APPENDICES

A PROBABILISTIC PROGRAMMING

In fig. 6, we present an HMM as a toy model. Using this, we briefly discuss the idea behind SMC inference.

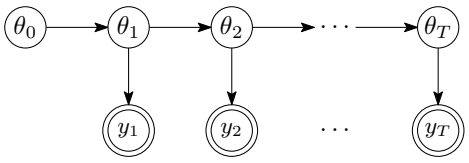
By way of motivation, consider the generative model $\mathbb{P}(\theta_{1:T}, y_{1:T})$ with hidden variables $\theta_{1:T}$ and observations $y_{1:T}$. In PP, we let the observing random variable y_t be the value of the t^{th} observe, and the hidden variables $\boldsymbol{\theta}_t = \theta_{1:t}$ be the execution trace before this observe (see fig. 7). The goal of, e.g., SMC inference in PP is to approximate $\mathbb{P}(\boldsymbol{\theta}_{1:T} \mid y_{1:T}) = \mathbb{P}(y_{1:t})^{-1}\mathbb{P}(\theta_{1:t}, y_{1:t})\ \forall t \in \{1, \ldots, T\}$. The goal of SMC inference in Anglican is to approximate $\mathbb{P}(\ \theta_{1:T} \mid y_{1:T}) = \mathbb{P}(y_{1:t})^{-1}\mathbb{P}(\theta_{1:t}, y_{1:t})\ \forall t \in \{1, \ldots, T\}$. The weighted approximation to $\mathbb{P}(\theta_{1:T} \mid y_{1:T})$, is achieved by generating a set of particles and importance weights at each time step t: $\{(\theta_{1:n}^{(p)}, w_t^{(p)})\}_{p=1}^{P}$. The approximation to the target distribution is then given by

$$\sum_{p=1}^{P} w_t^{(p)} \delta_{\boldsymbol{\theta}_{1:t}^{(p)}}(\boldsymbol{\theta}_{1:T}). \tag{9}$$

At each time step t particles are generated using a chosen proposal density $q_t(\boldsymbol{\theta}_t \mid \boldsymbol{\theta}_{t-1})$. This is used to propose new particles, given the set of particles $\{\bar{\boldsymbol{\theta}}_{1:t-1}^{(p)}\}$ re-sampled from the previous $t-1$ steps of the SMC estimate of $\mathbb{P}(\boldsymbol{\theta}_{1:t-1} \mid y_{1:t-1})$. The weights of the corresponding particles are given as follows:

$$W_t^{(p)} = \frac{\mathbb{P}\left(\boldsymbol{\theta}_t^{(p)} \mid \bar{\boldsymbol{\theta}}_{1:t-1}^{(p)}\right)\mathbb{P}\left(y_t \mid \boldsymbol{\theta}_t^{(p)}\right)}{q\left(\boldsymbol{\theta}_t^{(p)} \mid \bar{\boldsymbol{\theta}}_{1:t-1}^{(p)}\right)}, \tag{10}$$

and $w_t^{(p)} = W_t^{(p)}\left(\sum_{i=1}^{P} W_t^{(p)}\right)^{-1}$ for $p \in \{1, \ldots, P\}$. Finally, for each t in the proposal, re-sampling and re-weighting steps are iterated. This gives us a particle estimate of our target posterior along with an estimate of the marginal likelihood $\mathbb{P}(y_{1:T})$.



(a) HMM graphical model

```
(defquery hmm
  [observations initial-dist
      transition-dists observation-dists]
  (reduce
    (fn [states observation]
      (let [state (sample (get
          transition-dists (peek states)))]
        (observe (get observation-dists
            state) observation)
        (conj states state)))
    [(sample initial-dist)]
    observations))
```

(b) HMM in Anglican syntax.

Figure 6: The probabilistic programming philosophy illustrated; showing in (a) the graphical model and in (b) the resultant code upon which black-box inference is fully applicable.

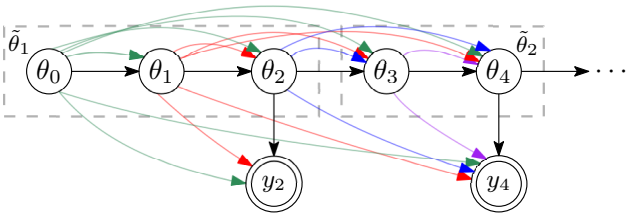
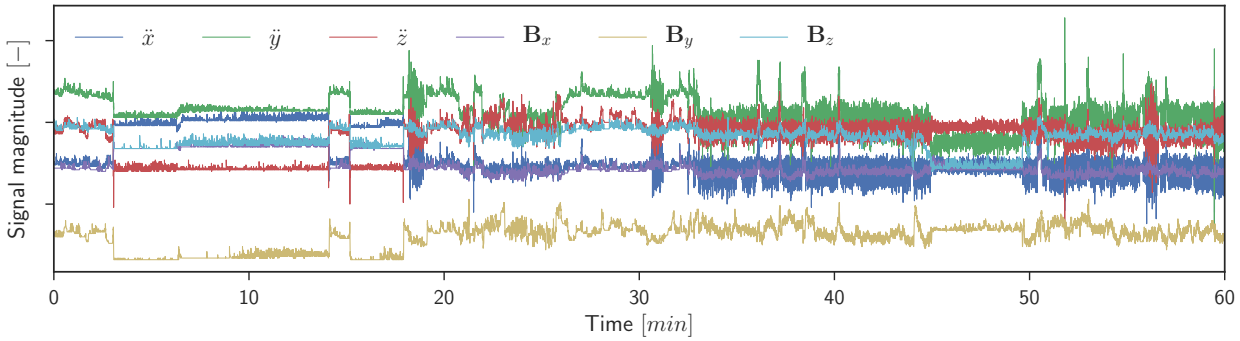
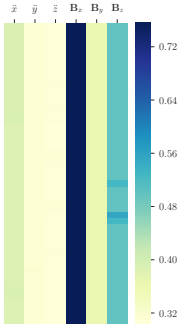


Figure 7: Model traces illustration, adapted from [31]. The execution trace is bounded by the dashed box, wherein $\tilde{\theta}_1 = \theta_1 \times \theta_2 \times \theta_3$ for example.

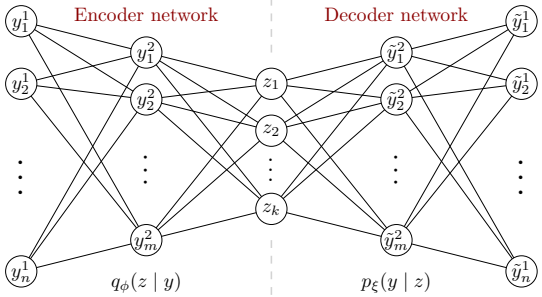
B NONLINEAR FEATURE LEARNING



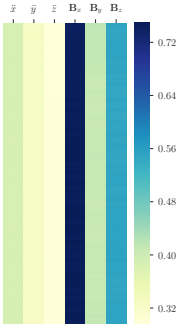
(a) One hour of raw data captured at 32Hz, containing a total of 115,200 multivariate observations.



(b) Example of original $3s$ feature (96×6).



(c) General structure of variational autoencoder.



(d) Reconstructed example feature.

Figure 8: The top panel shows the raw data used for this study, consisting of tri-axial accelerometer and magnetometer readings taken from the lion's collar. The bottom figures display the variational autoencoder framework, with original (b) and reconstructed (d) features shown.

In this study, we used a VAE to perform unsupervised feature learning - see fig. 8c. It is on these learned features that we perform our further analyses using state space models implemented in Anglican.

Traditional AEs are models designed to output a reconstruction of their input. VAEs, by extension, ensure that the transformations to or from the hidden representation are useful by sacrificing some fidelity, by imposing some sort of regularization or constraint. VAEs consist of an encoder, a decoder and a loss function. The encoder $q_\phi(z \mid y)$ consists of a neural network. It takes as input an observation y and outputs a latent representation z, and has weights and biases ϕ. The decoder $p_\xi(y \mid z)$ is similarly a neural net. Its input is the latent representation z, it outputs the parameters to the probability distribution of the observations, with weights and biases ξ. The encoder must learn an efficient compression of the data into a lower-dimensional representation z. Because q_ϕ is a Gaussian density, we can sample noisy representations of z. Conversely the decoder p_ξ takes the latent representation z and tries to decode the original input y. Throughout this computational flow, information will be lost, the magnitude of which we measure with the reconstruction log-likelihood $\log p_\xi(y \mid z)$, given by

$$\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{y})}[\log p_\xi(\mathbf{y}|\mathbf{z})] - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{y})||p_\xi(\mathbf{z})) \tag{11}$$

where bold variables contain all data points. This measure tells us how effectively the decoder has learned to reconstruct the input y, given its latent representation z. Consequently we can train the VAE using gradient descent to optimise the loss with respect to the parameters of the encoder and decoder.

Apart from giving us a low-dimensional representation of our sequential observation, the VAE construction lends itself further to our methodological design. We assume that our observation model is Gaussian in our generative modelling framework. The VAE, by construction, yields latent variables which are normally distributed and it is these that we go on to nonparametrically segment. The raw data is *not* normally distributed.

C MODEL PARAMETERS

Table 1: Common model and emission priors used for synthetic data experiments.

Model parameter	**Prior**
$\alpha, \gamma, \alpha', \gamma'$	$\Gamma(1,1)$
κ	$\Gamma(2,1)$
H_θ	$\mathcal{N}(0,1)$
$H_\mathcal{D}$	$\mathcal{U}(25,100), \mathcal{N}(\mu,\Sigma)^\dagger, \text{Pois}(\lambda)^\dagger$
μ_0	$\overline{\mathbf{Y}}$ (empirical average)
λ_0	$D+2$
Ψ	$0.75 \times \text{Cov}(\mathbf{Y})$
ν	0.1

† Used as part of parametric mixture duration distribution

Table 2: Experimental model and emission priors, used for inter-model comparison.

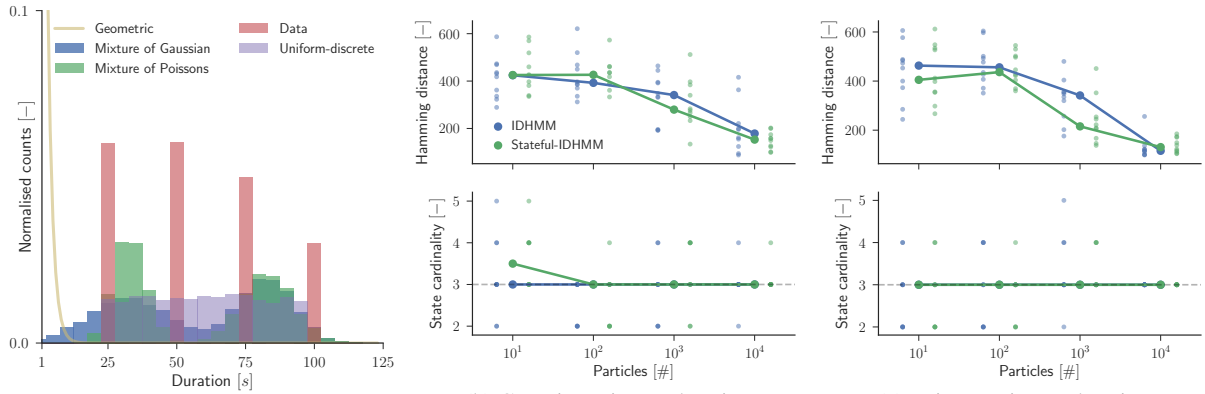
Model parameter	**Prior**
α, γ	$\Gamma(0.5,1)$
α', γ'	$\Gamma(1,1)$
κ	$\Gamma(5,1)$
H_θ	$\mathcal{N}(0,1)$
$H_\mathcal{D}$	$\mathcal{U}(1000)$
μ_0	$\overline{\mathbf{Y}}$
λ_0	$D+2$
Ψ	$C \times \text{Cov}(\mathbf{Y})$
C	$\mathcal{U}(0.5,2)$
ν	$\mathcal{U}(0.1,2.0)$

Table 3: Experimental model and emission priors, used for detailed analysis of hunt segment.

Model parameter	**Prior**
α, γ	$\Gamma(1,5)$
α', γ'	$\Gamma(2,1)$
H_θ	$\mathcal{N}(0,1)$
$H_\mathcal{D}$	$\text{Pois}(\lambda)^\dagger$
μ_0	$\overline{\mathbf{Y}}$
λ_0	$D+2$
Ψ	$C \times \text{Cov}(\mathbf{Y})$
C	$\Gamma(5,5)$
ν	$\Gamma(1,0.75)$

† Used as part of parametric mixture duration distribution

D BENEFIT OF NONPARAMETRIC DURATION DISTRIBUTIONS



(a) Duration distribution

(b) Gaussian mixture duration

(c) Poisson mixture duration

Figure 9: Different durations models used as priors, as well as the observations' duration model, are shown in fig. 9a. Our two contributed models are compared, under different durations priors, in fig. 9b and fig. 9c. Shaded small bullets, correspond to the specific experimental results (a total of ten trials were conducted for each particle count, for each model). The solid large bullets correspond to the median error across all trials in each particle count, for that model.

E RELATED MODELS

Hitherto we have alluded to two models which most resemble the construction which we propose: the HDP-HSMM of Johnson & Willsky [11] - see fig. 2c - and the infinite explicit duration HMM (IED-HMM) of Huggins & Wood [10]. We briefly describe these models and compare them to our contribution.

The HDP-HSMM can be seen as similar to the IDHMM with the two key differences that 1) the duration distributions are chosen from a parametric family rather than modelled non-parametrically and 2) the top level DP for state transitions is implemented through a stick-breaking construction. In particular, the model is given by

$$\begin{aligned}
\beta \mid \gamma &\sim \text{GEM}(\gamma) \\
\pi_i \mid \beta, \alpha &\overset{i.i.d.}{\sim} \mathcal{DP}(\alpha, \beta) & (\xi_i, \omega_i) &\overset{i.i.d.}{\sim} H \times G && (12) \\
z_s \mid z_{s-1} &\sim \bar{\pi}_{z_{s-1}} \\
d_s \mid \omega_{z_s} &\sim D(\omega_{z_s}) && && (13) \\
y_{t_s^1:t_s^2} &\overset{i.i.d.}{\sim} F(\xi_{z_s}) & t_s^1 &= \sum_{\bar{s}<s} d_{\bar{s}} && (14)
\end{aligned}$$

where $t_s^2 = t_s^1 + d_s - 1$. Johnson & Willsky [11] define $\bar{\pi} \triangleq \frac{\pi_{ij}}{1-\pi_{ii}}(1-\delta_{ij})$ to eliminate self-transitions in what they term their super-state sequence (z_s).

Secondly, the IED-HMM arises from the infinite structured hidden semi-Markov model Bayesian nonparametric framework [10, §5]. This framework, in which the IED-HMM is specified, directly parametrises state duration distributions, allowing for more heterogeneity and specificity, in state dwell durations when compared to the IED-HMM [10].

Summarising, our IDHMM differs from these two precedents primarily (ignoring construction and inference details) by modelling durations non-parametrically. By levering the statistical strength of hierarchical processes also for durations, we can model complex duration phenomena like multi-modal duration distributions which may be more difficult to treat in a parametric setting.

F ANGLICAN CODE

We list here an example of our Anglican code. Specifically, we include our implementation of the IDHMM. Our full code base can be found on https://goo.gl/14s8Sa.

```
(with-primitive-procedures
  [shape
   mmul
   model-params
   DP-ind]
  (defquery idhmm
    "Infinite duration hidden Markov model"
    [observations]
    (let
      [ ;; ==== Initialize Processes for State and Durations Transitions ====

       alpha-trans-0 (sample (model-params :alpha-trans-0-prior))
       _ (store "state-transition-base-process" (DP alpha-trans-0
                                                    (model-params :trans-dist)))

       alpha-dur-0 (sample (model-params :alpha-dur-0-prior))
       _ (store "duration-base-process" (DP alpha-dur-0
                                            (model-params :dur-dist)))

       ;; ==== Specify State and Duration Transitions ====

       alpha-dur (sample (model-params :alpha-dur-prior))
       transition-duration
       (fn [S D]
         (if (< 1 D)
           (- D 1)
           (let [ duration-base-process (retrieve "duration-base-process")
                 duration-group-process (or (retrieve ["duration-group-process" S])
                                            (DP-ind alpha-dur (produce duration-base-process)))
                 newpair (sample (produce duration-group-process))
                 newduration (first newpair)
                 newind (second newpair)
                 _ (if newind
                     (store "duration-base-process" (absorb duration-base-process
                         newduration))
                     0)
                 _ (store ["duration-group-process" S] (absorb duration-group-process
                     newduration))]
             newduration)))

       alpha-trans (sample (model-params :alpha-dur-prior))
       transition-state
       (fn [S D]
         (if (< 1 D)
           S
           (let [ transition-base-process (retrieve "state-transition-base-process")
                 transition-group-process (or (retrieve ["state-transition-process" S])
                                              (DP-ind alpha-trans (produce transition-base-process)))
                 ;; Now, we need to exclude self-transtions. We do this by sampling a
                     new state (and indicator) repeatedly
                 ;; and rejecting until we obtain one that is not equal to the state we
                     started from.
                 ;; Note that we only absorb into the DPs after we have sampled a state
                     that we will keep.
                 ;; Again, we always absorb into the group level transition process and
                     depending on the indicator which
                 ;; shows if we sampled from the base distribution of the group level
                     process, we also absorb in the base level
                 ;; transition process.
```

```
            newpair (loop [a [S false]]
            (if (= (first a) S)
              (recur (sample (produce transition-group-process)))
              a))
            newstate (first newpair)
            newind (second newpair)
            _ (if newind
                 (store "state-transition-base-process" (absorb
                     transition-base-process newstate))
                 0)
            _ (store ["state-group-transition-process" S] (absorb
                transition-group-process newstate))]
       newstate)))

    markov-step
    (fn [prev-item]
      (let [s (transition-state (first prev-item) (second prev-item))]
        [s (transition-duration s (second prev-item))]))

    ;; ==== Initialize Observation Process ====

    emperical-observation-mean (mean observations 0)
    rescaled-emperical-observation-cov (mmul
                                (model-params :cov-mat-scale)
                                (covariance observations 0))
    dof (+ 2 (second (shape observations)))
    inverse-cov-scale (sample (model-params :inverse-cov-scale))

    obs-base-proc (mvn-niw
                emperical-observation-mean
                inverse-cov-scale
                dof
                rescaled-emperical-observation-cov)]

  ;; ==== Condition on observations ====

  (reduce
    (fn [states-and-durations obs]
      (let [s_d (markov-step (peek states-and-durations))
            s (first s_d)
            proc (or (retrieve ["observation-process" s])
                     obs-base-proc)]

        (observe (produce proc) obs)
        (store ["observation-process" s] (absorb proc obs))
        (conj states-and-durations s_d)))
    (let [state-transition-base-process (retrieve "state-transition-base-process")
          duration-base-process (retrieve "duration-base-process")
          s (sample (produce state-transition-base-process))
          d (sample (produce duration-base-process))
          _ (store "state-transition-base-process" (absorb
              state-transition-base-process s) )
          _ (store "duration-base-process" (absorb duration-base-process d))]
      [[s d]])
    observations))))
```