Inference in differentiable generative models

Matthew Graham [1] **Amos Storkey** [1]

Abstract

Many generative models can be expressed as a differentiable function of random inputs drawn from a known probability distribution. This framework includes both learnt parametric generative models and a large class of procedurally defined simulator models. We present a method for performing efficient Markov chain Monte Carlo (MCMC) inference in such models when conditioning on observations of the model output. For some models this offers an asymptotically exact inference method where Approximate Bayesian Computation might otherwise be employed. We use the intuition that inference corresponds to integrating a density across the manifold corresponding to the set of inputs consistent with the observed outputs. This motivates the use of a constrained variant of Hamiltonian Monte Carlo which leverages the smooth geometry of the manifold to move between inputs exactly consistent with observations.

1. Introduction

There has been a long interest in probabilistic models which are defined *implicitly* [4, 7, 12] - that is where we can generate random values for the latent $\mathbf{z} \in \mathcal{Z}$ and observed $\mathbf{x} \in \mathcal{X}$ variables in the model, but we cannot tractably evaluate the probability distribution $\mathsf{P}_{\mathbf{x},\mathbf{z}}$ of those variables and more specifically its density $\mathsf{p}_{\mathbf{x},\mathbf{z}}$ with respect to an appropriate base measure. Although implicit models are challenging from an inferential perspective, they are ubiquitous in science and engineering in the form of probabilistic models defined by the simulation of a physical system. Typically simulator models are specified procedurally in code with any stochasticity introduced by drawing values from a random number generator. Recently implicit generative models have also been the subject of much interest in the machine learning community due to the advent of effective adversarial training methods [10] and the gains in modelling flexibility offered by dropping the requirement of being able to compute an explicit density on model outputs [22, 29].

[1]University of Edinburgh, Edinburgh, United Kingdom. Correspondence to: Matthew Graham <m.m.graham@ed.ac.uk>.

Any probabilistic model that we can programmatically generate values from can be expressed in the form of a deterministic function which takes as input a (possibly variable length) vector of random inputs sampled from a known distribution. This observation just corresponds to stating that we can track all of the calls to a random number generator in a program, and that given the values sampled from the random number generator all of the operations then performed by the program are deterministic.

To formalise this intuition we introduce some notation. We define an *input space* $\mathcal{U}$ and an associated vector of input random variables $\mathbf{u} \in \mathcal{U}$ which takes on values in this space, with $\mathbf{u}$ having a known distribution $\mathsf{P}_{\mathbf{u}} = P$. We then define *generator functions* $\boldsymbol{g}_{\mathbf{x}} : \mathcal{U} \to \mathcal{X}$ and $\boldsymbol{g}_{\mathbf{z}} : \mathcal{U} \to \mathcal{Z}$ such that we have $\mathbf{x} = \boldsymbol{g}_{\mathbf{x}}(\mathbf{u})$ and $\mathbf{z} = \boldsymbol{g}_{\mathbf{z}}(\mathbf{u})$. A common special case is when the input space is partitioned $\mathcal{U} = \mathcal{U}_1 \times \mathcal{U}_2$ and the unobserved variables $\mathbf{z}$ are generated from a subset of the random inputs $\mathbf{u}_1 \in \mathcal{U}_1$ (e.g. corresponding to sampling from a prior distribution over the parameters of a simulator model), with the observed variables $\mathbf{x}$ then generated from a function $\boldsymbol{g}_{\mathbf{x}|\mathbf{z}} : \mathcal{Z} \times \mathcal{U}_2 \to \mathcal{X}$, i.e. $\mathbf{x} = \boldsymbol{g}_{\mathbf{x}|\mathbf{z}}(\boldsymbol{g}_{\mathbf{z}}(\mathbf{u}_1), \mathbf{u}_2)$. In Appendix A in the supplementary material we show factor graphs visualising these model structures.

We define *differentiable generative models* as the restricted class of models whereby

1. $\mathcal{U} \subseteq \mathbb{R}^M$ and $\mathcal{X} \subseteq \mathbb{R}^{N_{\mathbf{x}}}$
2. P has density ρ with respect to the Lebesgue measure,
3. $\frac{\partial \rho}{\partial \boldsymbol{u}}$ exists P-almost everywhere,
4. $\frac{\partial \boldsymbol{g}_{\mathbf{x}}}{\partial \boldsymbol{u}}$ exists and is full row-rank P-almost everywhere.

In this paper we outline a MCMC method for performing inference in models meeting these conditions. This is an abridged version of a published conference paper [13].

2. Approximate Bayesian Computation

Approximate Bayesian Computation (ABC) [4, 20] is a popular approach for performing approximate inference generative models where we do not have access to an explicit joint density $\mathsf{p}_{\mathbf{x},\mathbf{z}}$ but can generate $(\mathbf{x}, \mathbf{z})$ samples. In ABC the simulated observed outputs $\mathbf{x}$ are decoupled from the observations $\bar{\mathbf{x}}$ by a noise model or *kernel* $\mathsf{p}_{\bar{\mathbf{x}}\,|\,\mathbf{x}}(\bar{\boldsymbol{x}}\,|\,\boldsymbol{x}) = k_\epsilon(\bar{\boldsymbol{x}};\, \boldsymbol{x})$ with tolerance parameter ϵ, e.g. $k_\epsilon(\bar{\boldsymbol{x}};\, \boldsymbol{x}) \propto \mathbb{I}[|\bar{\boldsymbol{x}} - \boldsymbol{x}| < \epsilon]/\epsilon^{N_{\mathbf{x}}}$ (uniform ball kernel) or $k_\epsilon(\bar{\boldsymbol{x}};\, \boldsymbol{x}) = \mathcal{N}\big(\bar{\boldsymbol{x}}\,|\,\boldsymbol{x},\, \epsilon^2 \boldsymbol{I}\big)$ (Gaussian kernel).

The *ABC posterior* is then defined as

$$\mathsf{p}_{\mathbf{z}\,|\,\bar{\mathbf{x}}}(\boldsymbol{z}\,|\,\bar{\boldsymbol{x}};\,\epsilon)) = \frac{\int_{\mathcal{X}} k_\epsilon(\bar{\boldsymbol{x}};\,\boldsymbol{x})\,\mathsf{p}_{\mathbf{x}\,|\,\mathbf{z}}(\boldsymbol{x}\,|\,\boldsymbol{z})\,\mathsf{p}_{\mathbf{z}}(\boldsymbol{z})\,\mathrm{d}\boldsymbol{x}}{\int_{\mathcal{X}} k_\epsilon(\bar{\boldsymbol{x}};\,\boldsymbol{x})\,\mathsf{p}_{\mathbf{x}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}} \quad (1)$$

and we can express expectations of functions of the latent variables $\mathbf{z}$ with respect to this approximate posterior

$$\mathbb{E}[f(\mathbf{z})\,|\,\bar{\mathbf{x}} = \bar{\boldsymbol{x}};\,\epsilon] = \int_{\mathcal{Z}} f(\boldsymbol{z})\,\mathsf{p}_{\mathbf{z}\,|\,\bar{\mathbf{x}}}(\boldsymbol{z}\,|\,\bar{\boldsymbol{x}};\,\epsilon)\,\mathrm{d}\boldsymbol{z} \quad (2)$$
$$= \frac{\int_{\mathcal{Z}}\int_{\mathcal{X}} f(\boldsymbol{z})\,k_\epsilon(\bar{\boldsymbol{x}};\,\boldsymbol{x})\,\mathsf{p}_{\mathbf{x}\,|\,\mathbf{z}}(\boldsymbol{x}\,|\,\boldsymbol{z})\,\mathsf{p}_{\mathbf{z}}(\boldsymbol{z})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{z}}{\int_{\mathcal{Z}}\int_{\mathcal{X}} k_\epsilon(\bar{\boldsymbol{x}};\,\boldsymbol{x})\,\mathsf{p}_{\mathbf{x}\,|\,\mathbf{z}}(\boldsymbol{x}\,|\,\boldsymbol{z})\,\mathsf{p}_{\mathbf{z}}(\boldsymbol{z})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{z}}.$$

Various Monte Carlo appproximate inference schemes can be used to estimate this expectation. The simplest is to generate a set of independent samples $\left\{\boldsymbol{x}^{(s)},\,\boldsymbol{z}^{(s)}\right\}_{s=1}^{S}$ from $\mathsf{P}_{\mathbf{x},\,\mathbf{z}}$[1] and use the ratio estimator

$$\mathbb{E}[f(\mathbf{z})\,|\,\bar{\mathbf{x}} = \bar{\boldsymbol{x}};\,\epsilon] \approx \frac{\sum_{s=1}^{S} f\left(\boldsymbol{z}^{(s)}\right) k_\epsilon\left(\bar{\boldsymbol{x}};\,\boldsymbol{x}^{(s)}\right)}{\sum_{s=1}^{S} k_\epsilon\left(\bar{\boldsymbol{x}};\,\boldsymbol{x}^{(s)}\right)}. \quad (3)$$

In the case of a uniform ball kernel this corresponds to the standard ABC reject algorithm, with expectations being estimated as averages over the latent variable samples for which the corresponding simulated outputs are within a (Euclidean) distance of ϵ from the observations.

Alternatively a MCMC scheme can be used to estimate the ABC posterior expectation in (2) [21]. A Markov chain is constructed with a stationary distribution with a density

$$\mathsf{p}_{\mathbf{x},\,\mathbf{z}\,|\,\bar{\mathbf{x}}}(\boldsymbol{x},\,\boldsymbol{z}\,|\,\bar{\boldsymbol{x}}) \propto k_\epsilon(\bar{\boldsymbol{x}};\,\boldsymbol{x})\,\mathsf{p}_{\mathbf{x}\,|\,\mathbf{z}}(\boldsymbol{x}\,|\,\boldsymbol{z})\,\mathsf{p}_{\mathbf{z}}(\boldsymbol{z}) \quad (4)$$

by proposing a new state $(\boldsymbol{x}',\,\boldsymbol{z}')$ given the current state $(\boldsymbol{x},\,\boldsymbol{z})$ by sampling $\boldsymbol{z}' \sim q(\cdot\,|\,\boldsymbol{z})$ from some perturbative proposal distribution q on $\mathcal{Z}$ and then generating a new $\boldsymbol{x}' \sim \mathsf{p}_{\mathbf{x}\,|\,\mathbf{z}}(\cdot\,|\,\boldsymbol{z})$. This is then accepted with probability

$$a(\boldsymbol{x}',\boldsymbol{z}'\,|\,\boldsymbol{x},\boldsymbol{z}) = \min\left[1,\,\frac{k_\epsilon(\bar{\boldsymbol{x}};\,\boldsymbol{x}')q(\boldsymbol{z}\,|\,\boldsymbol{z}')\mathsf{p}_{\mathbf{z}}(\boldsymbol{z}')}{k_\epsilon(\bar{\boldsymbol{x}};\,\boldsymbol{x})q(\boldsymbol{z}'\,|\,\boldsymbol{z})\mathsf{p}_{\mathbf{z}}(\boldsymbol{z})}\right], \quad (5)$$

the overall transition operator leaving (4) stationary. The samples of the chain state can then be used to compute consistent estimators of (2).

For both the ABC reject and ABC MCMC schemes just described, simulated observations $\boldsymbol{x}$ are independently generated from the conditional $\mathsf{p}_{\mathbf{x}\,|\,\mathbf{z}}(\boldsymbol{x}\,|\,\boldsymbol{z})$ given the current latent variables $\boldsymbol{z}$. The observed values $\bar{\boldsymbol{x}}$ are a zero-measure set in $\mathcal{X}$ under non-degenerate $\mathsf{p}_{\mathbf{x}\,|\,\mathbf{z}}(\boldsymbol{x}\,|\,\boldsymbol{z})$ and so as $\epsilon \to 0$ the probability of accepting a sample / proposed move becomes zero. Applying ABC with a non-zero ϵ therefore can be seen as a practically motivated relaxation of the constraint that true and simulated data exactly match, and hence the 'approximate' in *Approximate Bayesian Computation*.

[1] In ABC this usually involves generating $\mathbf{z}$ then simulating $\mathbf{x}$ given $\mathbf{z}$ however more generally we can just sample from the joint.

3. Inference in the input space

Using our definitions $\mathbf{x} = \boldsymbol{g}_{\mathbf{x}}(\mathbf{u})$ and $\mathbf{z} = \boldsymbol{g}_{\mathbf{z}}(\mathbf{u})$ we can reparametrise (2) as an integral over the input space to the generative model

$$\mathbb{E}[f(\mathbf{z})\,|\,\bar{\mathbf{x}} = \bar{\boldsymbol{x}};\,\epsilon] \propto \int_{\mathcal{U}} f \circ \boldsymbol{g}_{\mathbf{z}}(\boldsymbol{u})\,k_\epsilon(\bar{\boldsymbol{x}};\,\boldsymbol{g}_{\mathbf{x}}(\boldsymbol{u}))\,\rho(\boldsymbol{u})\,\mathrm{d}\boldsymbol{u}.$$

This indicates we can estimate ABC expectations by applying standard MCMC methods in the input space to construct a chain with stationary distribution with density

$$\pi_\epsilon(\boldsymbol{u}) \propto k_\epsilon(\bar{\boldsymbol{x}};\,\boldsymbol{g}_{\mathbf{x}}(\boldsymbol{u}))\,\rho(\boldsymbol{u}). \quad (6)$$

We can also however consider the limit of $\epsilon \to 0$. We have that the kernel term $k_\epsilon(\bar{\boldsymbol{x}};\,\boldsymbol{g}_{\mathbf{x}}(\boldsymbol{u})) \to \delta(\bar{\boldsymbol{x}} - \boldsymbol{g}_{\mathbf{x}}(\boldsymbol{u}))$ and so

$$\mathbb{E}[f(\mathbf{z})\,|\,\mathbf{x} = \bar{\boldsymbol{x}}] = \lim_{\epsilon\to 0}\{\mathbb{E}[f(\mathbf{z})\,|\,\bar{\mathbf{x}} = \bar{\boldsymbol{x}};\,\epsilon]\} \quad (7)$$
$$\propto \int_{\mathcal{U}} f \circ \boldsymbol{g}_{\mathbf{z}}(\boldsymbol{u})\,\delta(\bar{\boldsymbol{x}} - \boldsymbol{g}_{\mathbf{x}}(\boldsymbol{u}))\,\rho(\boldsymbol{u})\,\mathrm{d}\boldsymbol{u}.$$

The Dirac delta term restricts the integral across the input space $\mathcal{U}$ to an embedded, $M - N_{\mathbf{x}}$ dimensional, implicitly-defined manifold $\boldsymbol{g}_{\mathbf{x}}^{-1}[\bar{\boldsymbol{x}}] \equiv \{\boldsymbol{u} \in \mathcal{U} : \boldsymbol{g}_{\mathbf{x}}(\boldsymbol{u}) = \bar{\boldsymbol{x}}\}$. By applying the the *Co-Area Formula* [9, §3.2.12] the integral with respect to the Lebesgue measure across $\mathcal{U}$ in (7) can be rewritten as integral across the embedded manifold $\boldsymbol{g}_{\mathbf{x}}^{-1}[\bar{\boldsymbol{x}}]$ with respect the Hausdorff measure for the manifold

$$\mathbb{E}[f(\mathbf{z})\,|\,\mathbf{x} = \bar{\boldsymbol{x}}] \propto \quad (8)$$
$$\int_{\boldsymbol{g}_{\mathbf{x}}^{-1}[\bar{\boldsymbol{x}}]} f \circ \boldsymbol{g}_{\mathbf{z}}(\boldsymbol{u}) \left|\frac{\partial \boldsymbol{g}_{\mathbf{x}}}{\partial \boldsymbol{u}}\frac{\partial \boldsymbol{g}_{\mathbf{x}}}{\partial \boldsymbol{u}}^{\mathrm{T}}\right|^{-\frac{1}{2}} \rho(\boldsymbol{u})\,\mathrm{d}\mathcal{H}^{M-N_{\mathbf{x}}}(\boldsymbol{u}).$$

Therefore if we construct a Markov chain $\{\mathbf{u}^{(s)}\}_{s=1}^{S}$ restricted to $\boldsymbol{g}_{\mathbf{x}}^{-1}[\bar{\boldsymbol{x}}]$ and with an invariant distribution with density with respect to the Hausdorff measure of $\boldsymbol{g}_{\mathbf{x}}^{-1}[\bar{\boldsymbol{x}}]$

$$\pi(\boldsymbol{u}) \propto \left|\frac{\partial \boldsymbol{g}_{\mathbf{x}}}{\partial \boldsymbol{u}}\frac{\partial \boldsymbol{g}_{\mathbf{x}}}{\partial \boldsymbol{u}}^{\mathrm{T}}\right|^{-\frac{1}{2}} \rho(\boldsymbol{u}) \quad (9)$$

then if the chain is also aperiodic and irreducible we can form MCMC estimators which converge in the limit $S \to \infty$

$$\mathbb{E}\left[\frac{1}{S}\sum_{s=1}^{S}\left(f \circ \boldsymbol{g}_{\mathbf{z}}\left(\mathbf{u}^{(s)}\right)\right)\right] \to \mathbb{E}[f(\mathbf{z})\,|\,\mathbf{x} = \bar{\boldsymbol{x}}] \quad (10)$$

Intuitively the determinant term in (9) adjusts for the change in the infinitesimal 'thickness' (extent in directions orthogonal to the tangent space) of the manifold when mapping through the generator function $\boldsymbol{g}_{\mathbf{x}}$. The result (8) is given in a slightly different form in [6].

A general framework for performing asymptotically exact inference in differentiable generative models is therefore to define MCMC updates which explore the target density (9) on the pre-image manifold $\boldsymbol{g}_{\mathbf{x}}^{-1}[\bar{\boldsymbol{x}}]$. We propose here to use a method which simulates the dynamics of a constrained mechanical system.

4. Constrained Hamiltonian Monte Carlo

In *Hamiltonian Monte Carlo* (HMC) [8, 26] the vector variable of interest $\mathbf{u}$ is augmented with a momentum variable $\mathbf{p} \in \mathbb{R}^M$. The momenta are defined to be independent of $\mathbf{u}$ with a Gaussian distribuion $\mathcal{N}(\mathbf{0}, \mathbf{I})$[2]. The joint distribution then has a density proportional to $\exp[-H(\boldsymbol{u}, \boldsymbol{p})]$ where $H(\boldsymbol{u}, \boldsymbol{p}) = -\log \pi(\boldsymbol{u}) + \frac{1}{2}\boldsymbol{p}^\mathsf{T}\boldsymbol{p}$ is the Hamiltonian. The canonical Hamiltonian dynamic is described by

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = \frac{\partial H}{\partial \boldsymbol{p}} = \boldsymbol{p}, \quad \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = -\frac{\partial H}{\partial \boldsymbol{u}} = \frac{\partial \log \pi}{\partial \boldsymbol{u}}. \tag{11}$$

This dynamic is time-reversible, volume-preserving and exactly conserves the Hamiltonian. Symplectic integrators allow approximate integration of the Hamiltonian flow while maintaining the time-reversibility and volume-preservation properties [17]. These properties make simulated Hamiltonian dynamics an ideal proposal mechanism for a Metropolis MCMC method. The accept probability for a proposal $(\boldsymbol{u}_\mathrm{p}, \boldsymbol{p}_\mathrm{p})$ generated by simulating the dynamic forward from $(\boldsymbol{u}, \boldsymbol{p})$ is $\exp\big(H(\boldsymbol{u}, \boldsymbol{p}) - H(\boldsymbol{u}_\mathrm{p}, \boldsymbol{p}_\mathrm{p})\big)$. Typically the change in the Hamiltonian will be small and so the probability of acceptance high.

In our case the system is subject to a constraint of the form $\boldsymbol{g}_\mathbf{x}(\boldsymbol{u}) - \bar{\boldsymbol{x}} = \mathbf{0}$. By introducing Lagrangian multipliers λ_i for each of the constraints, the Hamiltonian for a constrained system can be written as $H(\boldsymbol{u}, \boldsymbol{p}) = -\log \pi(\boldsymbol{u}) + \frac{1}{2}\boldsymbol{p}^\mathsf{T}\boldsymbol{p} + \boldsymbol{\lambda}^\mathsf{T}(\boldsymbol{g}_\mathbf{x}(\boldsymbol{u}) - \bar{\boldsymbol{x}})$, with a corresponding constrained Hamiltonian dynamic

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = \boldsymbol{p}, \quad \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = \frac{\partial \log \pi}{\partial \boldsymbol{u}} - \frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}}^\mathsf{T} \boldsymbol{\lambda}, \tag{12}$$

$$\text{subject to } \boldsymbol{g}_\mathbf{x}(\boldsymbol{u}) = \bar{\boldsymbol{x}}, \quad \frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}}\boldsymbol{p} = \mathbf{0}. \tag{13}$$

A popular numerical integrator for simulating constrained Hamiltonian dynamics is RATTLE [2]. This a generalisation of the Störmer-Verlet (leapfrog) integrator typically used in standard HMC with additional projection steps in which the Lagrange multipliers $\boldsymbol{\lambda}$ are solved for to satisfy the conditions (13). RATTLE is symplectic on the constraint manifold and is time-reversible and volume-preserving [18].

The use of constrained dynamics in HMC has been proposed several times. In the molecular dynamics literature, both [14] and [19] suggest using a simulated constrained dynamic within a HMC framework. Most relevantly here [5] proposes using a constrained HMC variant to perform inference in distributions defined on implicitly defined manifolds. The authors of [5] give sufficient conditions on H and $\boldsymbol{g}_\mathbf{x}$ for the transition operator to be irreducible and aperiodic: that H is C^2 continuous, and $\boldsymbol{g}_\mathbf{x}^{-1}[\bar{\boldsymbol{x}}]$ is a connected smooth and differentiable manifold and $\frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}}$ has full row-rank everywhere.

[2]For notation simplicity we assume an identity mass matrix.

5. Method

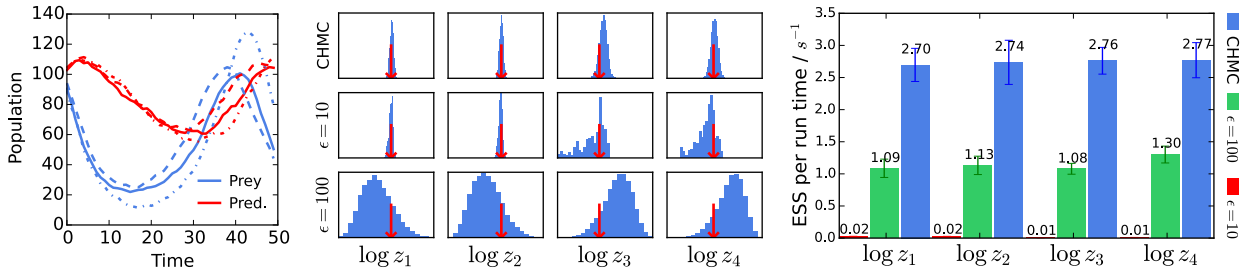
Our constrained HMC implementation is shown in Appendix C in Algorithm 1. We use a generalisation of the RATTLE scheme to simulate the dynamic. The inner updates of the state to solve for the geodesic motion on the constraint manifold are split into multiple smaller steps. This is a special case of the scheme described in [16] and allows more flexibility in choosing an appropriately small step-size to ensure convergence of the iterative solution of the equations projecting on to the constraint manifold while still allowing a more efficient larger step size for updates to the momentum.

Each inner geodesic time-step involves making an unconstrained update $\tilde{\boldsymbol{u}} \leftarrow \boldsymbol{u} - \delta t \boldsymbol{p}$ and then projecting $\tilde{\boldsymbol{u}}$ back on to $\boldsymbol{g}_\mathbf{x}^{-1}[\bar{\boldsymbol{x}}]$ by solving for $\boldsymbol{\lambda}$ which satisfy $\boldsymbol{g}_\mathbf{x}(\tilde{\boldsymbol{u}} - \frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}}^\mathsf{T}\boldsymbol{\lambda}) = \bar{\boldsymbol{x}}$. Here we use a-Newton method for solving the system of equations in the projection step. The true Newton update would involve recalculating the Jacobian and solving a dense linear system within the optimisation loop. Instead we use a symmetric quasi-Newton update as proposed in [3], the Jacobian matrix product $\frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}}\frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}}^\mathsf{T}$ evaluated at the previous state used to condition the moves. This matrix is positive-definite and a Cholesky decomposition can be calculated outside the optimisation loop allowing cheaper quadratic cost solves within the loop. For larger systems, the Cholesky decomposition of the Jacobian matrix prodcut will become a dominant cost, generally scaling cubically with $N_\mathbf{x}$. Conditional independency structure of many directed generative models however allows a significantly reduced quadratically scaling computational cost as explained in the supplementary material in Appendix D.

In the Newton iteration projection step, convergence is signalled when the elementwise maximum absolute difference $\|\boldsymbol{g}_\mathbf{x}(\boldsymbol{u}) - \bar{\boldsymbol{x}}\|_\infty$ is below a tolerance ϵ. This acts analogously to the ϵ parameter in ABC methods, however here we typically set this parameter to close to machine precision ($\epsilon = 10^{-8}$ in our experiments) and so the approximation introduced is comparable to that otherwise incurred for using non-exact arithmetic.

6. Example: Lotka–Volterra model

As an example application of our method, we consider inferring the posterior distribution of the parameters of a stochastic predator-prey model given (simulated) observed populations. In Appendices E and F we also show results for 3D human pose and camera parameter inference given 2D joint position information and in-painting of missing regions of digit images using a generative model trained on MNIST. In all experiments Theano [28] was used to specify the generator function and calculate the required derivatives. Python code for the experiments is available



(a) (b) (c)

Figure 1. **Lotka–Volterra** (a) Observed predator-prey populations (solid) and ABC sample trajectories with $\epsilon = 10$ (dashed) and $\epsilon = 100$ (dot-dashed). (b) Marginal empirical histograms for the (logarithm of the) four parameters (columns) from constrained HMC samples (top) and ABC samples with $\epsilon = 10$ (middle) and $\epsilon = 100$ (bottom). Horizontal axes shared across columns. Red arrows indicate true parameter values. (c) Mean ESS normalised by compute time for each of four parameters for ABC with $\epsilon = 10$ (red), $\epsilon = 100$ (green) and our method (blue). Error bars show ± 3 standard errors of mean.

at https://git.io/dgm. We considered a stochastic continuous state variant of the Lotka–Volterra model. In particular we consider parameter inference given a simulated solution of the following stochastic differential equations

$$\mathrm{d}x_1 = (z_1 x_1 - z_2 x_1 x_2)\mathrm{d}t + \mathrm{d}n_1, \tag{14}$$

$$\mathrm{d}x_2 = (-z_3 x_2 + z_4 x_1 x_2)\mathrm{d}t + \mathrm{d}n_2, \tag{15}$$

where x_1 represents the prey population, x_2 the predator population, $\{z_i\}_{i=1}^4$ the system parameters and n_1 and n_2 zero-mean, unit variance white noise processes.

The observed data was generated with an Euler-Maruyama discretisation, time-step 1, initial condition $x_1^{(0)} = x_2^{(0)} = 100$ and $z_1 = 0.4$, $z_2 = 0.005$, $z_3 = 0.05$, $z_4 = 0.001$ (chosen to give stable dynamics). The simulation was run for 50 time-steps with the observed outputs defined as the concatenated vector $\boldsymbol{x} = \left[x_1^{(1)}\ x_2^{(1)}\ \ldots\ x_1^{(50)}\ x_2^{(50)}\right]$. Log-normal priors $z_i \sim \log\mathcal{N}(-2, 1)$ were place on the system parameters.

We compared our method to various ABC approaches (§2) using a uniform ball kernel with radius ϵ. ABC rejection failed catastrophically, with no acceptances in 10^6 samples even with a large $\epsilon = 1000$. ABC MCMC with a Gaussian proposal distribution q also performed very poorly with the dynamic having zero acceptances over multiple runs of 10^5 updates for $\epsilon = 100$ and getting stuck at points in parameter space over many updates for larger $\epsilon = 1000$, even with small proposal steps. Based on a method proposed in the pseudo-marginal literature [25], we tried using alternating elliptical slice sampling [24] updates of the inputs $\mathbf{u}_1$ used to generate the parameters $\mathbf{z} = \boldsymbol{g}_{\mathbf{z}}(\mathbf{u}_1)$ and remaining random inputs $\mathbf{u}_2$ used to generate the observed variables $\mathbf{x} = \boldsymbol{g}_{\mathbf{x}|\mathbf{z}}(\mathbf{z}, \mathbf{u}_2)$ given $\mathbf{z}$. The slice sampling updates adapt the size of steps made to ensure a move can always be made. Using this method we were able to obtain reasonable convergence over long runs for both $\epsilon = 100$ and $\epsilon = 10$.

The results are summarised in Figure 1. Figure 1a shows the simulated data used as observations and ABC sample trajectories for $\epsilon = 10$ and $\epsilon = 100$. Though both samples follow the general trends of the observed data there are large discrepancies particularly for $\epsilon = 100$. Our method in contrast samples parameters generating trajectories matching the observations to within $\epsilon = 10^{-8}$ at all points. Figure 1b shows the marginal histograms for the parameters. As would be expected the inferred posterior on the parameters are significantly more tightly distributed about the true values used to generate the observations for our approach and the $\epsilon = 10$ case compared to the results for $\epsilon = 100$. Figure 1c shows the relative sampling efficiency of our approach against the ABC methods, as measured by the *effective sample sizes* (ESS) (computed with R-CODA [27]) normalised by run time averaged across 10 sampling runs for each method. Despite the significantly higher run time per-sample in our method, the reduced autocorrelation due to the much larger moves made by the Hamiltonian dynamic give much higher ESS even over the very approximate $\epsilon = 100$ case.

7. Discussion

We have presented a framework for performing inference in differentiable generative models. Though the constrained HMC updates are computationally costly, the gradient-based exploration of the state space can lead to significantly improved sampling efficiency over simpler methods. Further our approach in some cases allows asymptotically exact inference in differentiable generative models where ABC methods might otherwise be used. In our experiments we were able to condition on high-dimensional observations without the need for dimensionality reduction with summary statistics however if informative summaries are available and correspond to differentiable functions of the observations they can be exploited in our method by adding them to the generator definition. As well as being of practical importance itself, our approach should be useful in providing 'ground truth' inferences in test models to assess the affect of the approximations used in ABC methods.

References

[1] I. Akhter and M. J. Black. Pose-conditioned joint angle limits for 3D human pose reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[2] H. C. Andersen. RATTLE: A velocity version of the SHAKE algorithm for molecular dynamics calculations. *Journal of Computational Physics*, 1983.

[3] E. Barth, K. Kuczera, B. Leimkuhler, and R. D. Skeel. Algorithms for constrained molecular dynamics. *Journal of computational chemistry*, 1995.

[4] M. A. Beaumont, W. Zhang, and D. J. Balding. Approximate Bayesian computation in population genetics. *Genetics*, 2002.

[5] M. A. Brubaker, M. Salzmann, and R. Urtasun. A family of MCMC methods on implicitly defined manifolds. In *International Conference on Artificial Intelligence and Statistics*, 2012.

[6] P. Diaconis, S. Holmes, and M. Shahshahani. Sampling from a manifold. In *Advances in Modern Statistical Theory and Applications*, pages 102–125. Institute of Mathematical Statistics, 2013.

[7] P. J. Diggle and R. J. Gratton. Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 193–227, 1984.

[8] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Physics Letters B*, 1987.

[9] H. Federer. *Geometric measure theory*. 1969.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

[11] C. C. Gordon, T. Churchill, C. E. Clauser, B. Bradtmiller, J. T. McConville, I. Tebbets, and R. A. Walker. Anthropometric survey of US army personell: Final report. Technical report, United States Army, 1988.

[12] C. Gourieroux, A. Monfort, and E. Renault. Indirect inference. *Journal of applied econometrics*, 8(S1):S85–S118, 1993.

[13] M. Graham and A. Storkey. Asymptotically exact inference in differentiable generative models. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, pages 499–508, 2017.

[14] C. Hartmann and C. Schutte. A constrained hybrid Monte-Carlo algorithm and the problem of calculating the free energy in several variables. *ZAMM-Zeitschrift fur Angewandte Mathematik und Mechanik*, 2005.

[15] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations*, 2013.

[16] B. Leimkuhler and C. Matthews. Efficient molecular dynamics using geodesic integration and solvent–solute splitting. In *Proceedings of the Royal Society A*, 2016.

[17] B. Leimkuhler and S. Reich. *Simulating Hamiltonian dynamics*. Cambridge University Press, 2004.

[18] B. J. Leimkuhler and R. D. Skeel. Symplectic numerical integrators in constrained Hamiltonian systems. *Journal of Computational Physics*, 1994.

[19] T. Lelièvre, M. Rousset, and G. Stoltz. Langevin dynamics with constraints and computation of free energy differences. *Mathematics of computation*, 2012.

[20] J.-M. Marin, P. Pudlo, C. P. Robert, and R. J. Ryder. Approximate Bayesian computational methods. *Statistics and Computing*, 2012.

[21] P. Marjoram, J. Molitor, V. Plagnol, and S. Tavaré. Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 2003.

[22] S. Mohamed and B. Lakshminarayanan. Learning in implicit generative models. In *Proceedings of the International Conference on Learning Representations*, 2017.

[23] I. Murray. Differentiation of the Cholesky decomposition. *arXiv preprint arXiv:1602.07527*, 2016.

[24] I. Murray, R. P. Adams, and D. J. MacKay. Elliptical slice sampling. In *The Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, volume 9 of *JMLR: W&CP*, pages 541–548, 2010.

[25] I. Murray and M. M. Graham. Pseudo-marginal slice sampling. In *International Conference on Artificial Intelligence and Statistics*, 2016.

[26] R. M. Neal. *MCMC using Hamiltonian dynamics*, pages 113–162. 2011.

[27] M. Plummer, N. Best, K. Cowles, and K. Vines. CODA: Convergence diagnosis and output analysis for MCMC. *R News*, 6(1):7–11, 2006.

[28] Theano development team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, 2016.

[29] D. Tran, R. Ranganath, and D. M. Blei. Deep and hierarchical implicit models. *arXiv preprint arXiv:1702.08896*, 2017.

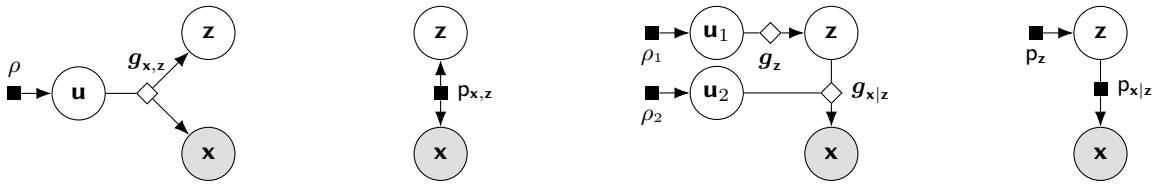


Figure 2. Factor graphs of undirected and directed generative models. Circular nodes represent random variables, filled square nodes probabilistic factors and unfilled diamonds deterministic factors. Shaded circular nodes are observed. Panel (a) shows the more general undirected model case in which observed variables $\mathbf{x}$ and latent variables $\mathbf{z}$ are jointly generated from inputs $\mathbf{u}$ by mapping through a function $\boldsymbol{g}_{\mathbf{x},\mathbf{z}}$, with (b) showing an equivalent factor graph after marginalising out the inputs $\mathbf{u}$. Panel (c) shows the directed model case in which we first generate the latent variables $\mathbf{z}$ from a subset of the inputs $\mathbf{u}_1$ then generate the observed variables $\mathbf{x}$ from $\mathbf{z}$ and the remaining inputs $\mathbf{u}_2$, with (d) showing a resulting natural directed factorisation of joint distribution when marginalising out $\mathbf{u}_1$ and $\mathbf{u}_2$.

A. Directed and undirected generative models

In Section 1 we described the general case of a generative model in which both the observed and unobserved variables being jointly generated by a function $\boldsymbol{g}_{\mathbf{x},\mathbf{z}}$. This structure is shown as a factor graph in Figure 2a and a corresponding factor graph for just $\mathbf{x}$ and $\mathbf{z}$ with $\mathbf{u}$ marginalised out shown in Figure 2b. As briefly mentioned in Section 1, a common special case is when the input space is partitioned $\mathcal{U} = \mathcal{U}_1 \times \mathcal{U}_2$ and the unobserved variables $\mathbf{z}$ are generated from a subset of the random inputs $\mathbf{u}_1 \in \mathcal{U}_1$, with the observed variables $\mathbf{x}$ then generated from a function $\boldsymbol{g}_{\mathbf{x}|\mathbf{z}}$ which takes as input both the remaining random variables $\mathbf{u}_2 \in \mathcal{U}_2$ and the generated unobserved variables $\mathbf{z}$, i.e. $\mathbf{x} = \boldsymbol{g}_{\mathbf{x}|\mathbf{z}}(\mathbf{z}, \mathbf{u}_2) = \boldsymbol{g}_{\mathbf{x}|\mathbf{z}}(\boldsymbol{g}_{\mathbf{z}}(\mathbf{u}_1), \mathbf{u}_2)$. This is illustrated as a factor graph in Figure 2c. Again a corresponding factor graph with $\mathbf{u}$ marginalised out is shown in Figure 2d, with in this case the structure of the generator making a *directed* factorisation in terms $\mathsf{p}_{\mathbf{z}}$ and $\mathsf{p}_{\mathbf{x}|\mathbf{z}}$ natural.

B. Evaluating the target density and its gradient

For the Hamiltonian dynamics we need to be able to evaluate the logarithm of the target density up to an additive constant and its gradient with respect to $\boldsymbol{u}$. Dropping the dependence of the Jacobian on $\boldsymbol{u}$ for brevity we have that

$$\log \pi(\boldsymbol{u}) = \log \rho(\boldsymbol{u}) - \frac{1}{2} \log\left|\mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}} \mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}}{}^{\mathsf{T}}\right| - \log Z \tag{16}$$

where Z is the normalising constant for the density which is independent of $\boldsymbol{u}$.

In general evaluating the determinant $|\mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}} \mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}}{}^{\mathsf{T}}|$ has computational cost which scales as $\mathcal{O}(MN_{\mathbf{x}}^2)$. However as part of the constrained dynamics updates the lower-triangular Cholesky decomposition $\boldsymbol{L}$ of $\mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}} \mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}}{}^{\mathsf{T}}$ is calculated. Using basic properties of the matrix determinant we have

$$\log \pi(\boldsymbol{u}) = \log \rho(\boldsymbol{u}) - \sum_{i=1}^{N_{\mathbf{x}}} \log(L_{ii}) - \log Z. \tag{17}$$

Given the Cholesky factor $\boldsymbol{L}$ we can therefore can evaluate the logarithm of the target density up to an additive constant at a marginal computational cost that scales linearly with $N_{\mathbf{x}}$. For the gradient we can use reverse-mode AD to calculate the gradient of (17) with respect to $\boldsymbol{u}$. This requires propagating partial derivatives through the Cholesky decomposition [23]; efficient implementations for this are present in many automatic differentiation frameworks.

Alternatively using the standard result for derivative of a log determinant and the invariance of the trace to cyclic permutations we have that the gradient of the log determinant term in (16) can be manipulated in to the form

$$\frac{1}{2} \frac{\partial}{\partial u_i} \log\left|\mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}} \mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}}{}^{\mathsf{T}}\right| = \operatorname{trace}\left(\left(\mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}} \mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}}{}^{\mathsf{T}}\right)^{-1} \frac{\partial \mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}}}{\partial u_i} \mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}}{}^{\mathsf{T}}\right) \tag{18}$$

$$= \operatorname{trace}\left(\mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}}{}^{\mathsf{T}} \left(\mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}} \mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}}{}^{\mathsf{T}}\right)^{-1} \frac{\partial \mathbf{J}_{\boldsymbol{g}_{\mathbf{x}}}}{\partial u_i}\right) \tag{19}$$

We denote the matrix vectorisation operator vec such that for a $M \times N$ matrix $\boldsymbol{A}$, we have $\operatorname{vec}(\boldsymbol{A}) = [A_{1,1}, \ldots, A_{M,1}, A_{1,2}, \ldots, A_{N,M}]^{\mathsf{T}}$. Then as the trace of a matrix product defines an inner product we have that

$\mathrm{trace}(\boldsymbol{A}\boldsymbol{B}) = \mathrm{vec}(\boldsymbol{A})^\mathsf{T}\mathrm{vec}(\boldsymbol{B})$. We can therefore write the gradient of the log determinant term as

$$\frac{1}{2}\frac{\partial}{\partial \boldsymbol{u}}\log\left|\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}{}^\mathsf{T}\right| = \mathrm{vec}\left(\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}{}^\mathsf{T}\left(\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}{}^\mathsf{T}\right)^{-1}\right)^\mathsf{T}\frac{\partial \mathrm{vec}\left(\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}\right)}{\partial \boldsymbol{u}} \tag{20}$$

The matrix inside the left vec operator can be computed once by reusing the Cholesky factorsation of $\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}{}^\mathsf{T}$ to solving the system of equations by forward and backward substitution. We then have an expression in the form of a vector-Jacobian product which is provided as an efficient primitive in many AD frameworks, e.g. as `Lop` in Theano, and like the gradient (which is actually a special case) can be evaluated at cost which is a constant over head of evaluating the forward function (i.e. the cost of evaluating $\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}$ here).

C. Constrained Hamiltonian Monte Carlo implementation

Algorithm 1 Constrained HMC in a differentiable generative model

Require:
- $\boldsymbol{g}_\mathbf{x}$: observed variable generator function;
- ϕ : potential energy function $\phi(\boldsymbol{u}) = -\log\rho(\boldsymbol{u}) + \frac{1}{2}\log|\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}(\boldsymbol{u})\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}(\boldsymbol{u})|$;
- $\boldsymbol{x}$: observed data values being conditioned on;
- $\boldsymbol{u}$: current chain state (model inputs) with $\|\boldsymbol{g}_\mathbf{x}(\boldsymbol{u}) - \boldsymbol{x}\|_\infty < \epsilon$;
- $(\varphi, \boldsymbol{J}, \boldsymbol{L})$: cached values of ϕ, $\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}$ and $\mathrm{chol}\left(\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}{}^\mathsf{T}\right)$ evaluated at $\boldsymbol{u}$;
- ϵ : convergence tolerance for Newton iteration;
- M : number of Newton iterations to try before rejecting for non-convergence;
- δt : integrator time step;
- N_s : number of time steps to simulate;
- N_g : number of geodesic steps per time step.

Ensure:
- $\boldsymbol{u}_\mathrm{n}$: new chain state with $\|\boldsymbol{g}_\mathbf{x}(\boldsymbol{u}_\mathrm{n}) - \boldsymbol{x}\|_\infty < \epsilon$;
- $(\varphi_\mathrm{n}, \boldsymbol{J}_\mathrm{n}, \boldsymbol{L}_\mathrm{n})$: values of ϕ, $\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}$ and $\mathrm{chol}\left(\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}{}^\mathsf{T}\right)$ evaluated at new $\boldsymbol{u}_\mathrm{n}$.

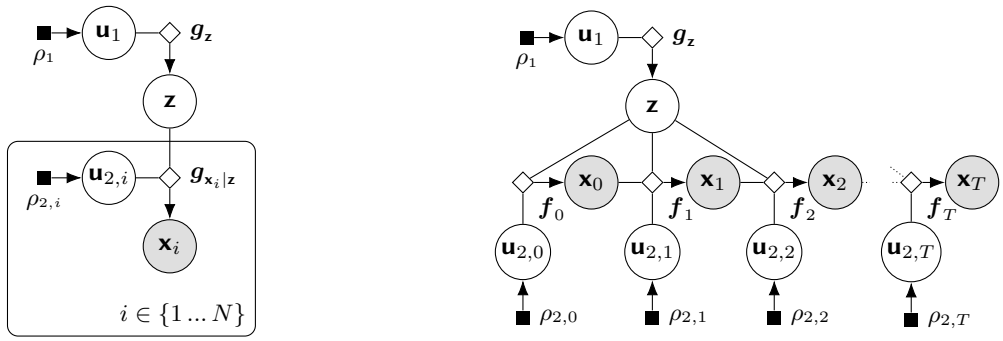
$\boldsymbol{n} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
$\boldsymbol{p} \leftarrow$ PROJECTMOM$(\boldsymbol{n}, \boldsymbol{J}, \boldsymbol{L})$
$\boldsymbol{u}_\mathrm{p}, \boldsymbol{p}_\mathrm{p}, \boldsymbol{J}_\mathrm{p}, \boldsymbol{L}_\mathrm{p} \leftarrow$ SIMDYN$(\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{J}, \boldsymbol{L})$
$\varphi_\mathrm{p} \leftarrow \phi(\boldsymbol{u})$
$r \sim \mathcal{U}(0, 1)$
$p_a \leftarrow \exp\left(\varphi + \frac{1}{2}\boldsymbol{p}^\mathsf{T}\boldsymbol{p} - \varphi_\mathrm{p} - \frac{1}{2}\boldsymbol{p}_\mathrm{p}^\mathsf{T}\boldsymbol{p}_\mathrm{p}\right)$
if $r < p_a$
  $\boldsymbol{u}_\mathrm{n}, \varphi_\mathrm{n}, \boldsymbol{J}_\mathrm{n}, \boldsymbol{L}_\mathrm{n} \leftarrow \boldsymbol{u}_\mathrm{p}, \varphi_\mathrm{p}, \boldsymbol{J}_\mathrm{p}, \boldsymbol{L}_\mathrm{p}$
else
  $\boldsymbol{u}_\mathrm{n}, \varphi_\mathrm{n}, \boldsymbol{J}_\mathrm{n}, \boldsymbol{L}_\mathrm{n} \leftarrow \boldsymbol{u}, \varphi, \boldsymbol{J}, \boldsymbol{L}$

function SIMDYN$(\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{J}, \boldsymbol{L})$
  $\tilde{\boldsymbol{p}} \leftarrow \boldsymbol{p} - \frac{\delta t}{2}\nabla\phi(\boldsymbol{u})$
  $\boldsymbol{p} \leftarrow$ PROJECTMOM$(\tilde{\boldsymbol{p}}, \boldsymbol{J}, \boldsymbol{L})$
  $\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{J}, \boldsymbol{L} \leftarrow$ SIMGEO$(\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{J}, \boldsymbol{L})$
  for $s \in \{2 \dots N_s\}$
    $\tilde{\boldsymbol{p}} \leftarrow \boldsymbol{p} - \delta t\nabla\phi(\boldsymbol{u})$
    $\boldsymbol{p} \leftarrow$ PROJECTMOM$(\tilde{\boldsymbol{p}}, \boldsymbol{J}, \boldsymbol{L})$
    $\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{J}, \boldsymbol{L} \leftarrow$ SIMGEO$(\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{J}, \boldsymbol{L})$
  $\tilde{\boldsymbol{p}} \leftarrow \boldsymbol{p} - \frac{\delta t}{2}\nabla\phi(\boldsymbol{u})$
  $\boldsymbol{p} \leftarrow$ PROJECTMOM$(\tilde{\boldsymbol{p}}, \boldsymbol{J}, \boldsymbol{L})$
  return $\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{J}, \boldsymbol{L}$

function PROJECTMOM$(\boldsymbol{p}, \boldsymbol{J}, \boldsymbol{L})$
  return $\boldsymbol{p} - \boldsymbol{J}^\mathsf{T}\boldsymbol{L}^{-\mathsf{T}}\boldsymbol{L}^{-1}\boldsymbol{J}\boldsymbol{p}$

function PROJECTPOS$(\boldsymbol{u}, \boldsymbol{J}, \boldsymbol{L})$
  $\boldsymbol{\delta} \leftarrow \boldsymbol{g}_\mathbf{x}(\boldsymbol{u}) - \boldsymbol{x}$
  $i \leftarrow 0$
  while $\|\boldsymbol{\delta}\|_\infty > \epsilon \wedge i < M$
    $\boldsymbol{u} \leftarrow \boldsymbol{u} - \boldsymbol{J}^\mathsf{T}\boldsymbol{L}^{-\mathsf{T}}\boldsymbol{L}^{-1}\boldsymbol{\delta}$
    $\boldsymbol{\delta} \leftarrow \boldsymbol{g}_\mathbf{x}(\boldsymbol{u}) - \boldsymbol{x}$
    $i \leftarrow i + 1$
  if $i = M$
    raise REJECTMOVE
  return $\boldsymbol{u}$

function SIMGEO$(\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{J}, \boldsymbol{L})$
  for $i \in \{1 \dots N_g\}$
    $\tilde{\boldsymbol{u}} \leftarrow \boldsymbol{u} + \frac{\delta t}{N_g}\boldsymbol{p}$
    $\boldsymbol{u}' \leftarrow$ PROJECTPOS$(\tilde{\boldsymbol{u}}, \boldsymbol{J}, \boldsymbol{L})$
    $\boldsymbol{J} \leftarrow \mathbf{J}_{\boldsymbol{g}_\mathbf{x}}(\boldsymbol{u}')$
    $\boldsymbol{L} \leftarrow \mathrm{chol}\left(\boldsymbol{J}\boldsymbol{J}^\mathsf{T}\right)$
    $\tilde{\boldsymbol{p}} \leftarrow \frac{N_g}{\delta t}(\boldsymbol{u}' - \boldsymbol{u})$
    $\boldsymbol{p} \leftarrow$ PROJECTMOM$(\tilde{\boldsymbol{p}}, \boldsymbol{J}, \boldsymbol{L})$
    $\boldsymbol{u}_r \leftarrow \boldsymbol{u}' - \frac{\delta t}{N_g}\boldsymbol{p}$
    $\boldsymbol{u}_r \leftarrow$ PROJECTPOS$(\boldsymbol{u}_r, \boldsymbol{J}, \boldsymbol{L})$
    if $\|\boldsymbol{u} - \boldsymbol{u}_r\|_\infty > \sqrt{\epsilon}$
      raise REJECTMOVE
    $\boldsymbol{u} \leftarrow \boldsymbol{u}'$
  return $\boldsymbol{u}, \boldsymbol{p}, \boldsymbol{J}, \boldsymbol{L}$



(a) Independent observations

(b) Markovian observations

Figure 3. Factor graphs of examples of structured directed generative models.

D. Exploiting model structure

For larger systems, the Cholesky decomposition of the constraint Jacobian matrix product $\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}{}^\mathsf{T}$ (line 42) will become a dominant cost, generally scaling cubically with $N_\mathbf{x}$. In many models however conditional independency structure will mean that not all observed variables $\mathbf{x}$ are dependent on all of the input variables $\mathbf{u}$ and so the Jacobian $\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}$ has sparsity structure which can be exploited to reduce this worst-case cost.

In particular two common cases are directed generative models in which the observed variables $\mathbf{x}$ can be split into groups $\{\mathbf{x}_g\}_{g=1}^G$ such that all of the $\mathbf{x}_i$ are either conditionally independent given the latent variables $\mathbf{z} = \boldsymbol{g}_\mathbf{z}(\mathbf{u}_1)$ (for example independent and identically distributed observations), or each $\mathbf{x}_i$ is conditionally independent of all $\{\mathbf{x}_j\}_{j<i-1}$ given $\mathbf{x}_{i-1}$ and $\mathbf{z}$ (most commonly Markov chains for example from a SDE model though observations with more general tree structured dependencies can also be ordered into this form). Figure 3 shows factor graphs for directed generative models with these two structures, with the conditional independencies corresponding to each $\mathbf{x}_i$ being generated as a function of only a subset $\mathbf{u}_{2,i}$ of the random input variables $\mathbf{u}_2$. Equivalently these structures correspond to generator functions $\boldsymbol{g}_\mathbf{x}$ which can be expressed in one of the two forms

$$\mathbf{x}_i = \boldsymbol{g}_{\mathbf{x}_i|\mathbf{z}}(\mathbf{z}, \mathbf{u}_{2,i}) \qquad \text{(independent)} \tag{21}$$

$$\mathbf{x}_i = \boldsymbol{f}_i\big(\mathbf{z}, \boldsymbol{f}_{i-1}(\mathbf{z}, \boldsymbol{f}_{i-2}(...), \mathbf{u}_{2,i-1}), \mathbf{u}_{2,i}\big) = \boldsymbol{g}_{\mathbf{x}_i|\mathbf{z}}(\mathbf{z}, \mathbf{u}_{2,\leq i}) \qquad \text{(Markov).} \tag{22}$$

For models with these structures the generator Jacobian $\mathbf{J}_{\boldsymbol{g}_\mathbf{x}} = [\frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}_1} \mid \frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}_2}]$ has a component $\frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}_2}$ which is either block-diagonal (independent) or block-triangular (Markov). Considering first the simplest case where each $(\mathbf{x}_i, \mathbf{u}_{2,i})$ pair are single dimensional, the Cholesky factor of $\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}{}^\mathsf{T} = \frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}_1}\frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}_1}{}^\mathsf{T} + \frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}_2}\frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}_2}{}^\mathsf{T}$ can then be computed by low-rank Cholesky updates of the triangular / diagonal matrix $\frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}_2}$ with each of the columns of $\frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}_1}$. As $\dim(\boldsymbol{u}_1) = L$ is often significantly less than the number of observations being conditioned on $N_\mathbf{x}$, the resulting $\mathcal{O}(LN_\mathbf{x}^2)$ cost of the low-rank Cholesky updates is a significant improvement over the original $\mathcal{O}(N_\mathbf{x}^3)$. For cases in which each $(\mathbf{x}_i, \mathbf{u}_{2,i})$ pair are both vectors of dimension D (i.e. $N_\mathbf{x} = GD$) and so $\frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}_2}$ is block diagonal / triangular, then the Cholesky factorisation of $\frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}_2}\frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}_2}{}^\mathsf{T}$ can be computed at a cost $\mathcal{O}(GD^3)$ for block diagonal, and $\mathcal{O}(G^2D^3)$ for block triangular $\frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}_2}$, with then again $\mathcal{O}(LN_\mathbf{x}^2)$ cost low-rank updates of this Cholesky factor by the columns of $\frac{\partial \boldsymbol{g}_\mathbf{x}}{\partial \boldsymbol{u}_1}$ performed. When $\mathbf{x}_i$ and $\mathbf{u}_{2,i}$ are vectors of differing dimensions, with generally in this case $\dim(\mathbf{u}_{2,i}) > \dim(\mathbf{x}_i)$ due to the requirement the total number of random inputs M is at least $N_\mathbf{x}$, then though we could choose a subset of each $\mathbf{u}_{2,i}$ of equal dimension to $\mathbf{x}_i$ so as to identify a block-triangular component, generally any gain from exploiting this structure will be minimal and in practice it seems likely to be more efficient to compute the Cholesky of $\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}\mathbf{J}_{\boldsymbol{g}_\mathbf{x}}{}^\mathsf{T}$ directly.

E. Human pose and camera model inference

As a second experiment we considered inferring a three-dimensional human pose and camera model from two-dimensional projections of joint positions. We used a 19 joint skeleton model, with a learnt prior distribution over poses parametrised by 47 local joint angles $\mathbf{z}_a$. The pose model was learnt from the *PosePrior* motion capture data-set [1] with a Gaussian *variational autoencoder* (VAE) [15] trained to match the distribution of the motion capture joint angle data. The circular

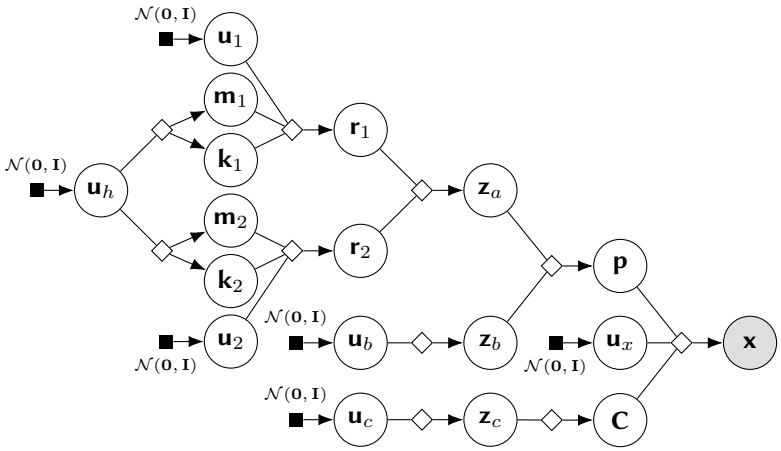


Figure 4. Factor graph of human pose differentiable generative model. The operations corresponding to the deterministic nodes ($\diamond$) in the graph are described in Algorithm 2.

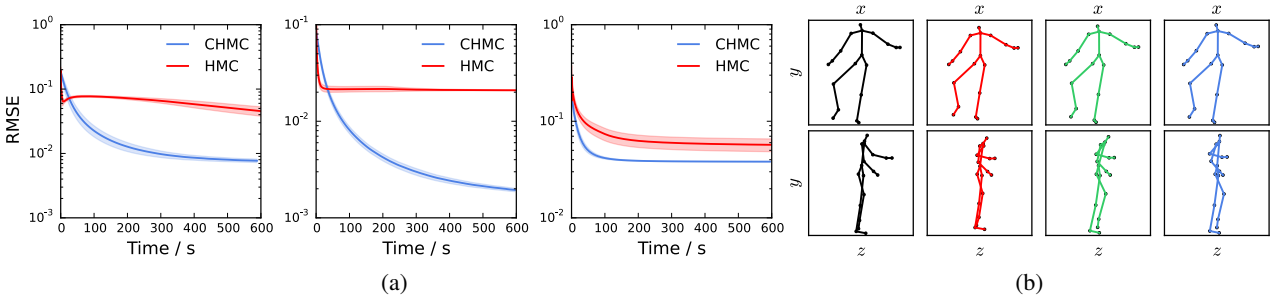


Figure 5. **Human pose** (a) RMSEs of 3D pose posterior mean estimates given binocular projections, using samples from our method (blue) versus running HMC in hierarchical model (red) for three different scenes sampled from the prior. Horizontal axes show computation time to produce number of samples in estimate. Solid curves are average RMSE over 10 runs with different seeds and shaded regions show ± 3 standard errors of mean. (b) Orthographic projections (top: front view, bottom: side view) of 3D poses consistent with monocular projections. Left most pair (black) shows pose used to generate observations, right three show constrained HMC samples.

topology of the angular data is poorly matched by the Euclidean space a Gaussian VAE typically learns a distribution on, and simply 'unwrapping' the angles to e.g. $[-\pi, \pi)$ leads to unnatural discontinuities at the $\pm\pi$ cut-point, this both making the initial learning problem challenging (as there is no in-built prior knowledge of continuity across the cut-point) and tending to lead to a learned latent space less amenable to MCMC inference as 'nearby' poses with one or more joint angles on opposite sides of the cut-point will likely end up corresponding to points far apart in the latent space.

During training we therefore mapped each vector of 47 joint angles $\boldsymbol{z}_a^{(i)}$ (corresponding to a single motion capture datapoint) to a pair of 47-dimensional vectors $(\boldsymbol{r}_1^{(i)}, \boldsymbol{r}_2^{(i)})$ by sampling a Gaussian random vector $\boldsymbol{n}^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and then computing $\boldsymbol{r}_1^{(i)} = \exp \boldsymbol{n}^{(i)} \odot \cos \boldsymbol{z}_a^{(i)}$ and $\boldsymbol{r}_2^{(i)} = \exp \boldsymbol{n}^{(i)} \odot \sin \boldsymbol{z}_a^{(i)}$ and training the VAE to maximise (a variational lower bound) on the joint marginal density of the $\{\boldsymbol{r}_1^{(i)}, \boldsymbol{r}_2^{(i)}\}_i$ pairs. At the cost of doubling the dimension, this leads to an embedding in a Euclidean space which does not introduce any arbitary cut-points and empirically seemed to lead to better sample quality from the learned generative model compared to learning the angles directly. Given the trained model we can generate a vector of angles $\mathbf{z}_a$ using the model by sampling a Gaussian code (latent representation) vector $\mathbf{u}_h$ from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ then sampling a pair of 47-dimensional vectors $\mathbf{r}_1$ and $\mathbf{r}_2$ from the learnt Gaussian decoder model given $\mathbf{u}_h$ (and further Gaussian random input vectors $\mathbf{u}_1$ and $\mathbf{u}_2$), and finally recovering an angle by computing $\mathbf{z}_a = \text{atan2}(\mathbf{r}_2, \mathbf{r}_1)$. The resulting distribution on $\mathbf{z}_a$ is only implicitly defined, but the overall generative model is differentiable with respect to the input vectors $\mathbf{u}_h$, $\mathbf{u}_1$ and $\mathbf{u}_2$.

The *PosePrior* motion capture data includes recordings from only a relatively small number of distinct actors and so limited variation in the 'bone-lengths' of the skeleton model. Therefore a serparate log-normal model for the bone lengths $\mathbf{z}_b$ was fitted using data from the *ANSUR* anthropometric data-set [11], due to symmetry in the skeleton thirteen independent lengths

Algorithm 2 Human pose model generator functions

Require:
$\{\boldsymbol{W}_\ell, \boldsymbol{b}_\ell\}_{\ell=0}^{L}$: parameters of learnt differentiable network model of pose angles;
$\boldsymbol{\mu}_b$, $\boldsymbol{\Sigma}$: mean and covariance of skeleton bone lengths;
$\boldsymbol{\mu}_{c,:2}$, $\boldsymbol{\sigma}_{c,:2}$: means and standard deviations of camera world x, y coordinates;
$\mu_{c,2}$, $\sigma_{c,2}$: mean and standard deviation of logarith of camera world z coordinate;
ϵ : image joint position observation noise standard deviation;
JOINTPOSITIONS : maps pose angles and bone lengths to joint positions;
CAMERAMATRIX : maps camera coordinates to projective camera matrix;
PROJECT : uses camera matrix to map world coordinates to image coordinates;
PARTITION : partitions a vector in a specified number of equal length parts;
FLATTEN : flattens a multidimensional array to a vector.

function $\boldsymbol{g}_{\mathbf{z}}([\mathbf{u}_h; \mathbf{u}_1; \mathbf{u}_2; \mathbf{u}_b; \mathbf{u}_c])$
  $\mathbf{h}_L \leftarrow \textsc{DiffNet}(\mathbf{u}_h)$
  $\mathbf{m}_1, \mathbf{k}_1, \mathbf{m}_2, \mathbf{k}_2 \leftarrow \textsc{Partition}(\mathbf{h}_L, 4)$
  $\mathbf{r}_1 \leftarrow \exp(\mathbf{k}_1) \odot \mathbf{u}_1 + \mathbf{m}_1$
  $\mathbf{r}_2 \leftarrow \exp(\mathbf{k}_2) \odot \mathbf{u}_2 + \mathbf{m}_2$
  $\mathbf{z}_a \leftarrow \text{atan2}(\mathbf{r}_2, \mathbf{r}_1)$
  $\mathbf{z}_b \leftarrow \exp(\boldsymbol{\mu}_b + \boldsymbol{\Sigma}_b \mathbf{u}_b)$
  $\mathbf{z}_{c,:2} \leftarrow \boldsymbol{\sigma}_{c,:2} \odot \mathbf{u}_{c,:2} + \boldsymbol{\mu}_{c,:2}$
  $\mathrm{z}_{c,2} \leftarrow \exp(\sigma_{c,2} \mathrm{u}_{c,2} + \mu_{c,2})$
  return $[\mathbf{z}_a; \mathbf{z}_b; \mathbf{z}_c]$

function $\textsc{DiffNet}(\mathbf{u}_h)$
  $\mathbf{h}_0 \leftarrow \tanh(\boldsymbol{W}_0 \mathbf{u}_h + \boldsymbol{b}_0)$
  for $\ell \in \{1 \ldots L-1\}$
    $\mathbf{h}_\ell \leftarrow \tanh(\boldsymbol{W}_\ell \mathbf{h}_{\ell-1} + \boldsymbol{b}_\ell) + \mathbf{h}_{\ell-1}$
  return $\boldsymbol{W}_L \mathbf{h}_{L-1} + \boldsymbol{b}_L$

function $\boldsymbol{g}_{\mathbf{x}|\mathbf{z}}([\mathbf{z}_a; \mathbf{z}_b; \mathbf{z}_c], \mathbf{u}_x)$
  $\mathbf{P} \leftarrow \textsc{JointPositions}(\mathbf{z}_a, \mathbf{z}_b)$
  $\mathbf{C} \leftarrow \textsc{CameraMatrix}(\mathbf{z}_c)$
  $\mathbf{X} \leftarrow \textsc{Project}(\mathbf{C}, \mathbf{P})$
  return $\textsc{Flatten}(\mathbf{X}) + \epsilon \mathbf{u}_x$

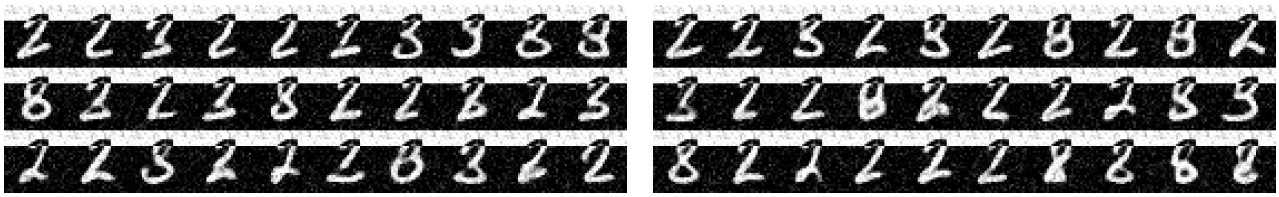
being specified. A simple pin-hole projective camera model with three position parameters $\mathbf{z}_c$ and fixed focal-length was used[3]. A log-normal prior distribution was placed on the depth co-ordinate $\mathrm{z}_{c,2}$ to enforce positivity with normal priors on the other two co-ordinates $\mathrm{z}_{c,0}$ and $\mathrm{z}_{c,1}$.

Given a generated triplet of joint-angles, bone length and camera parameters $\mathbf{z}_a$, $\mathbf{z}_b$ and $\mathbf{z}_c$, a simulated two-dimensional projection of the skeleton $\mathbf{x}$ is produced by first mapping the joint-angles and bone-lengths to a 4×19 matrix of joint positions $\mathbf{P}$ in (homegeneous) world-coordinates by recursing through the skeleton tree. A 3×4 projective camera matrix $\mathbf{C}$ is generated from $\mathbf{z}_c$ and then used to project the world-coordinate joint positions to a 2×19 matrix $\mathbf{X}$ of joint positions in two-dimensional image-coordinates. The projected positions matrix $\mathbf{X}$ is flattened to a vector and a Gaussian vector with standard deviation ϵ added to the projected position vector to give the $19 \times 2 = 38$ dimensional observed vector $\mathbf{x}$. The noise standard deviation ϵ is chosen so that the noise in the projected joint positions is non-obvious in generated projections. The overall corresponding model generator functions $\boldsymbol{g}_{\mathbf{x}|\mathbf{z}}$ and $\boldsymbol{g}_{\mathbf{z}}$ are described procedurally in Algorithm **??** and a factor graph summarising the relationships between the variables in the model shown in Figure 4.

Although the Gaussian observed output noise is necessarily not needed to apply our proposed constrained HMC method as the generator without the final additive noise still defines a valid differentiable generative model, using the noisy observation model means that an explicit hierarchical joint density on is defined on $\{\mathbf{x}, \mathbf{u}_h, \mathbf{r}_1, \mathbf{r}_2, \mathbf{z}_b, \mathbf{z}_c\}$ allowing comparison of our constrained HMC method with (non-constrained) HMC as a baseline. Further adding noise to the output ensures the generator Jacobian is full-rank everywhere and also significantly simplifies the process of finding an initial $\mathbf{u}$ such that the generated $\mathbf{x}$ matches observations.

We first considered binocular pose estimation, with the variables defining the three-dimensional scene information $\mathbf{z}_a$, $\mathbf{z}_b$ and $\mathbf{z}_c$, inferred given a pair of two-dimensional projections from two simulated cameras with a known offset in their positions (in this case the generator function is adjusted accordingly to output an $19 \times 2 \times 2 = 76$ dimensional observed vector $\mathbf{x}$ corresponding to the concatenation of the flattened projected joint positions from both 'cameras'). In this binocular case, the disparity in projected joint positions between the two projections gives information about the distances of the correspondings joints from the image plane in the depth direction and so we would expect the posterior distribution on the three-dimensional pose to be tightly distributed around the true values used to generate the observations. We compared our constrained HMC method to running standard HMC on the conditional density of $\{\mathbf{u}_h, \mathbf{r}_1, \mathbf{r}_2, \mathbf{z}_b, \mathbf{z}_c\}$ given $\mathbf{x}$.

[3]The camera orientation was assumed fixed to avoid replicating the degrees of freedom specified by the angular orientation of the root joint of the skeleton: only the relative camera–skeleton orientation is important.



(a) Constrained HMC samples

(b) HMC in hierarchical model samples

Figure 6. **MNIST** In-painting samples. The top black-on-white quarter of each image is the fixed observed region and the remaining white-on-black region the proposed in-painting. In left-right, top-bottom scan order the images in (a) are consecutive samples from a constrained HMC run; in (b) the images are every 40^{th} sample from a HMC run to account for the $\sim 40\times$ shorter run-time per sample. All images in this run are show in Figure 7.

Figure 5a shows the *root mean squared error* (RMSE) between the posterior mean estimate of the three-dimensional joint positions and the true positions used to generate the observations as the number of samples included in the estimate increases for three test scenes. For both methods the horizontal axis has been scaled by run time. The constrained HMC method (blue curves) tends to give position estimates which converge more quickly to the true position. In this case standard HMC performs relatively poorly despite the signficantly cheaper cost of each integrator step compared to the constrained dynamics. This is at least in part due to the small output noise standard deviation ϵ used which requires a small integrator step to be used to maintain reasonable accept rates.

We also considered inferring 3D scene information from a single 2D projection. Monocular projection is inherently information destroying with significant uncertainty to the true pose and camera parameters which generated the observations. Figure 5b shows pairs of orthographic projections of 3D poses: the left most column is the pose used to generate the projection conditioned on and the right three columns are poses sampled using constrained HMC consistent with the observations. The top row shows front x–y views, corresponding to the camera view though with a orthographic rather than perspective projection, the bottom row shows side z–y views with the z axis the depth from the camera. The dynamic is able to move between a range of plausible poses consistent with the observations while reflecting the inherent depth ambiguity from the monocular projection.

F. MNIST in-painting

As a final experiment we considered inferring in-paintings for a missing region of a digit image $\mathbf{z}$ given knowledge of the rest of the pixel values $\mathbf{x}$. A Gaussian VAE trained on MNIST was used as the generative model, with a 50-dimensional hidden code $\mathbf{h}$. We compared our method to running HMC in the known conditional distribution on $\mathbf{h}$ given $\mathbf{x}$ ($\mathbf{z}$ can then be directly sampled from its Gaussian conditional distribution given $\mathbf{h}$).

Example samples are shown in Figure 6. In this case the constrained and standard HMC approaches appear to be performing similarly, with both able to find a range of plausible in-paintings given the observed pixels. Without cost adjustment the standard HMC samples show greater correlation between subsequent updates, however for a fairer comparison the samples shown were thinned to account for the approximately $40\times$ larger run-time per constrained HMC sample. Although the constrained dynamic does not improve efficiency here neither does it seem to hurt it.

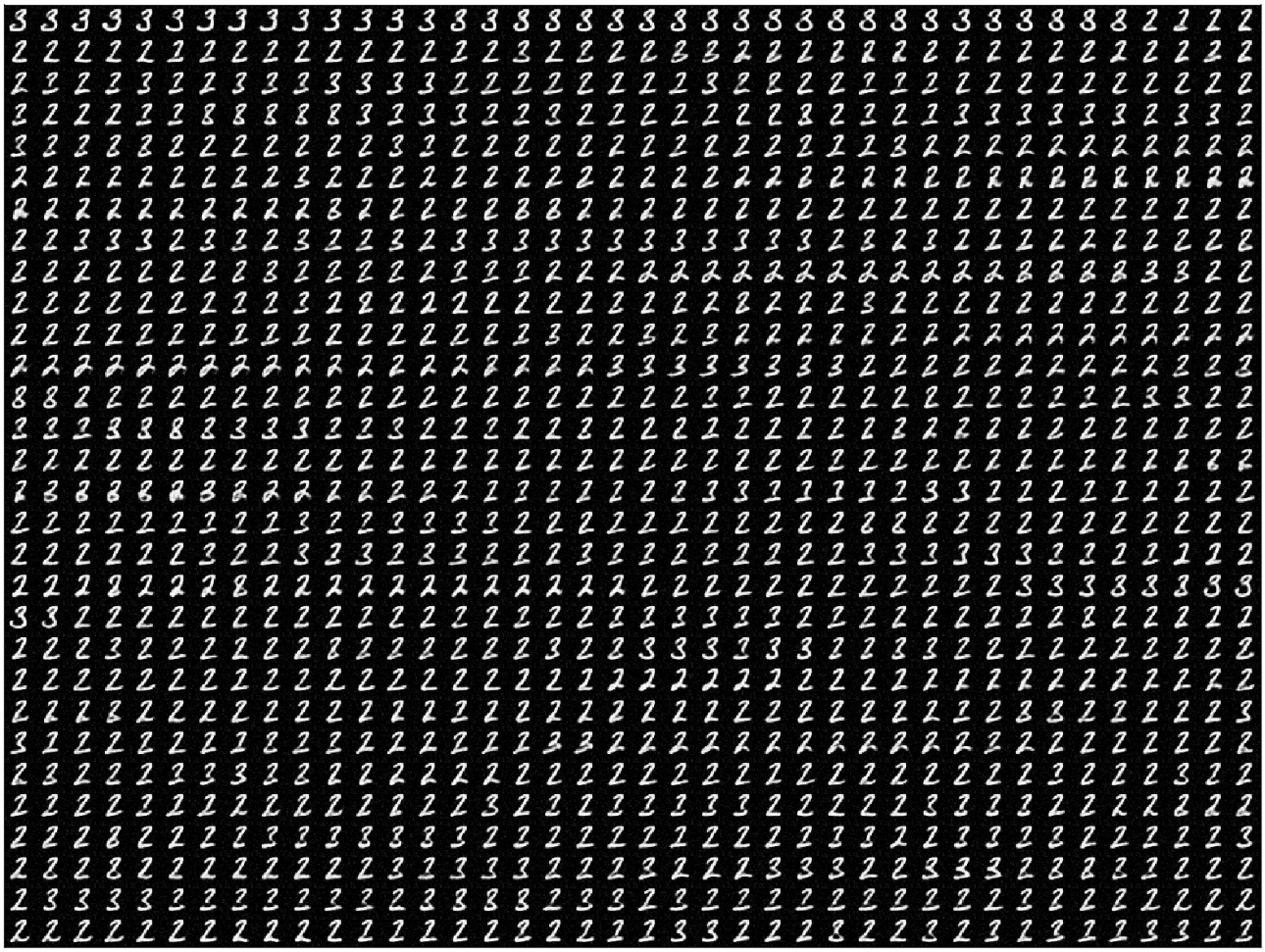


Figure 7. All in-painted MNIST image samples from a single HMC run previously shown thinned by factor 40 in Figure 6b, with samples in chain following left-to-right, top-to-bottom order.