling       0.3       2014-08-23 CRAN (R 3.4.1)
##  lattice        0.20-35   2017-03-25 CRAN (R 3.4.1)
##  lazyeval       0.2.0     2016-06-12 CRAN (R 3.4.1)
##  magrittr       1.5       2014-11-22 CRAN (R 3.4.1)
##  MASS           7.3-47    2017-02-26 CRAN (R 3.4.1)
##  Matrix         1.2-10    2017-05-03 CRAN (R 3.4.1)
##  methods      * 3.4.1     2017-06-30 local
##  munsell        0.4.3     2016-02-13 CRAN (R 3.4.1)
##  plyr           1.8.4     2016-06-08 CRAN (R 3.4.1)
##  R6             2.2.2     2017-06-17 CRAN (R 3.4.1)
##  RColorBrewer   1.1-2     2014-12-07 CRAN (R 3.4.1)
##  Rcpp           0.12.12   2017-07-15 CRAN (R 3.4.1)
##  RcppEigen      0.3.3.3.0 2017-05-01 CRAN (R 3.4.1)
##  reshape2       1.4.2     2016-10-22 CRAN (R 3.4.1)
##  rlang          0.1.2     2017-08-09 CRAN (R 3.4.1)
##  rstan        * 2.16.2    2017-07-03 CRAN (R 3.4.1)
##  scales         0.5.0     2017-08-24 CRAN (R 3.4.1)
##  StanHeaders  * 2.16.0-1  2017-07-03 CRAN (R 3.4.1)
##  stats        * 3.4.1     2017-06-30 local
##  stats4         3.4.1     2017-06-30 local
##  stringi        1.1.5     2017-04-07 CRAN (R 3.4.1)
##  stringr        1.2.0     2017-02-18 CRAN (R 3.4.1)
##  tibble         1.3.4     2017-08-22 CRAN (R 3.4.1)
##  tools          3.4.1     2017-06-30 local
##  utils        * 3.4.1     2017-06-30 local
##  viridisLite    0.2.0     2017-03-24 CRAN (R 3.4.1)
```

5 References

Bakis, Raimo. 1976. "Continuous Speech Recognition via Centisecond Acoustic States." *The Journal of the Acoustical Society of America* 59 (S1). ASA: S97–S97.

Baum, Leonard E, and Ted Petrie. 1966. "Statistical Inference for Probabilistic Functions of Finite State Markov Chains." *The Annals of Mathematical Statistics* 37 (6). JSTOR: 1554–63.

Baum, Leonard E, and George Sell. 1968. "Growth Transformations for Functions on Manifolds." *Pacific Journal of Mathematics* 27 (2). Mathematical Sciences Publishers: 211–27.

Baum, Leonard E. 1972. "An Inequality and Associated Maximaization Technique in Stattistical Estimation for Probablistic Functions of Markov Process." *Inequalities* 3: 1–8.

Baum, Leonard E., and J. A. Eagon. 1967. "An Inequality with Applications to Statistical Estimation for Probabilistic Functions of Markov Processes and to a Model for Ecology." *Bulletin of the American Mathematical Society* 73 (3). American Mathematical Society (AMS): 360–64. doi:10.1090/s0002-9904-1967-11751-8.

Baum, Leonard E., Ted Petrie, George Soules, and Norman Weiss. 1970. "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains." *The Annals of Mathematical Statistics* 41 (1). Institute of Mathematical Statistics: 164–71. doi:10.1214/aoms/1177697196.

Bengio, Yoshua, and Paolo Frasconi. 1995. "An Input Output Hmm Architecture."

Betancourt, Michael. 2017. "Identifying Bayesian Mixture Models." *Stan Case Studies*, no. Volume 4. http://mc-stan.org/users/documentation/case-studies/identifying_mixture_models.html.

Carpenter, Bob, Andrew Gelman, Matt Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Michael A Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. "Stan: A Probabilistic Programming Language." *Journal of Statistical Software* 76 (1). doi:10.18637/jss.v076.i01.

Gelman, Andrew, and Donald B Rubin. 1992. "Inference from Iterative Simulation Using Multiple Sequences." *Statistical Science*. JSTOR, 457–72.

Jelinek, Frederick. 1976. "Continuous Speech Recognition by Statistical Methods." *Proceedings of the IEEE* 64 (4). IEEE: 532–56.

Jordan, Michael I. 2003. "An Introduction to Probabilistic Graphical Models." preparation.

Murphy, Kevin P. 2012. *Machine Learning*. MIT Press Ltd.

Rabiner, Lawrence R. 1990. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition." In *Readings in Speech Recognition*, 267–96. Elsevier. doi:10.1016/b978-0-08-051584-7.50027-9.

Stan Development Team. 2017a. "RStan: The R Interface to Stan." http://mc-stan.org/.

———. 2017b. "Shinystan: Interactive Visual and Numerical Diagnostics and Posterior Analysis for Bayesian Models." http://mc-stan.org/.

———. 2017c. *Stan Modeling Language: User's Guide and Reference Manual: Version 2.15.0.*

Viterbi, A. 1967. "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm." *IEEE Transactions on Information Theory* 13 (2). Institute of Electrical; Electronics Engineers (IEEE): 260–69. doi:10.1109/tit.1967.1054010.