2nd Symposium on Advances in Approximate Bayesian Inference, 2019 1–18

Coping With Simulators That Don't Always Return

Andrew Warrington ANDREW.WARRINGTON@KEBLE.OX.AC.UK
University of Oxford

Saeid Naderiparizi
University of British Columbia

Frank Wood
University of British Columbia

Abstract

Deterministic models are approximations of reality that are often easier to build and interpret than stochastic alternatives. Unfortunately, as nature is capricious, observational data can never be fully explained by deterministic models in practice. Observation and process noise need to be added to adapt deterministic models to behave stochastically, such that they are capable of explaining and extrapolating from noisy data. Adding process noise to deterministic simulators can induce a failure in the simulator resulting in no return value for certain inputs – a property we describe as "brittle." We investigate and address the wasted computation that arises from these failures, and the effect of such failures on downstream inference tasks. We show that performing inference in this space can be viewed as rejection sampling, and train a conditional normalizing flow as a proposal over noise values such that there is a low probability that the simulator crashes, increasing computational efficiency and inference fidelity for a fixed sample budget when used as the proposal in an approximate inference algorithm.

1. Introduction

In order to compensate for epistemic uncertainty due to modelling approximations and unmodeled aleatoric uncertainty, deterministic simulators are often "converted" to "stochastic" simulators by randomly perturbing the state at each time step. In practice, models adapted in this way often provide better inferences (Møller et al., 2011; Saarinen et al., 2008; Lv et al., 2008; Pimblott and LaVerne, 1990; Renard et al., 2013). State-independent white noise with heuristically tuned variance is often used to perturb the state (Adhikari and Agrawal, 2013; Brockwell and Davis, 2016; Fox, 1997; Reddy and Clinton, 2016; Du and Sam, 2006; Allen, 2017; Mbalawata et al., 2013). However, naively adding noise to the state will, in many applications, render the perturbed input state "invalid," inducing failure (Razavi et al., 2019; Lucas et al., 2013; Sheikholeslami et al., 2019). These failures waste computational resources and reduce sample diversity, worsening inference performance.

Examples of failure modes include ordinary differential equation (ODE) solvers not converging to the required tolerance in the allocated time, or, the state crossing into an unhandled configuration, such as solid bodies overlapping. Establishing the cause of failure is non-trivial and hence, the simulation artifact can be sensitive to seemingly inconsequential alterations to the state – a property we describe as "brittle."

The principal contribution of this paper is a technique for minimizing this failure rate. We proceed by first framing sampling from brittle simulators as rejection sampling. We then eliminate rejections by learning the state-dependent density over perturbations that do not induce failure, using conditional autoregressive flows (Papamakarios et al., 2017). Doing so renders the joint distribution unchanged and retains the interpretability afforded by the simulator, but improves sample efficiency. We show that using the learned proposal increases the fidelity of the inference results attainable on a range of examples.

2. Methodology

We denote the brittle deterministic simulator as $f : \mathcal{X} \to \{\mathcal{X},\ \perp\}, \mathcal{X} = \mathbb{R}^D$, where a return value of $\perp$ denotes failure. Over the whole support, f defines a many-to-one function, as many states map to $\perp$, however we only require that f is one-to-one in the accepted region, a condition satisfied by ODE models. A stochastic, additive perturbation to state, denoted $\mathbf{z}_t \in \mathcal{X}$, is proposed such that $\mathbf{x}_t \leftarrow f(\mathbf{x}_{t-1} + \mathbf{z}_t),\ \mathbf{z}_t \sim p(\cdot|\mathbf{x}_{t-1})$, although this is often state independent. We include more detailed derivations in Supplementary Materials Section B.

Brittle Simulators as Rejection Samplers The naive approach to iterate the perturbed system is to repeatedly sample from the proposal distribution and evaluate f until the simulator successfully exists. We begin by showing that this process defines a rejection sampler. The use of f and $p(\mathbf{z}_t|\cdot)$ implicitly specifies a distribution over successfully iterated states, $\overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1})$; and consequently a second distribution over *accepted* perturbations, denoted $\overline{p}(\mathbf{z}_t|\mathbf{x}_{t-1})$, which, under the process outlined above, can be written as:

$$\bar{p}(\mathbf{z}_t|\mathbf{x}_{t-1}) = \begin{cases} \frac{1}{M_p} p(\mathbf{z}_t|\mathbf{x}_{t-1}), & \text{if } f(\mathbf{x}_{t-1} + \mathbf{z}_t) \neq \perp \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where the normalizing constant M_p is the acceptance rate under p. In regions that fail the sample is rejected with certainty. In the accepted region, $\bar{p} \propto p$, which is a sufficient condition for a rejection sampler to be valid, without needing to evaluate M_p or $\bar{p}$. This represents a rejection sampler with an acceptance rule of $\mathbb{I}\left[f(\mathbf{x}_{t-1} + \mathbf{z}_t) \neq \perp\right]$ targeting $\bar{p}(\mathbf{x}_t|\mathbf{x}_{t-1})$.

Change of Variable in Brittle Simulator We now seek to learn a proposal distribution over $\mathbf{z}_t$ values conditioned on the current state $\mathbf{x}_{t-1}$, denoted q_ϕ, parameterized by ϕ, to replace p but placing no mass on regions that are rejected, resulting in an acceptance rate tending to unity. We denote q_ϕ as the proposal we train, which, coupled with the simulator, implicitly defines a proposal over accepted samples, denoted $\overline{q}_\phi$. We wish to minimize the distance between joint distribution implicitly specified over *accepted* iterated states using p, f and $\overline{q}_\phi$, amortized across state space (Le et al., 2016):

$$\phi^* = \arg\min_\phi \mathbb{E}_{p(\mathbf{x}_{t-1})} \left[\mathcal{D}_{\mathrm{KL}} \left[\overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1}) || \overline{q}_\phi(\mathbf{x}_t|\mathbf{x}_{t-1})\right]\right]. \tag{2}$$

Expanding the Kullback–Leibler divergence ($\mathcal{D}_{\mathrm{KL}}$), applying a change of variables yields, and noting that the Jacobian terms can be cancelled yields:

$$\phi^* = \operatorname*{argmax}_\phi \mathbb{E}_{p(\mathbf{x}_{t-1})} \mathbb{E}_{\mathbf{z}_t \sim \overline{p}(\cdot|\mathbf{x}_{t-1})} \left[\log \overline{q}_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})\right]. \tag{3}$$

However, $\bar{q}_\phi$ is defined implicitly *after* rejection sampling, and so we can adapt (1) for q_ϕ and substitute back into (3). Differentiation of the acceptance rate (M_{q_ϕ}) is intractable. However, noting that ϕ^* is a maximizer for *both* q_ϕ and $\bar{q}_\phi$, we can instead optimize $q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})$:

$$\phi^* = \underset{\phi}{\operatorname{argmax}}\, \mathbb{E}_{p(\mathbf{x}_{t-1})}\mathbb{E}_{\mathbf{z}_t\sim\bar{p}(\mathbf{z}_t|\mathbf{x}_{t-1})}\left[\log\left(q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})\right)\right]. \tag{4}$$

By removing the rejection sampler as we have, we have implicitly specified that the proposal distribution must have an acceptance rate of one. This term is differentiable with respect to ϕ and so we can maximize this quantity using stochastic gradients.

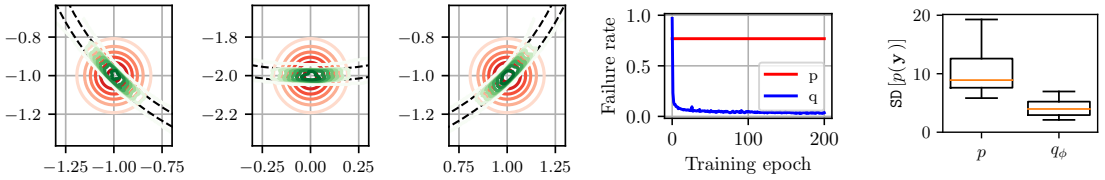
Importantly the flow is not explicitly trained to maximize the acceptance rate. The flow is trained to minimize the KL divergence between the implicitly specified distribution over *accepted samples* and the learned proposal distribution. Accordingly the flow retains the shape of the proposal distribution in regions of state space that do not yield failure (this can be seen by comparing red and green contours in the interior of the dashed lines in Figure 1a) and hence the learned distribution cannot collapse to add trivially small perturbations, as would be the case if we had directly optimized for high acceptance rates. By exploiting change of variables we are able to "project" back through the rejection sampling procedure and hence we can optimize q_ϕ as we do not need to compute the derivative of the rejection rate as we would have otherwise needed to do.

Finally, we note that generation of data training data and learning of the autoregressive flow is a computationally intensive procedure. However simulators can take on the order of seconds to iterate and so the intention of this work is to create a technique that maximizes computational efficiency when deployed. Sampling from the flow takes on the order of milliseconds, can be accelerated using GPUs and scale favourably in the number of samples being produced. Furthermore, the training procedure is performed once and hence represents an offline, one-off cost exchanged for higher efficiency deployment. The training data can also be generated using large-scale distributed computing (as the mini-batching process is inherently embarrassingly parallelizable) that may not be available or practical for use at deployment time.

Implementation We use a conditional masked autoregressive flow (Papamakarios et al., 2017) as the structure of q_ϕ, with 5 single-layer MADE blocks (Germain et al., 2015), 256 hidden units per layer and batch normalization layers at the input to each intermediate MADE block. The dimensionality of the flow is the number of states perturbed in the original model. To introduce conditioning we use the current state vector, $\mathbf{x}_{t-1}$, as input to a hypernetwork (Ha et al., 2016) that outputs the parameters for each layer in the flow. The networks are implemented in PyTorch (Paszke et al., 2017), and optimized using ADAM (Kingma and Ba, 2014).

3. Experiments

We demonstrate on two examples here, and an additional two experiments in the appendix. In these experiments we first aim to demonstrate that learning the required conditional autoregressive flow is tractable and faithfully represents the conditional distribution over accepted perturbations. We then use the learned proposal in a particle-based sequential



(a) (b) (c)

Figure 1: Results for the annulus problem introduced in Section 3.1. 1a indicates the permissible region as a black dashed band, where p and q_ϕ is shown in red and green respectively. The shape of q_ϕ is the same as p *inside* the band, with little mass outside of the band. This shows the flow has learned $\bar{p}$ effectively with a low rejection rate. 1b confirms q_ϕ all-but eliminates rejection. 1c shows the reduction in the variance of the evidence across 100 independent sequential Monte Carlo sweeps of 100 independent datasets.

Monte Carlo state-space inference scheme and show that lower-variance inference results can be obtained for a fixed sample budget.

3.1. Annulus

In this example, the (unknown) true generative model of the observed data is a constant speed circular orbit around the origin in the x-y plane. We perform inference using a misspecified model that only simulates constant velocity forward motion, such that $\mathbf{x}_t \in \mathbb{R}^4$, with Gaussian perturbations to position and velocity. We impose a failure constraint limiting the change in the distance of the point from the origin to a fixed threshold. This condition mirrors the notion that states in brittle simulators have large allowable covariances in particular directions, but very narrow permissible perturbations in other directions.

Figure 1a and Figure 1b shows q_ϕ has effectively learned $\bar{p}(\mathbf{z}_t|\mathbf{x}_{t-1})$, reducing rejection rate under q_ϕ to less than 4% compared to approximately 75% under p. We then use the learned q_ϕ as the proposal in a particle filter (Doucet et al., 2001), an approximate inference algorithm often applied to posterior inference in time-series models. We use a fixed sample budget and hence failed samples are discarded, without retrying a new sample from the proposal. The results in Figure 1c show that we are able to recover lower variance evidence approximations using q_ϕ compared to p, achieving a paired t-test score of < 0.0001. This experiment confirms we are able to learn a proposal that incurs lower rejection, and that reducing the rejection rate increases fidelity of inference (for a fixed computational budget).

3.2. MuJoCo

We now apply our method to the robotics simulator MuJoCo (Todorov et al., 2012), using the built-in example "tosser." MuJoCo allows some overlap between solid objects to simulate the contact dynamics. This is an example of model misspecification borne out of the requirements of reasonably writing a simulator. We therefore place a hard limit on the amount objects are allowed to overlap. We add Gaussian perturbations to the state.

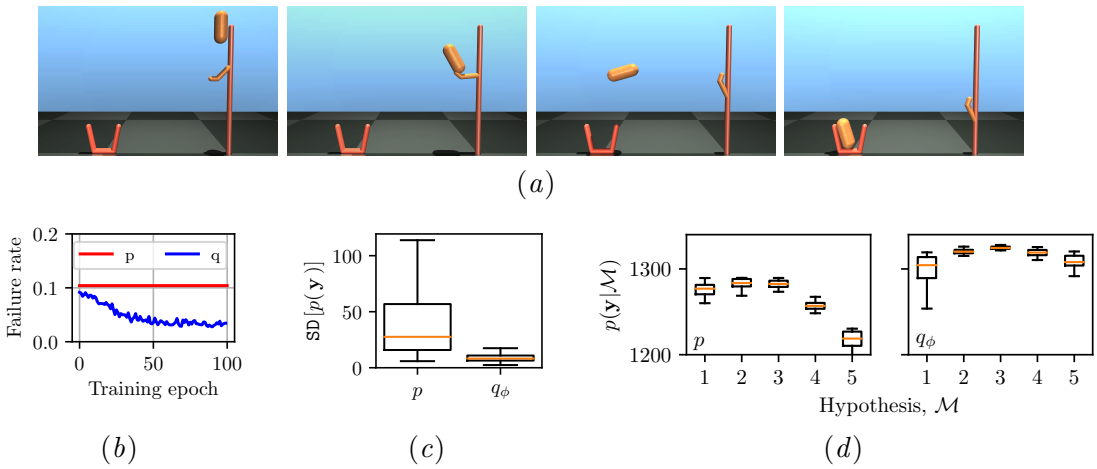


Figure 2: Results of the "tosser" experiment introduced in Section 3.2. 2a shows a typical state evolution. 2b shows the conditional autoregressive flow we learn markedly reduces the number of rejections. 2c shows the results of performing sequential Monte Carlo using p and q_ϕ. 2d shows the results of performing hypothesis testing, where the correct hypothesis (3) not selected using p, but is using q_ϕ.

Figure 2 shows the results of this experiment. Collisions are generally rare events and hence the rejection rate of p is just 10%. Figure 2b shows that the autoregressive flow learns a proposal with a significantly lower rejection rate, reaching 3% rejection. However these rejections are concentrated in the critical regions of state-space and so this reduction yields an large reduction in the variance of the evidence approximation, as shown in Figure 2c. We conclude by applying our method to hypothesis testing, selecting the mass of the capsule. Shown in Figure 2d, using p results in higher variance evidence approximations than when q_ϕ is used, causing p to select the wrong model, with a reasonable level of significance ($p = 0.125$), while using q_ϕ selects the correct hypothesis with $p = 0.0127$.

4. Conclusion

In this paper we have tackled reducing simulator failures caused by naively perturbing the input state. We achieve this by defining these simulators as rejection samplers and learning a conditional autoregressive flow to estimate the state-dependent proposal distribution conditioned on acceptance. We show that using this learned proposal reduces the variance of inference results when used as the proposal in a subsequent approximate inference scheme. This work has readily transferable practical contributions in the scientific community where naively modified simulation platforms are widely deployed.

References

Ratnadip Adhikari and Ramesh K Agrawal. An introductory study on time series modeling and forecasting. *arXiv preprint arXiv:1302.6613*, 2013.

Linda JS Allen. A primer on stochastic epidemic models: Formulation, numerical simulation, and analysis. *Infectious Disease Modelling*, 2(2):128–142, 2017.

Jordan Hylke Boyle, Stefano Berri, and Netta Cohen. Gait modulation in c. elegans: an integrated neuromechanical model. *Frontiers in computational neuroscience*, 6:10, 2012.

Peter J Brockwell and Richard A Davis. *Introduction to time series and forecasting*. springer, 2016.

Laure Coutin, Jean-Marc Guglielmi, and Nicolas Marie. On a fractional stochastic hodgkin–huxley model. *International Journal of Biomathematics*, 11(05):1850061, 2018.

Arnaud Doucet, Nando De Freitas, and Neil Gordon. An introduction to sequential monte carlo methods. In *Sequential Monte Carlo methods in practice*, pages 3–14. Springer, 2001.

Nguyen Huu Du and Vu Hai Sam. Dynamics of a stochastic lotka–volterra model perturbed by white noise. *Journal of mathematical analysis and applications*, 324(1):82–97, 2006.

Neil R Edwards, David Cameron, and Jonathan Rougier. Precalibrating an intermediate complexity climate model. *Climate dynamics*, 37(7-8):1469–1482, 2011.

Maurice F Fallon, Hordur Johannsson, and John J Leonard. Efficient scene simulation for robust monte carlo localization using an rgb-d camera. In *2012 IEEE international conference on robotics and automation*, pages 1663–1670. IEEE, 2012.

Ronald F Fox. Stochastic versions of the hodgkin-huxley equations. *Biophysical journal*, 72 (5):2068–2074, 1997.

Andrea Gamba. Real options valuation: A monte carlo approach. *Faculty of Management, University of Calgary WP*, (2002/3), 2003.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015.

Samuel Gershman and Noah Goodman. Amortized inference in probabilistic reasoning. In *Proceedings of the annual meeting of the cognitive science society*, volume 36, 2014.

Joshua H. Goldwyn and Eric Shea-Brown. The what and where of adding channel noise to the hodgkin-huxley equations. *PLOS Computational Biology*, 7(11):1–9, 11 2011.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

Konstantinos Kalogeropoulos, Gareth O Roberts, Petros Dellaportas, et al. Inference for stochastic volatility models using time change transformations. *The Annals of Statistics*, 38(2):784–807, 2010.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Tuan Anh Le, Atilim Gunes Baydin, and Frank Wood. Inference compilation and universal probabilistic programming. *arXiv preprint arXiv:1610.09900*, 2016.

DD Lucas, R Klein, J Tannahill, D Ivanova, S Brandon, D Domyancic, and Y Zhang. Failure analysis of parameter-induced simulation crashes in climate models. *Geoscientific Model Development*, 6(4):1157–1171, 2013.

Qiming Lv, Manuel K. Schneider, and Jonathan W. Pitchford. Individualism in plant populations: Using stochastic differential equations to model individual neighbourhood-dependent plant growth. *Theoretical Population Biology*, 74(1):74 – 83, 2008. ISSN 0040-5809. doi: https://doi.org/10.1016/j.tpb.2008.05.003.

Isambi S. Mbalawata, Simo Särkkä, and Heikki Haario. Parameter estimation in stochastic differential equations with markov chain monte carlo and non-linear kalman filtering. *Computational Statistics*, 28(3):1195–1223, Jun 2013. ISSN 1613-9658. doi: 10.1007/s00180-012-0352-y.

Jan Kloppenborg Møller, Henrik Madsen, and Jacob Carstensen. Parameter estimation in a simple stochastic differential equation for phytoplankton modelling. *Ecological modelling*, 222(11):1793–1799, 2011.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

Simon M. Pimblott and Jay A. LaVerne. Comparison of stochastic and deterministic methods for modeling spur kinetics. *Radiation Research*, 122(1):12–23, 1990. ISSN 00337587, 19385404.

Saman Razavi, Razi Sheikholeslami, Hoshin V. Gupta, and Amin Haghnegahdar. Vars-tool: A toolbox for comprehensive, efficient, and robust sensitivity and uncertainty analysis. *Environmental Modelling & Software*, 112:95 – 107, 2019. ISSN 1364-8152.

Krishna Reddy and Vaughan Clinton. Simulating stock prices using geometric brownian motion: Evidence from australian companies. *Australasian Accounting, Business and Finance Journal*, 10(3):23–47, 2016.

Philippe Renard, Andres Alcolea, and D Gingsbourger. Stochastic versus deterministic approaches. In *Environmental Modelling: Finding Simplicity in Complexity, Second Edition (eds J. Wainwright and M. Mulligan)*, pages 133–149. Wiley Online Library, 2013.

Antti Saarinen, Marja-Leena Linne, and Olli Yli-Harja. Stochastic differential equation model for cerebellar granule cell excitability. *PLOS Computational Biology*, 4(2):1–11, 02 2008. doi: 10.1371/journal.pcbi.1000004.

R. Sheikholeslami, S. Razavi, and A. Haghnegahdar. What do we do with model simulation crashes? recommendations for global sensitivity analysis of earth and environmental systems models. *Geoscientific Model Development Discussions*, 2019:1–32, 2019. doi: 10.5194/gmd-2019-17.

Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. *Artificial intelligence*, 128(1-2):99–141, 2001.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

Appendix A. Background

A.1. Smoothing Deterministic Models

Deterministic simulators are often stochastically perturbed to increase the diversity of the achievable simulations and to fit data more effectively. White noise perturbation to time series systems is common, such as the widely used ARMA models (Adhikari and Agrawal, 2013; Brockwell and Davis, 2016). The most straightfoward example of this however is the widely used Kalman filter (Kalman et al., 1960). The Kalman filter, at its core, is deterministic transition model which is then perturbed with additive Gaussian noise. The form of the process and noise kernels are chosen such the system has a closed form representation. Without the additive process noise, the Kalman filter is deterministic and would be unable to represent the variability in the real-world.

More complex systems cannot be analyzed in closed form like the Kalman filter. Accordingly deterministic simulators of the dynamics with stochastic perturbations and numerical methods are used in practice. Specific examples of such systems that are: stochastic Hodgkin Huxley models of neural dynamics (Fox, 1997; Coutin et al., 2018; Goldwyn and Shea-Brown, 2011; Saarinen et al., 2008), computational finance analysis of asset prices (Gamba, 2003; Reddy and Clinton, 2016; Kalogeropoulos et al., 2010), predator-prey dynamics (Du and Sam, 2006), epidemiology (Allen, 2017) and mobile robotics (Thrun et al., 2001; Fallon et al., 2012).

A.2. Simulator Failure

As simulators become more complex, guaranteeing robustness is more difficult, and individual function evaluations are more expensive. Lucas et al. (2013) and Edwards et al. (2011) establish the sensitivity of earth science models to static input parameter values by building a discriminative classifer for parameters that induce failure. Sheikholeslami et al. (2019) take an alternative approach instead treating simulator failure as an imputation problem, fitting a function regressor to predict the outcome of the failed experiment given the neighbouring experiments that successfully terminated. However these methods are limited by the lack of clear probabilistic interpretation in terms of the originally specified joint distribution in time series models and their ability to scale to high dimensions.

A.3. Autoregressive Flows

Autoregressive flows (AFs) (Papamakarios et al., 2017) are a flexible class of density estimators. AFs define a density, $q_\phi(\mathbf{x})$, trainable using stochastic gradient descent to approximate the target distribution $p(\mathbf{x})$, by minimizing the KL-divergence between the target distribution and the approximation:

$$\phi^* = \arg\min_\phi \mathbb{KL}\left[p(\mathbf{x})||q_\phi(\mathbf{x})\right], \tag{5}$$

AFs operate by transforming samples from a "base distribution" through a series of learned warpings, interpretable as a change of variables, into samples distributed according to the target distribution. The flow layers are designed such that the required Jacobians and inverses are cheaply computable.

A popular flow structure is the masked autoencoder for distribution estimation (Germain et al., 2015), or MADE, that facilitates GPU-based parallelization. Multiple MADE blocks are used in masked autoregressive flows (MAF) (Papamakarios et al., 2017) overcoming the ordering dependency of autoregressive flows. AFs are also capable of learning conditional distributions by making the parameters of the flow dependent on the data using hypernetworks (Ha et al., 2016).

Appendix B. Methodology

We include here a more complete derivation of the results presented in the main text.

The overarching aim of this work is to develop a flexible proposal over perturbations that places minimal mass on perturbations that cause the simulator to not return a value, while not changing the originally specified model. Doing so reduces the wasted computational cost incurred by simulations failing, and also increases the effective sample size for a given sample budget.

We consider deterministic models, expressed as simulators, describing the time-evolution of a state $\mathbf{x}_t \in \mathcal{X}$, where we denote application of the simulator iterating the state as $\mathbf{x}_t \leftarrow f(\mathbf{x}_{t-1})$. However, brittle simulators fail for "invalid" inputs, which we denote as a return value of $\perp$ (read as "bottom") from the simulator. Hence the complete definition of f is $f : \mathcal{X} \to \{\mathcal{X}, \perp\}$. We denote the region of valid inputs as $\mathcal{X}_A \subset \mathcal{X}$, and the region of invalid inputs as $\mathcal{X}_R \subset \mathcal{X}$, such that $\mathcal{X}_A \cup \mathcal{X}_R = \mathcal{X}$, where the boundary between these regions is unknown. Over the whole support, f defines a many-to-one function, as all $\mathcal{X}_R$ maps to $\perp$. However, the algorithm we go on to derive only requires that f is one-to-one in the accepted region. This is not uncommon in real simulators, and is satisfied by, for example, ODE models.

A stochastic, additive perturbation to state, denoted $\mathbf{z}_t \in \mathcal{X}$, is applied to induce a distribution over states. The distribution of this perturbation is denoted $p(\mathbf{z}_t|\mathbf{x}_{t-1})$, although, in practice, this distribution is often state independent. The iterated state is therefore calculated as $\mathbf{x}_t \leftarrow f(\mathbf{x}_{t-1} + \mathbf{z}_t)$. We define the random variable $A_t \in \{0, 1\}$ to denote whether the perturbation (as $\mathbf{x}_{t-1}$ is being conditioned on) is accepted.

B.1. Brittle Simulators as Rejection Samplers

The naive approach to sampling from the perturbed system, shown in Algorithm 1, is to repeatedly sample from the proposal distribution and evaluate f until the simulator successfully exists. This procedure defines $A_t = \mathbb{I}\left[f(\mathbf{x}_{t-1} + \mathbf{z}_t) \neq \perp\right]$, $\mathbf{z}_t \sim p(\mathbf{z}_t|\mathbf{x}_{t-1})$, i.e. successfully iterated samples are accepted with certainty. This approach incurs significant wasted computation as the simulator must be called repeatedly, with failed iterations being discarded. Therefore the objective of this work is to derive a more efficient sampling mechanism. We begin by showing that Algorithm 1 defines a rejection sampler, with a specific form, targeting the space of successfully iterated states. This reasoning is illustrated in Figure 3.

The behavior of f and the distribution $p(\mathbf{z}_t|\cdot)$ implicitly define a distribution over successfully iterated states. We denote this "target" distribution as $\overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1}) = p(\mathbf{x}_t|\mathbf{x}_{t-1}, A_t = 1)$, where the bar indicates that the sample was accepted, and hence places no probability mass on failures. Note there is no bar on $p(\mathbf{z}_t|\mathbf{x}_{t-1})$ above, indicating that it is defined

Algorithm 1 Sampling from a brittle simulator.

Data: Current state $\mathbf{x}_{t-1}$, brittle simulator f, perturbation proposal $p(\mathbf{z}_t|\mathbf{x}_{t-1})$.
Result: Iterated state $\mathbf{x}_t$ and perturbation $\mathbf{z}_t$.
$\mathbf{x}_t \leftarrow \perp$
while $\mathbf{x}_t == \perp$ **do**
  $\mathbf{z}_t \sim p(\mathbf{z}_t|\mathbf{x}_{t-1})$
  $\mathbf{x}_t \leftarrow f(\mathbf{x}_{t-1} + \mathbf{z}_t)$
end
return $\mathbf{x}_t, \mathbf{z}_t$

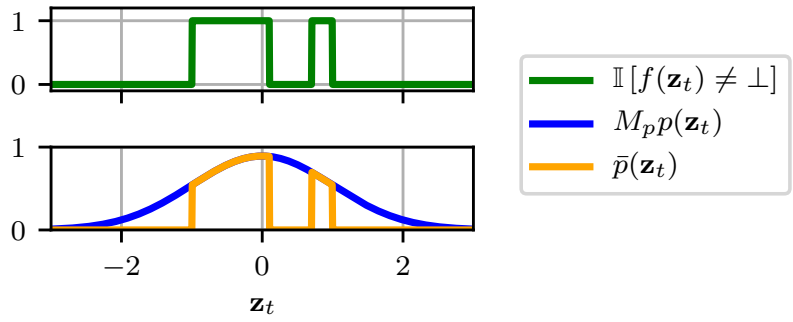


Figure 3: Graphical representation of how a brittle deterministic simulator acts as a rejection sampler, targeting $\bar{p}(\mathbf{z}_t|\mathbf{x}_{t-1})$. For clarity here we assume $\mathbf{x}_{t=1} = 0$ and $\mathbf{z}_t$ is independent of $\mathbf{x}_t$. The simulator, $f(\mathbf{z}_t)$, returns $\perp$ for some unknown input regions, shown in green. The proposal over $\mathbf{z}_t$ is shown in blue. The target distribution, $\bar{p}(\mathbf{z}_t)$, shown in orange, is implicitly defined as $\bar{p}(\mathbf{z}_t) = \frac{1}{M_p} p(\mathbf{z}_t)\mathbb{I}[f(\mathbf{z}_t) \neq \perp]$, where M_p is the normalizing constant from $\bar{p}$, equal to the acceptance rate. Accordingly the proposal distribution, scaled by M_p, is *exactly* equal to $\bar{p}(\mathbf{z}_t)$ in the accepted region. Therefore sampling from p until f successfully exits, as in Algorithm 1, can be seen as constructing a rejection sampler with proposal $p(\mathbf{z}_t)$, and acceptance ratio, $\frac{\bar{p}(\mathbf{z}_t)}{Mp(\mathbf{z}_t)}$. This reduces to $\mathbb{I}[f(\mathbf{z}_t) \neq \perp]$, requiring no additional parameters.

with no knowledge of the accept/reject behaviors of f and hence probability mass may be placed on regions that yield failure. The functional form of $\bar{p}$ is unavailable, and the density cannot be evaluated for any input value. Importantly, $\bar{p}$ is the distribution specified a-priori by the modeler, sampled from by the entire simulation pipeline, and hence any algorithm we develop must also target $\bar{p}(\mathbf{x}_t|\mathbf{x}_{t-1})$.

The existence of $\bar{p}(\mathbf{x}_t|\mathbf{x}_{t-1})$ implies the existence of a second distribution: the distribution over *accepted* perturbations, denoted $\overline{p}(\mathbf{z}_t|\mathbf{x}_{t-1})$. Note that this distribution is also conditioned on acceptance under the chosen simulator indicated be the presence of a bar. We assume f is one-to-one in the accepted regime, and so the change of variables rule can be applied to directly relate this to $\overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1})$. Under our initial algorithm for sampling

from a brittle simulator we can therefore write the following identity:

$$\bar{p}(\mathbf{z}_t|\mathbf{x}_{t-1}) = \begin{cases} \frac{1}{M_p} p(\mathbf{z}_t|\mathbf{x}_{t-1}), & \text{if } f(\mathbf{x}_{t-1} + \mathbf{z}_t) \neq \perp \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where the normalizing constant M_p is the acceptance rate under p. By inspecting (6), accepting with certainty perturbations that exit successfully can be seen as proportionally shifting mass from regions of p where the simulator does not exit to regions where it does.

In a rejection sampler, the probability of accepting a proposed sample is proportional to the ratio between the target distribution and the proposal, scaled by a constant such that:

$$Mp(\mathbf{z}_t|\cdot) \geq \bar{p}(\mathbf{z}_t|\cdot) \; \forall \mathbf{z}_t \in \mathcal{X}. \tag{7}$$

As we have already stated, we cannot evaluate the target density, but we can establish if the density is non-zero (indicated by the simulator not failing). A sufficient condition to ensure the correctness of a rejection sampler in this scenario is that the proposal density is proportional to the target density wherever the target density has support. Applying this condition to our scenario implies that if the simulator fails, the density under the target distribution is known to be zero and the sample should be rejected with certainty, regardless of the density under the proposal distribution. In the accepted region, the sample should be accepted with probability $\bar{p}(\mathbf{z}_t|\cdot)/Mp(\mathbf{z}_t|\cdot)$, where M is selected to satisfy (7). However, from (6), it can be seen $\bar{p} \propto p$ hence proposal and target are proportional irrespective of the choice of M, and the value of M_p, satisfying the above criteria. The acceptance rule of the rejection sampler is therefore reduced to $\mathbb{I}\left[f(\mathbf{x}_{t-1} + \mathbf{z}_t) \neq \perp\right]$. Importantly, we do not need to evaluate M_p, M, or $\bar{p}$ to use Algorithm 1 as a valid rejection sampler.

This simple probabilistic interpretation of the behavior of the simulation process enables us to establish (6) as a definition of $\bar{p}$ valid across the entire input domain of f – a definition we now exploit to learn an efficient proposal.

B.2. Change of Variable in Brittle Simulator

We now derive how we can learn the proposal distribution, denoted q_ϕ parameterized by ϕ, to replace p, such that the acceptance rate under q_ϕ (denoted M_{q_ϕ}) tends towards unity, minimizing wasted computation, while also retaining the same joint distribution as the originally specified model. We denote q_ϕ as the proposal we train, which, coupled with the simulator, implicitly define a proposal over accepted samples, denoted $\overline{q}_\phi$.

Expressing this mathematically, we wish to minimize the distance between joint distribution implicitly specified over *accepted* iterated states using the a-priori specified proposal distribution p, and the joint distribution defined implicitly as $\overline{q}_\phi$:

$$\phi^* = \arg\min_\phi \mathbb{E}_{p(\mathbf{x}_{t-1})} \left[\mathcal{D}_{\mathrm{KL}} \left[\bar{p}(\mathbf{x}_t|\mathbf{x}_{t-1}) || \overline{q}_\phi(\mathbf{x}_t|\mathbf{x}_{t-1})\right]\right], \tag{8}$$

where we select the Kullback-Leibler (KL) divergence as the metric of distance between distributions. The outer expectation defines this objective as amortized across state space. As is standard in amortized and compiled inference methods we can generate the samples by directly sampling from the model (Le et al., 2016; Gershman and Goodman, 2014).

We eventually perform this minimization using stochastic gradient descent, and so this expectation defines the distribution from which we sample the minibatches used, and so we drop this expectation for compactness.

Expanding the KL term yields:

$$\phi^* = \arg\min_{\phi} \mathbb{E}_{\mathbf{x}_t \sim \overline{p}(\cdot|\mathbf{x}_{t-1})} \left[\log\left(w\right)\right], \tag{9}$$

$$w = \frac{\overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1})}{\overline{q}_\phi(\mathbf{x}_t|\mathbf{x}_{t-1})}. \tag{10}$$

Noting that $\overline{q}_\phi$ and $\overline{p}$ are defined only on accepted samples, where f is one-to-one, we can apply a change of variables defined for $\overline{q}_\phi$ as:

$$\overline{q}_\phi(\mathbf{x}_t|\mathbf{x}_{t-1}) = \overline{q}_\phi(f^{-1}(\mathbf{x}_t)|\mathbf{x}_{t-1}) \left|\frac{\mathrm{d}f^{-1}(\mathbf{x}_t)}{\mathrm{d}\mathbf{x}_t}\right|, \tag{11}$$

and likewise for p. This transforms the distribution over $\mathbf{x}_t$ into a distribution over $\mathbf{z}_t$ and a Jacobian term:

$$w = \frac{\overline{p}(f^{-1}(\mathbf{x}_t)|\mathbf{x}_{t-1}) \left|\frac{\mathrm{d}f^{-1}(\mathbf{x}_t)}{\mathrm{d}\mathbf{x}_t}\right|}{\overline{q}_\phi(f^{-1}(\mathbf{x}_t)|\mathbf{x}_{t-1}) \left|\frac{\mathrm{d}f^{-1}(\mathbf{x}_t)}{\mathrm{d}\mathbf{x}_t}\right|}. \tag{12}$$

Noting that the same Jacobian terms appear in the numerator and denominator we are able to cancel these:

$$w = \frac{\overline{p}(f^{-1}(\mathbf{x}_t)|\mathbf{x}_{t-1})}{\overline{q}_\phi(f^{-1}(\mathbf{x}_t)|\mathbf{x}_{t-1})}, \tag{13}$$

taking care to also apply the change variables in the distribution we are sampling from in (9). We can now discard the remaining p term as it is independent of ϕ, and noting that $f^{-1}(\mathbf{x}_t) = \mathbf{x}_{t-1} + \mathbf{z}_t$ we can write:

$$\phi^* = \operatorname*{argmax}_{\phi} \mathbb{E}_{\mathbf{z}_t \sim \overline{p}(\cdot|\mathbf{x}_{t-1})} \left[\log \overline{q}_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})\right]. \tag{14}$$

It can now be read off that minimizing the KL stated in (9) can be performed by setting $\overline{q}_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})$ equal to $\overline{p}(\mathbf{z}_t|\mathbf{x}_{t-1})$. Had we have discarded p a step earlier, we would have been unable to eliminate the Jacobian terms inside the logarithm.

However, this distribution is defined *after* rejection sampling, and can only be defined as in (6):

$$\overline{q}_\phi(\mathbf{z}_t|\mathbf{x}_{t-1}) = q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1}, A_t = 1), \tag{15}$$

$$= \begin{cases} \frac{1}{M_{q_\phi}} q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1}) & \text{if } f(\mathbf{x}_{t-1} + \mathbf{z}_t) \neq \perp, \\ 0 & \text{otherwise,} \end{cases} \tag{16}$$

denoting M_{q_ϕ} as the acceptance rate under q_ϕ. Note again that q_ϕ is not dependent on the accept/reject characteristics of f. Differentiation of M_{q_ϕ} is intractable. Further, there

is an infinite family of q_ϕ proposals that yield $\overline{p} = \overline{q}_\phi$, that have non-zero rejection rates. However, we observe that ϕ^* is a maximizer for *both* q_ϕ and $\bar{q}_\phi$ in the limit of zero rejection, and so we can instead optimize $q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})$:

$$\phi^* = \underset{\phi}{\operatorname{argmax}}\, \mathbb{E}_{\mathbf{z}_t \sim \overline{p}(\mathbf{z}_t|\mathbf{x}_{t-1})} \left[\log\left(q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})\right)\right], \tag{17}$$

with no consideration of the rejection behavior. Additionally, by removing the rejection sampler as we have, we have implicitly specified that the proposal distribution must have an acceptance rate of one. This term is differentiable with respect to ϕ and so we can maximize this quantity using stochastic gradients, where samples from the outer expectation over $\mathbf{x}_{t-1}$ defines the distribution from which we sample minibatches from.

This expression shows that we can learn the distribution over accepted $\mathbf{x}_t$ values by learning the distribution over $\mathbf{z}_t$, *without* needing to calculate the Jacobian or inverse of the transformation defined by f. We can now perform density estimation on the *accepted* samples from the a-priori specified rejection sampler to learn a proposal for *accepted* $\mathbf{x}_t$ samples, thus minimizing wasted computation, targeting the same overall joint distribution, and retaining interpretability by utilizing the simulator.

Appendix C. Experiments

In this section we include additional results figures and experimental details for the annulus and MuJoCo experiment presented in the main text, along with an additional two experiments.

C.1. Additional Experiment – Bouncing Balls

Our first additional example uses a simulator of balls elastically bouncing in a square enclosure, as shown in Figure 4a. The dimensionality of the state vector, $\mathbf{x}_t$, is four times the number of balls – the x-y coordinate and velocity of the centre of mass, per ball. We add a small amount of Gaussian noise at each iteration to the position and velocity of each ball. This perturbation induces the possibility that two balls overlap, or, a ball intersects with the wall. Both of these represent invalid physical arrangements and so the simulator returns $\perp$ for such configurations. We note that here, we are conditioning on the state of *both* balls simultaneously, and proposing the perturbation to the state *jointly*.

Figure 4 displays the results of this experiment. Figure 5 shows the distribution over x-y perturbations of a single ball, conditioned on the other ball being static and stationary. Green contours show the perturbations learned by autoregressive flow such that failure is not induced. In Figure 4c we plot the rejection rate under p and q_ϕ as a function of the position of the first ball, with the second ball fixed in the position shown, showing that rejection has been all but eliminated. We again see a reduction in the variance of the evidence approximation when comparing p and q in an SMC scheme, as shown in Figure 4d. This example demonstrates the applicability of the autoregressive flow; but also demonstrates how a seemingly simple simulator becomes brittle when naively perturbed.

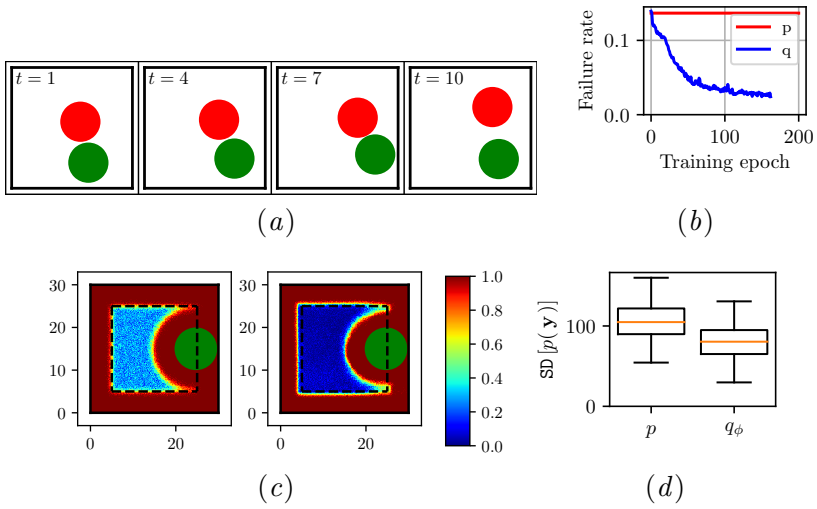


Figure 4: Results of the bouncing balls experiment introduced in Section C.1, with two radius 5, unit mass balls in an enclosure of size 30. 4a shows an example trajectory of the system. 4b shows the autoregressive flow learning to nearly eliminate rejections. 4c shows the rejection rate as a function of the position of the first ball, with the second ball in the position shown. The trained proposal (right) has all but eliminated rejection in the permissible space compared to the a-priori specified proposal (left). The rejection rate under p is much higher in the interior as the second ball may also leave the enclosure, whereas q_ϕ has practically eliminated rejection by *jointly* proposing perturbations. 4d shows the reduction in variance achieved by using q_ϕ. Although the reduction appears more modest compared to, say, the annulus example, it still achieves a paired t-test score of < 0.0001, indicating a strong level of statistical significance.

C.2. Additional Experiment – Neuroscience Simulator

We conclude by applying our algorithm to a simulator for the widely studied *Caenorhabditis elegans* roundworm. WormSim, presented by Boyle et al. (2012), is a simulator of the body of the worm, driven by a surrogate for the true neural architecture of *Caenorhabditis elegans*, and uses a 510 dimensional state representation. We apply perturbations to the 98 dimensional subspace defining the 49 x-y coordinate pairs physical position of the worm, while conditioning on the full 510 dimensional state vector. The expected rate of failure increases sharply as a function of the scale of the perturbation applied, as shown in Figure 6a, as the integrator used in WormSim is unable to integrate highly perturbed states.

We then train a autoregressive flow targeting $\bar{p}$, where the rejection rate during training is shown in Figure 6b. We see that we are able to learn an autoregressive flow with lower rejection rates, reaching approximately 53% rejection, for a p with approximately 75% rejection. Although the rejection rate is higher than ultimately desired, we include this example to show how rejections occur in real simulators through integrator failure. We believe larger flows with regularized parameters can reduce the rejection rate further.

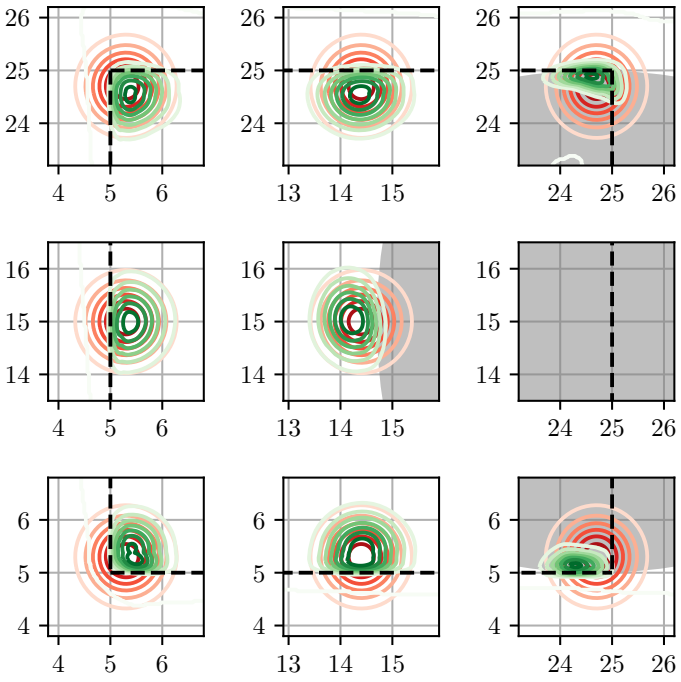


Figure 5: Shown is the a-priori specified proposal distribution, p, over the perturbation to the position of the first ball specified in the model, and the learned proposal, q_ϕ in green, for the bouncing balls experiment introduced in Section C.1. The edge of the permissible region of the enclosure is shown as a black dashed line. The second ball is fixed at $[25, 15]$, and the resulting invalid region induced shaded. The flow has learned to deflect away from the disallowed regions.

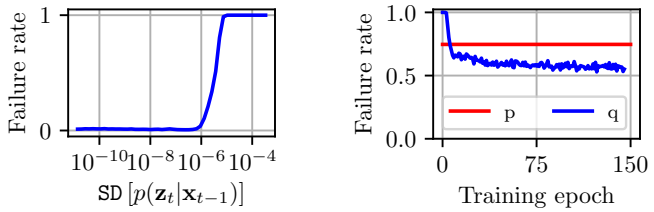


Figure 6: Results from the WormSim example introduced in Section **??**. 6a: The rate at which the simulator fails increases sharply as a function of the standard deviation of the applied perturbation. **??**: The reduction in rejections during training.

C.3. Additional Experimental Details – Annulus

The procedure we used for generating the true data is

$$
\begin{aligned}
&\delta t = 1,\\
&t \in [0, \ldots, 100],\\
&x_0 \sim \mathcal{N}(0, 1),\\
&y_0 \sim \mathcal{N}(0, 1),\\
&r = \sqrt{x_0^2 + y_0^2},\\
&s_0 \sim \mathcal{N}(0, 0.1\sqrt{2}),\\
&a_0 \leftarrow \arctan\left(\frac{y_t}{y_t}\right),\\
&\dot{x}_0 \sim -s_0 \times \sin(a_0),\\
&\dot{y}_0 \sim s_0 \times \cos(a_0),\\
&x_t \leftarrow x_{t-1} + \delta t \times \dot{x}_{t-1},\\
&y_t \leftarrow y_{t-1} + \delta t \times \dot{y}_{t-1},\\
&s_t \leftarrow \sqrt{\dot{x}_{t-1}^2 + \dot{y}_{t-1}^2},\\
&a_t \leftarrow \arctan\left(\frac{y_t}{y_t}\right),\\
&\dot{x}_t \leftarrow -s_t \times \sin a_t,\\
&\dot{y}_t \leftarrow s_t \times \cos a_t,\\
&\mathbf{y}_t \leftarrow [\mathcal{N}(x_t, 0.1),\ \mathcal{N}(y_t, 0.1)],
\end{aligned}
$$

where $\mathbf{y}_t$ denotes the t^{th} observation.

The model used at inference is:

$$
\begin{aligned}
&\delta t = 1,\\
&t \in [0, \ldots, 100],\\
&x_0 \sim \mathcal{N}(0, 1),\\
&y_0 \sim \mathcal{N}(0, 1),\\
&\dot{x}_0 \sim \mathcal{N}(0, 0.1),\\
&\dot{y}_0 \sim \mathcal{N}(0, 0.1),\\
&x_t \leftarrow x_{t-1} + \delta t \times \dot{x}_{t-1} + z_{x_t},\ z_{x_t} \sim \mathcal{N}(0, 0.1),\\
&y_t \leftarrow y_{t-1} + \delta t \times \dot{y}_{t-1} + z_{y_t},\ z_{x_t} \sim \mathcal{N}(0, 0.1),\\
&\dot{x}_t \leftarrow \dot{x}_{t-1} + z_{\dot{x}_t},\ z_{x_t} \sim \mathcal{N}(0, 0.1),\\
&\dot{y}_t \leftarrow \dot{y}_{t-1} + z_{\dot{y}_t},\ z_{y_t} \sim \mathcal{N}(0, 0.1),
\end{aligned}
$$

Failure is defined as the change in radius being greater than 0.03.

To compute the variances of the SMC sweep we generate 100 random traces. We then perform 50 SMC sweeps per trace, using 100 particles, and compute the evidence.

C.4. Additional Experimental Details – MuJoCo

We use the configuration "tosser" included in MuJoCo Todorov et al. (2012), only modifying it by removing the second unused bucket. We use completely standard simulation configuration. We introduce the limit on overlap leveraging MuJoCos in-built collision detection, rejecting overlaps above 0.005. Typical overlaps in the standard execution of MuJoCo are below this limit. An integration time of 0.002 is used.

We observe only the x-y position of the capsule with Gaussian distributed noise, with standard deviation 0.1. We perturb the x-y position and velocity of the capsule with Gaussian distributed noise, with standard deviation 0.005 and 0.1 respectively. We perturb the angle and angular velocity of the capsule with Gaussian distributed noise, with standard deviation 0.05 and 0.05 respectively. These values were chosen to be in line with typical simulated values in the tosser example.

We place a prior over the initial position and velocity with standard deviation 0.01 for positions and 0.1 for velocities, and mean equal to their true position.

In this, the state input to the normalizing flow is the position and angle, and derivatives, of the capsule, as well as the state of the actuator. The actuators state is unobserved and is not perturbed under the model. We also input time into the normalizing flow as the control dynamics are not constant with time.

To compute the variances of the SMC sweep we generate 50 random traces. We then perform 20 SMC sweeps per trace, using 100 particles, and compute the evidence.