Conditional Flow Matching for Time Series Modelling

Ella Tamir [*1] **Najwa Laabid** [*1] **Markus Heinonen** [1] **Vikas Garg** [1 2] **Arno Solin** [1]

Abstract

Learning dynamical systems from long trajectories is a challenging problem due to the complexity of the loss landscape. Inspired by conditional flow matching in generative modelling, we propose a new approach for training neural ODEs based on regressing vector fields of conditional probability paths defined per trajectory. Our Conditional Flow Matching for Time Series (CFM-TS) objective outperforms neural ODEs trained with the adjoint method on three simulated tasks, including a pendulum system where the neural ODE does not converge at all.

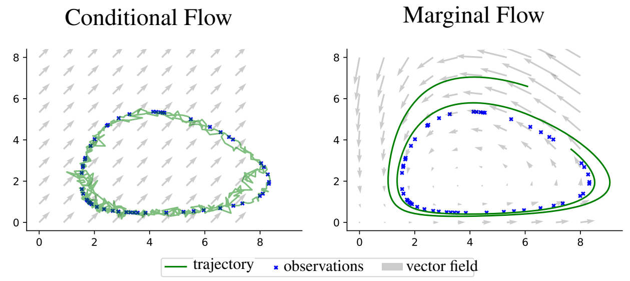

Figure 1. Illustrating CFM-TS using the Lotka-Volterra system. Simple conditional flows are approximated for each trajectory (*left*), and combined to model the more complex, true underlying dynamics (*right*).

1. Introduction

Learning the unlerdying dynamics of a system from observed data can be achieved by approximating its vector field with a neural network, a concept known as Neural ODEs (Chen et al., 2018b; Rubanova et al., 2019; Yildiz et al., 2019). However, the complexity of the loss landscape for these models increases with the length of the observed trajectories, such that their training fails to converge for even moderately long observation horizons (Ribeiro et al., 2020; Metz et al., 2022). Early solutions split long trajectories into smaller segments and ensure the continuity of the probabilistic model through constraints (Hedge et al., 2022; Iakovlev et al., 2023). Yet, fitting long trajectories without discrete approximations remains an open problem.

The challenge of learning complex continuous dynamics arises in another setting, namely when training continuous normalizing flows (Chen et al., 2018a) in generative modelling (Grathwohl et al., 2019; Lipman et al., 2023). The goal here is to learn an infinite number of transformations (expressed as an ODE) mapping samples from p_0 to samples from an unknown data distribution q. One approach to training CNFs is Conditional Flow Matching (CFM), which approximates the time derivative of the flow using vector fields and probability paths defined *per sample* (Lipman et al., 2023). Optimizing a neural network to match the per-sample (i.e. conditional) vector fields is equivalent to optimizing for the original complex flow dynamics, breaking down the task of learning complex ODEs into combining simpler, pre-defined vector fields.

We adapt the idea of combining per-sample flows from CFM to learning expressive time-series models. We start by showing that the conditional flow matching framework is easily extendable to conditioning on observed trajectories instead of single samples. We then explore two methods to define our conditional flows: linearly interpolating between the observations via Brownian Bridges (CFM-BB), or estimating the moments of the probability paths using Gaussian Processes (CFM-GP). We test both models on three simulated data sets, and show that both outperform vanilla neural ODEs, most notably in long trajectories.

Our contributions are as follows: *(i)* we introduce Conditional Flows for Time Series modelling (CFM-TS), a general algorithm for combining information from multiple initial value problems, *(ii)* we explore Gaussian Processes and Brownian Bridges as methods to learn the conditional flows, *(iii)* we present promising empirical results in simulated dynamics, indicating the stability of our model on time-series data with long trajectories compared to a basic neural ODE approach.

[*]Equal contribution [1]Department of Computer Science, Aalto University, Espoo, Finland [2]Yai Yai, Ltd. Correspondence to: Najwa Laabid <najwa.laabid@aalto.fi>, Ella Tamir <ella.tamir@aalto.fi>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

1.1. Related Work

Neural ODEs Recent applications of Neural ODEs include generative modelling and irregular time series analysis (Rubanova et al., 2019). Latent ODE models in particular show promise in handling data sampled at non-uniform intervals (Rubanova et al., 2019), and can be adapted to learn long trajectories via multiple-shooting (Iakovlev et al., 2023).

Conditional Flow Matching This is a training objective for CFM where the goal is to regress a parametric model to match conditional vector fields pre-defined per sample (Lipman et al., 2023). Recent work extends CFM to learning an optimal mapping between arbitrary distributions via Schrödinger bridges (Tong et al., 2024a;b).

Gaussian Processes Gaussian processes (GPs) (Rasmussen & Williams, 2006) have been used in many applications in machine learning and Bayesian statistics (Calderhead et al., 2008; Hedge et al., 2022). New methods focus on efficient mathematical formulations and advances in inference methods (Wilson et al., 2020; 2021).

2. Methods

Consider a trajectory $\mathbf{x}_{1:n} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$, observed at arbitrary time points $t_{1:n} = (t_1, \dots, t_n)$ s.t. $\mathbf{x}_i := \mathbf{x}(t_i) \in \mathbb{R}^d$. In practice, we observe noisy measurements of the state $\mathbf{x}_i$: our data is $\mathcal{D} = \{\mathbf{Y}^k\}_{k=1}^K$ where each $\mathbf{Y}^k$ is a time series observed at the collection of time stamps $\{t_i\}_{i=1}^{N_k}$, possibly different for each trajectory. We would like to infer the dynamics of the system generating $\mathbf{x}_{1:n}$ from the collection of time-series observations in D. These dynamics can be modelled as an ODE

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = \mathbf{u}_t(\mathbf{x}(t)) \tag{1}$$

where $\mathbf{u}_t$ is the vector field describing the instantenous rate of change of the state at time t. Learning the dynamics of the system is equivalent to approximating the vector field $\mathbf{u}_t$. This can be done by regressing a neural network $\mathbf{v}_\theta(\mathbf{x}, t)$ using the MSE loss (with p_t the marginal distribution of $\mathbf{x}(t)$ at time t)

$$\mathcal{L}(\theta) = \mathbb{E}_{t, p_t(\mathbf{x})} \|\mathbf{v}_\theta(\mathbf{x}, t) - \mathbf{u}_t(\mathbf{x})\|^2. \tag{2}$$

Lipman et al. (2023) propose a similar regression objective, dubbed Flow Matching, for training continuous normalizing flows. In our time-series setting, much like in flow-based generative modelling, we have no prior information about suitable (and tractable) $\mathbf{u}_t$ and p_t. We can construct both by combining flows and vector fields defined *per trajectory*. A similar objective is called Conditional Flow Matching in Lipman et al. (2023), where it is used for generative modelling from p_0 to p_T.

2.1. Conditional flows for time series data

In the spirit of Lipman et al. (2023), we marginalize $p_t(\mathbf{x})$ over the observed trajectories $\mathbf{Y}$, setting

$$p_t(\mathbf{x}) = \int p_t(\mathbf{x} \mid \mathbf{Y}) p(\mathbf{Y}) \, \mathrm{d}\mathbf{Y}, \tag{3}$$

where $p(\mathbf{Y})$ is an unknown data distribution. We set the marginal vector field as

$$\mathbf{u}_t(\mathbf{x}) = \int \mathbf{u}_t(\mathbf{x} \mid \mathbf{Y}) \frac{p_t(\mathbf{x} \mid \mathbf{Y}) p(\mathbf{Y})}{p_t(\mathbf{x})} \, \mathrm{d}\mathbf{Y}, \tag{4}$$

where each $\mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})$ is a vector field corresponding to a trajectory $\mathbf{Y}$. We then define a conditional objective,

$$\mathcal{L}(\theta) = \mathbb{E}_{t, p(\mathbf{Y}), p_t(\mathbf{x} \mid \mathbf{Y})} \|\mathbf{v}_\theta(\mathbf{x}, t) - \mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})\|^2 \tag{5}$$

$$\approx \sum_{i=1}^{n} \sum_{k=1}^{K} \sum_{m=1}^{M} \|\mathbf{v}_\theta(\mathbf{x}_m(t_i), t_i) - \mathbf{u}_{t_i}(\mathbf{x}_m(t_i) \mid \mathbf{Y}^k)\|^2. \tag{6}$$

For this framework to hold, we extend Theorems 1 and 2 from Lipman et al. (2023) from conditioning on individual samples to conditioning on observed trajectories.

Theorem 1. *If $\mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})$ generates the conditional densities $p_t(\mathbf{x} \mid \mathbf{Y})$, then $\mathbf{u}_t$ as in Eq. (4) generates the density p_t as in Eq. (3).*

Theorem 2. *For $p_t(\mathbf{x}) > 0$, the loss functions in Eq. (2) and Eq. (5) are equal up to a constant independent of θ.*

The proofs are given in App. A. With these results, we only need to define the conditional probability density path $p_t(\mathbf{x} \mid \mathbf{Y})$ and the corresponding conditional vector field $\mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})$ to train our model.

2.2. Brownian-Bridge Flows with Kalman Smoothing (BB-Flows)

One simple way to define the conditional flows is via a linear interpolation between two consecutive Gaussian distributions (Lipman et al., 2023; Tong et al., 2024a). The matching vector field is also obtained in closed form through a direct application of Equation 15 in (Lipman et al., 2023). We give both the flow and the vector field in Eq. (7) and Eq. (8) respectively.

$$p_t(\mathbf{x} \mid \mathbf{Y}_{0,1}) = \mathcal{N}(\mathbf{m}_0 + \lambda \Delta \mathbf{m}_{1,0} \mathbf{P}_0 + \lambda \Delta \mathbf{P}_{1,0}) \tag{7}$$

$$\mathbf{u}_t(\mathbf{x} \mid \mathbf{Y}_{0,1}) = \frac{\Delta \mathbf{P}_{1,0}}{\Delta t_{1,0}} \frac{(\mathbf{x} - \mathbf{m}_0)}{\mathbf{P}_0} + \frac{\Delta \mathbf{m}_{1,0}}{\Delta t_{1,0}} \tag{8}$$

where $\lambda = \frac{t - t_0}{t_1 - t_0}$, $\mathbf{Y}_{0,1} = \{y_0, y_1\}$ are the two observations surrounding $\mathbf{x}(t)$, i.e. $t \in [t_0, t_1]$, and $\Delta \mathbf{m}_{1,0} = \mathbf{m}_1 - \mathbf{m}_0$ (resp. for t and $\mathbf{P}$). The variables $\mathbf{m}_0$, $\mathbf{m}_1$, $\mathbf{P}_0$, and $\mathbf{P}_1$ represent the means and covariances of the marginals at times t_0 and t_1 respectively. We estimate these marginals using a Kalman Smoother with zero drift, i.e. assuming the differences between the values of $\mathbf{x}_t$ is only due to Gaussian noise. More details on Kalman Smoothing can be found in App. B.

2.3. Gaussian Process Flows (GP-Flows)

Here we model the conditional probability path for a particular trajectory $\mathbf{Y}$ using a Gaussian Process (GP, Rasmussen & Williams, 2006)

$$\mathbf{x}_t \mid \mathbf{Y} \sim \mathcal{GP}(\mu_*, \mathbf{\Sigma}_*), \tag{9}$$

with $\mathbf{x}_t \mid \mathbf{Y}$ the flow pushing samples from $\mathbf{x}(0)$ to $\mathbf{x}(t)$ on trajectory $\mathbf{Y}$, μ_* is the posterior mean function and $\mathbf{\Sigma}_*$ is the posterior covariance matrix at arbitrary time t. Here, we approximate the vector field $\mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})$ with the derivative of the flow w.r.t time. Our flow being a sample from a GP means its derivative is also a GP (Calderhead et al., 2008). Since we would like to stay in the domain of deterministic dynamics, we use as an approximation for $\mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})$ the mean of the GP derivative conditional on a sampled state $\mathbf{x}$, $\dot{\mathbf{m}}_{|\mathbf{x}}$, which has the following analytical form (Calderhead et al., 2008)

$$\mathbf{u}_t(\mathbf{x} \mid \mathbf{Y}) \approx \dot{\mathbf{m}}_{|\mathbf{x}} = \mathbf{\Sigma}_{\dot{\mathbf{x}}\mathbf{x}} \mathbf{\Sigma}_{\mathbf{x}\mathbf{x}}^{-1} \mathbf{x}, \tag{10}$$

where $\mathbf{x}$ is a state sampled from the posterior GP at a new given time point t, $\mathbf{\Sigma}_{\dot{\mathbf{x}},\mathbf{x}|_\mathbf{Y}}$ is the covariance matrix between the derivative and the state, and $\mathbf{\Sigma}_{\mathbf{x},\mathbf{x}|_\mathbf{Y}}$ is the covariance matrix of the state at the observed time points. We thus need a GP model to capture the covariance between a state $\mathbf{x}_t$ and its derivative $\dot{\mathbf{x}}_t$. We define this GP as a joint model of the state and the derivative

$$\mathbf{x}_t, \dot{\mathbf{x}}_t \mid \mathbf{Y} \sim \mathcal{GP}\left([\mu_*, \dot{\mu}_*], \dot{\mathbf{\Sigma}}_*\right) \tag{11}$$

where μ_* and $\dot{\mu}_*$ are the mean of the state and the derivative respectively, and $\dot{\mathbf{\Sigma}}_* = \begin{pmatrix} \mathbf{\Sigma}_{\mathbf{x}\mathbf{x}}, \mathbf{\Sigma}_{\mathbf{x}\dot{\mathbf{x}}} \\ \mathbf{\Sigma}_{\dot{\mathbf{x}}\mathbf{x}}, \mathbf{\Sigma}_{\dot{\mathbf{x}}\dot{\mathbf{x}}} \end{pmatrix}$ is the covariance of the joint state-derivative GP. We define an observation model that only accounts for the state observations

$$\begin{aligned} \mathbf{y}|_{t=t_i} &= \mathbf{H}(\mathbf{x}, \dot{\mathbf{x}})|_{t=t_i} + \epsilon_i & (12)\\ &= (1, 0) \cdot (\mathbf{x}, \dot{\mathbf{x}})|_{t=t_i} + \epsilon_i & (13)\\ &= \mathbf{x}(t_i) + \epsilon_i, & (14) \end{aligned}$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. App. C includes more details on computing the GP posterior mean and covariance, in addition to the log-likelihood of the model and the formula for the RBF kernel.

Our loss with the conditional GP flow becomes

$$\mathcal{L}(\theta) = \mathbb{E}_{t,\mathbf{x},\mathbf{Y}} \|\mathbf{v}_\theta(\mathbf{x}, t) - \mathbf{\Sigma}_{\dot{\mathbf{x}}\mathbf{x}} \mathbf{\Sigma}_{\mathbf{x}\mathbf{x}}^{-1} \mathbf{x}\|^2. \tag{15}$$

Algorithm 1 Preprocess p_t and $\mathbf{u}_t$ using BB-Flows

```
  Input: $n$, $T$, $D = \{\mathbf{Y}^k\}_{k=1}^K$
  $\mathbf{t} = (t_1, t_2, \ldots, t_n)$ where $t_i = \frac{i-1}{n-1}T$ for $i = 1, \ldots, n$
  Initialize lists: Flows ← [], VectorFields ← []
  for $\mathbf{Y}^k \in D$ do
    Get $\mathbf{m_t}$ and $\mathbf{P_t}$ from $\mathbf{Y}^k$ using Kalman Smoothing
    Set $\mathbf{m}_0 = \mathbf{m_t}$ and $\mathbf{m}_1 = (\mathbf{m_t}[1], \ldots, \mathbf{m_t}[n-1])$
    $\mathbf{x(t)} \sim \mathcal{N}(\mathbf{m}_0 + \lambda\Delta\mathbf{m}_{1,0}, \mathbf{P}_0 + \lambda\Delta\mathbf{P}_{1,0})$
    $\mathbf{u_t}(\mathbf{x} \mid \mathbf{Y}^k) = \frac{\Delta\mathbf{P}_{1,0}}{\Delta t_{1,0}} \frac{(\mathbf{x(t)} - \mathbf{m}_0)}{\mathbf{P}_0} + \frac{\Delta\mathbf{m}_{1,0}}{\Delta t_{1,0}}$
    Append $\mathbf{x(t)}$ to Flows
    Append $\mathbf{u_t}(\mathbf{x} \mid \mathbf{Y}^k)$ to VectorFields
  end for
  Return Flows, VectorFields
```

Algorithm 2 Preprocess p_t and $\mathbf{u}_t$ using GP-Flows

```
  Input: $n$, $T$, $D = \{\mathbf{Y}^k\}_{k=1}^K$
  $\mathbf{t} = (t_1, t_2, \ldots, t_n)$ where $t_i = \frac{i-1}{n-1}T$ for $i = 1, \ldots, n$
  Initialize lists: Flows ← [], VectorFields ← []
  for $\mathbf{Y}^k \in D$ do
    Fit the hyperparameters of $\mathcal{GP}$ to $\mathbf{Y}^k$
    Get the posterior $\mathcal{GP}\left([\mu_*, \dot{\mu}_*], \dot{\mathbf{\Sigma}}_*\right)$
    $\mathbf{x(t)}, \dot{\mathbf{x}}\mathbf{(t)} \mid \mathbf{Y}^k \sim \mathcal{GP}\left([\mu_*, \dot{\mu}_*], \dot{\mathbf{\Sigma}}_*\right)$
    $\dot{\mathbf{m}}_{|\mathbf{x}} = \mathbf{\Sigma}_{\dot{\mathbf{x}}\mathbf{x}} \mathbf{\Sigma}_{\mathbf{x}\mathbf{x}}^{-1} \mathbf{x}$
    Append $\mathbf{x(t)}$ to Flows
    Append $\dot{\mathbf{m}}_{|\mathbf{x}}$ to VectorFields
  end for
  Return Flows, VectorFields
```

In practice, for both BB-Flows and GP-Flows, we preprocess the target flows and vector fields for a set of linearly spaced time points $\mathbf{t} = (t_1, t_2, \ldots, t_n)$ s.t. $t_i = \frac{i-1}{n-1}T$ for $i = 1, \ldots, n$, and minibatch over these flows and vector fields during training. The preprocessing for both methods is given in Algorithm 1 and Algorithm 2 respectively.

3. Experiments

We test our method on three simulated systems: 1D pendulum ODE, 1D sine ODE, and a 2D Lotka-Volterra system. The process of simulating each dataset and additional model hyperparameters are described in App. D.

Experimental set-up We generate 300 trajectories for the Lotka-Volterra system, 100 for the sine ODE, and only 1 trajectory for the pendulum. All models are trained for $400 - 600$ epochs with a learning rate of 0.01 and a batch size of 258. For evaluation, we use an 80%/20% train/test ratio on the number of trajectories when many are available, and generate observations at new locations within the same trajectory interval for the single pendulum trajectory. We visualize the trajectories generated by the learned vector

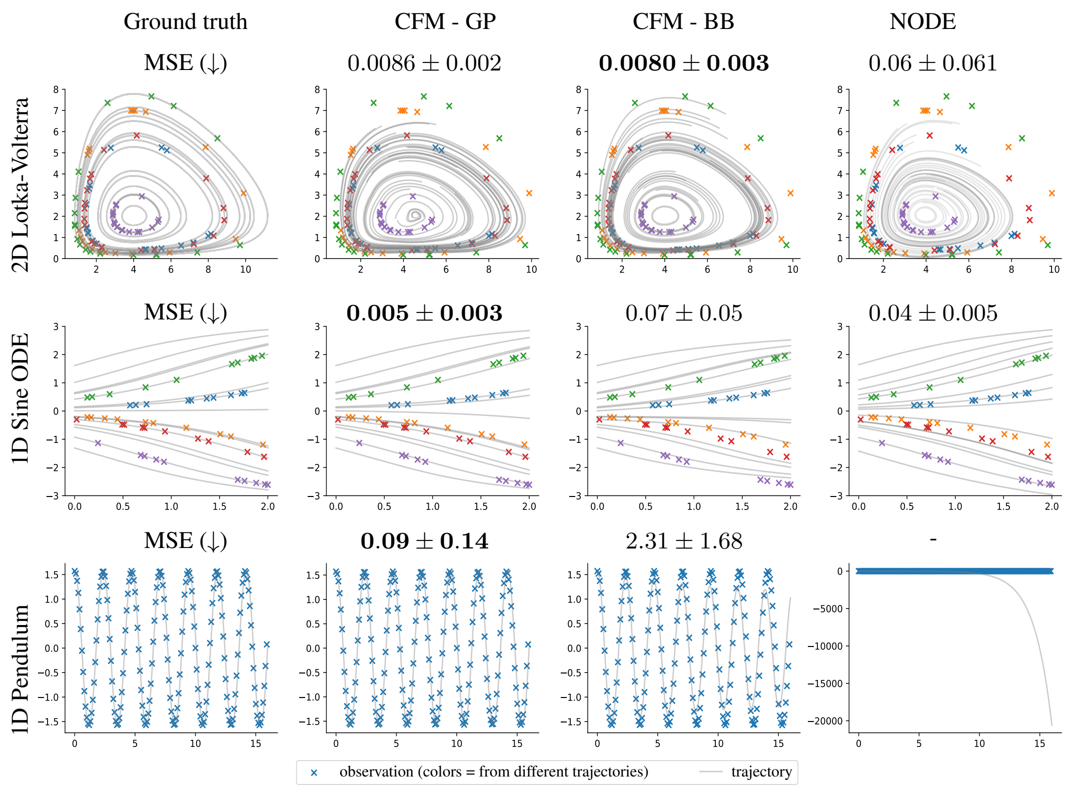

Figure 2. Comparing the true and learned trajectories for all models across all three tasks. The colored markers represent the training observations for each trajectory. NODE fails completely on 1D Pendulum and performs poorly on the Lotka-Volterra, while CFM-BB struggles with 1D Pendulum and performs best on the Lotka-Volterra. CFM-GP performs well across all tasks.

Table 1. The MSE loss(↓) (mean and standard deviation over 5 runs) for all models. **L-V** is Lotka-Volterra. We do not report the pendulum results for neural ODEs, as the loss did not change during training.

	L-V	Sine	Pendulum
NODE	0.06±0.061	0.04±0.005	—
CFM-BB	**0.0080±0.003**	0.07±0.05	2.31±1.68
CFM-GP	0.0086 ± 0.002	**0.005±0.003**	**0.09±0.14**

field compared to the test trajectories in Fig. 2. In Table 1, we report the mean and standard deviation of the MSE between the true and predicted trajectories over 5 runs for each model-dataset pair.

Results On the Lotka-Volterra system, both CFM-BB and CFM-GP outperform NODE, with CFM-BB achieving the lowest MSE. More elaborate experiments are required to elucidate the extent of our model's superiority and the reason behind it. Fig. 2 also shows that the trajectories reconstructed by CFM-BB look closest to the test trajectories. CFM-GP outperforms CFM-BB on Pendulum and Sine. We believe this is due to the GP kernel having more flexibility for handling sine-like functions compared to the linear-interpolation of CFM-BB. A basic NODE peforms worst at all tasks, particularly on the Pendulum system, where the loss does not change at all. Non-convergence of NODE on the full pendulum trajectory was also reported by Iakovlev et al. (2023) (Appendix A), and is due to the complexity of the loss landscape increasing with the length of the trajectory (see Figure 15 in (Iakovlev et al., 2023)).

4. Discussion and Conclusions

We present Conditional Flow Matching for Time Series (CFM-TS), a general method for learning a neural ODE by linearly combining the dynamics of multiple irregularly sampled time series. We explore two approaches for fitting the individual time series: BB-Flows and GP-Flows. We show the superiority of boths methods over a simple neural ODE, particularly on long trajectories, with GP-Flows excelling in particular at sine-like data trajectories. In future work, we plan to: 1) improve GP-Flows with a more suitable and efficient sampling (Wilson et al., 2020), and 2) combine our method with encoder-decoder architectures in order to model latent dynamics and compare to more advanced baselines (Iakovlev et al., 2023; Yildiz et al., 2019).

References

Calderhead, B., Girolami, M., and Lawrence, N. Accelerating bayesian inference over nonlinear differential equations with gaussian processes. In *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems 31*, pp. 6571–6583. Curran Associates, Inc., 2018a.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018b.

Grathwohl, W., Chen, R. T. Q., Bettencourt, J., Sutskever, I., and Duvenaud, D. FFJORD: free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations, ICLR*, 2019.

Hedge, P., Yildiz, C., Lähdesmäki, H., Kaski, S., and Heinonen, M. Variational multiple shooting for bayesian odes with gaussian processes. In *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2022)*, Proceedings of Machine Learning Research, pp. 790–799. JMLR, 2022.

Iakovlev, V., Yildiz, C., Heinonen, M., and Lähdesmäki, H. Latent neural ODEs with sparse bayesian multiple shooting. In *The Eleventh International Conference on Learning Representations*, 2023.

Lipman, Y., Chen, R. T. Q., Ben-Hamu, H., Nickel, M., and Le, M. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023.

Metz, L., Freeman, C. D., Schoenholz, S. S., and Kachman, T. Gradients are not all you need, 2022.

Rasmussen, C. E. and Williams, C. K. I. *Gaussian processes for machine learning.* Adaptive computation and machine learning. MIT Press, 2006.

Ribeiro, A. H., Tiels, K., Umenberger, J., Schön, T. B., and Aguirre, L. A. On the smoothness of nonlinear system identification. *Automatica*, 121, 2020. doi: https://doi.org/10.1016/j.automatica.2020.109158.

Rubanova, Y., Chen, R. T. Q., and Duvenaud, D. K. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Särkkä, S. and Svensson, L. *Bayesian Filtering and Smoothing*. Cambridge University Press, Cambridge, UK, second edition, 2013. ISBN 978-1-107-02375-6.

Tong, A., FATRAS, K., Malkin, N., Huguet, G., Zhang, Y., Rector-Brooks, J., Wolf, G., and Bengio, Y. Improving and generalizing flow-based generative models with minibatch optimal transport. *Transactions on Machine Learning Research*, 2024a.

Tong, A., Malkin, N., Fatras, K., Atanackovic, L., Zhang, Y., Huguet, G., Wolf, G., and Bengio, Y. Simulation-free schrödinger bridges via score and flow matching, 2024b.

Wilson, J., Borovitskiy, V., Terenin, A., Mostowsky, P., and Deisenroth, M. Efficiently sampling functions from Gaussian process posteriors. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 10292–10302. PMLR, 2020.

Wilson, J. T., Borovitskiy, V., Terenin, A., Mostowsky, P., and Deisenroth, M. P. Pathwise conditioning of gaussian processes. 22(1), 2021.

Yildiz, C., Heinonen, M., and Lahdesmaki, H. Ode2vae: Deep generative second order odes with bayesian neural networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

A. Proofs of theoretical results

Here we extend the theoretical analysis presented in Lipman et al. (2023) in order to be able to marginalize over conditional probability paths dependent on time-series data.

Theorem 1. *If $\mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})$ generates the conditional densities $p_t(\mathbf{x} \mid \mathbf{Y})$, then $\mathbf{u}_t$ as in Eq. (4) generates the density p_t as in Eq. (3).*

For this result, it is sufficient to show that $\mathbf{u}_t(\mathbf{x})$ and $p_t(\mathbf{x})$ satisfy the continuity equation,

$$\frac{d}{dt} \log p_t(\mathbf{x}) + div(\mathbf{u}_t(\mathbf{x}) p_t(\mathbf{x})) = 0. \tag{16}$$

Following the derivation in (Lipman et al., 2023), we get

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t} p_t(\mathbf{x}) &= \int \big(\frac{\mathrm{d}}{\mathrm{d}t} p_t(\mathbf{x} \mid \mathbf{Y})\big) p(\mathbf{Y})\, \mathrm{d}\mathbf{Y} \\
&= -\int \mathrm{div}\big(\mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})\, p_t(\mathbf{x} \mid \mathbf{Y})\big) p(\mathbf{Y})\, \mathrm{d}\mathbf{Y} \\
&= -\mathrm{div}\left(\int \mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})\, p_t(\mathbf{x} \mid \mathbf{Y}) p(\mathbf{Y})\, \mathrm{d}\mathbf{Y}\right) \\
&= -\mathrm{div}\left(\mathbf{u}_t(\mathbf{x})\, p_t(\mathbf{x})\right),
\end{aligned}$$

where the first equality is a result of Eq. (3), the second equality comes from the fact that $\mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})$ and $p_t(\mathbf{x} \mid \mathbf{Y})$ satisfy the continuity equation, and the last equality comes from the definition of the conditional vector field in Eq. (4).

Theorem 2. *Assume that $p_t(\mathbf{x} \mid \mathbf{Y}) > 0$, then the loss functions in Eq. (2) and Eq. (5) are equal up to a constant independent of θ.*

For this result, we again follow (Lipman et al., 2023) with minor modifications. Notice that we may decompose the square losses within Eq. (2) and Eq. (5) respectively as

$$\begin{aligned}
&\|\mathbf{v}_\theta(\mathbf{x}, t) - \mathbf{u}_t(\mathbf{x})\|^2 = \|\mathbf{v}_\theta(\mathbf{x}, t)\|^2 - 2\langle \mathbf{v}_\theta(\mathbf{x}, t), \mathbf{u}_t(\mathbf{x})\rangle + \|\mathbf{u}_t(\mathbf{x})\|^2, \\
&\|\mathbf{v}_\theta(\mathbf{x}, t) - \mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})\|^2 = \|\mathbf{v}_\theta(\mathbf{x}, t)\|^2 - 2\langle \mathbf{v}_\theta(\mathbf{x}, t), \mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})\rangle + \|\mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})\|^2.
\end{aligned}$$

As the terms $\|\mathbf{u}_t(\mathbf{x})\|^2$ and $\|\mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})\|$ are independent of the neural network parameters θ, they are irrelevant for evaluating $\nabla_\theta \mathcal{L}_{CFM}$. For the first term, by the definition of p_t in Eq. (3),

$$\begin{aligned}
\mathbb{E}_{\mathbf{x} \sim p_t(\mathbf{x})} \|\mathbf{v}_\theta(\mathbf{x}, t)\|^2 &= \int \|\mathbf{v}_\theta(\mathbf{x}, t)\|^2 p_t(\mathbf{x})\, \mathrm{d}\mathbf{x} \\
&= \int\int \|\mathbf{v}_\theta(\mathbf{x}, t)\|^2 p_t(\mathbf{x} \mid \mathbf{Y}) p(\mathbf{Y})\, \mathrm{d}\mathbf{Y}\, \mathrm{d}\mathbf{x} \\
&= \mathbb{E}_{t, p(\mathbf{Y}), p(\mathbf{x} \mid \mathbf{Y})} \|\mathbf{v}_\theta(\mathbf{x}, t)\|^2.
\end{aligned}$$

For the inner product term, again applying definitions of $\mathbf{u}_t$ and p_t,

$$\begin{aligned}
\mathbb{E}_{\mathbf{x} \sim p_t(\mathbf{x})} \langle \mathbf{v}_\theta(\mathbf{x}, t), \mathbf{u}_t(\mathbf{x})\rangle &= \int \langle \mathbf{v}_\theta(\mathbf{x}, t), \mathbf{u}_t(\mathbf{x})\rangle p_t(\mathbf{x})\, \mathrm{d}\mathbf{x} \\
&= \int \langle \mathbf{v}_\theta(\mathbf{x}, t), \int \frac{\mathbf{u}_t(\mathbf{x} \mid \mathbf{Y}) p(\mathbf{Y}) p_t(\mathbf{x} \mid \mathbf{Y})}{p_t(\mathbf{x})} d\mathbf{Y}\rangle p_t(\mathbf{x})\, \mathrm{d}\mathbf{x} \\
&= \int \langle \mathbf{v}_\theta(\mathbf{x}, t), \int \mathbf{u}_t(\mathbf{x} \mid \mathbf{Y}) p(\mathbf{Y}) p_t(\mathbf{x} \mid \mathbf{Y})\rangle\, \mathrm{d}\mathbf{x} \\
&= \int\int \langle \mathbf{v}_\theta(\mathbf{x}, t), \mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})\rangle p(\mathbf{Y}) p_t(\mathbf{x} \mid \mathbf{Y})\, \mathrm{d}\mathbf{x} \\
&= \mathbb{E}_{t, p(\mathbf{Y}), p_t(\mathbf{x} \mid \mathbf{Y})} \langle \mathbf{v}_\theta(\mathbf{x}, t), \mathbf{u}_t(\mathbf{x} \mid \mathbf{Y})\rangle.
\end{aligned}$$

The two losses in Eq. (5) and Eq. (2) are thus equivalent to within a constant.

B. Kalman Smoothing

Kalman smoothing is an algorithm to estimate the states of a linear dynamic system under Gaussian noise (Särkkä & Svensson, 2013). We consider a model with zero drift, which implies that the state of the system at any time point is expected to be the same as the previous time point unless influenced by noise. There are two steps in the Kalman smoothing algorithm: filtering and smoothing.

Filtering The state of the system is estimated at a new time point $t + 1$ given the state at time point t. Filtering is divided to two sub-steps: prediction and update. The prediction step estimates the state of the system at the next time point,

$$\mathbf{m}_{t+1} = \mathbf{A}\mathbf{m}_t, \tag{17}$$

$$\mathbf{P}_{t+1} = \mathbf{A}\mathbf{P}_t\mathbf{A}^\top + \mathbf{Q} \tag{18}$$

where $\mathbf{A}$ is the transition matrix (also known as the drift) and $\mathbf{m}_t$ is the state of the system at time t. Zero drift in this context implies that $\mathbf{A} = I$, and $\mathbf{Q}$ is the covariance matrix of the process noise. If observations are available at time t, they can be used to correct the prediction during the update step,

$$\mathbf{K}_t = \mathbf{P}_t\mathbf{H}^\top(\mathbf{H}\mathbf{P}_t\mathbf{H}^\top + \mathbf{R})^{-1}, \tag{19}$$

$$\mathbf{m}^f_{t+1} = \mathbf{m}_t + \mathbf{K}_t(\mathbf{Y}_{t+1} - \mathbf{H}\mathbf{m}_t), \tag{20}$$

$$\mathbf{P}^f_{t+1} = (\mathbf{I} - \mathbf{K}_t\mathbf{H})\mathbf{P}_t, \tag{21}$$

where $\mathbf{H}$ is the observation matrix, $\mathbf{Y}_{t+1}$ is the observation at time $t + 1$, and $\mathbf{R}$ is the observation noise covariance matrix.

Smoothing The state of the system is estimated at each time point given all the observations. This is done by updating the state estimates from the filtering step starting from the last time point, so in this case we update the state and covariance at time t using the estimates at time $t + 1$. The smoothing equation is given by

$$\mathbf{m}^s_t = \mathbf{m}^f_t + \mathbf{G}_t(\mathbf{m}^s_{t+1} - \mathbf{m}^f_{t+1}), \tag{22}$$

$$\mathbf{P}^s_t = \mathbf{P}^f_t + \mathbf{G}_t(\mathbf{P}^s_{t+1} - \mathbf{P}^f_{t+1})\mathbf{K}^\top_t. \tag{23}$$

where $\mathbf{G}_t = \mathbf{P}^f_t\mathbf{A}\mathbf{P}^f_{t+1}$ is the smoothing gain at time t. By applying this algorithm to the time series data, we can estimate the marginal distribution of the state of the system at each time point t which is assumed to be Gaussian: $\mathcal{N}(\mathbf{m}_t, \mathbf{P}_t)$. Given these estimates, we can define our flow as a linear interpolation between consecutive time points (Eq. (7)) with a corresponding vector field (Eq. (8)).

C. Gaussian Processes

Here we include general formulas for the basic concepts relevant to a Gaussian Process with an RBF kernel: 1) the posterior of a Gaussian Process (GP) given a set of observations, 2) the likelihood used to optimize the hyperparameters of the kernel through MLE, and 3) the RBF kernel. For simplicity we include the equations for a GP modelling the state function alone. Equations for the GP modelling both the state and its derivative can be obtained by assuming $\mathbf{x}_t = (\mathbf{x}_t, \dot{\mathbf{x}}_t)$, $\mathbf{m} = (\mathbf{m}, \dot{\mathbf{m}})$, and $\mathbf{K} = \begin{bmatrix} \mathbf{K}_{\mathbf{x}\mathbf{x}} & \mathbf{K}_{\mathbf{x}\dot{\mathbf{x}}} \\ \mathbf{K}_{\dot{\mathbf{x}}\mathbf{x}} & \mathbf{K}_{\dot{\mathbf{x}}\dot{\mathbf{x}}} \end{bmatrix}$.

We start with a Gaussian Process prior

$$\mathbf{x}_t \sim \mathcal{GP}(\mathbf{m}(t), \mathbf{K}) \tag{24}$$

where $\mathbf{m}$ is the mean function and K is the covariance matrix with hyperparameters ω. Assuming observations $\mathbf{y}$ at training points $\mathbf{t}$ with noise $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, the posterior distribution of $\mathbf{x}_t$ at new test points $\mathbf{t}_*$ is given by:

$$\mathbf{x}_{\mathbf{t}_*}|\mathbf{t}, \mathbf{y}, \mathbf{t}_* \sim \mathcal{N}(\mu_*, \mathbf{\Sigma}_*) \tag{25}$$

where the mean μ_* and covariance $\mathbf{\Sigma}_*$ are computed as follows:

$$\mu_* = \mathbf{m}(\mathbf{t}_*) + \mathbf{K}^\top_*\mathbf{K}^{-1}(\mathbf{y} - \mathbf{m}(\mathbf{t})) \tag{26}$$

$$\mathbf{\Sigma}_* = \mathbf{K}_{\mathbf{t}_*\mathbf{t}_*} - \mathbf{K}^\top_{\mathbf{t}_*\mathbf{t}}\mathbf{K}^{-1}_{\mathbf{t}\mathbf{t}}\mathbf{K}_{\mathbf{t}_*\mathbf{t}} \tag{27}$$

Our covariance matrix $\mathbf{K}$ is defined using an RBF kernel, which is given below for completness

$$k(t, t') = \sigma^2 \exp\left(-\frac{1}{2l^2}(t - t')^2\right), \tag{28}$$

where σ^2 is the variance and l is the lengthscale, which can also be written as $\theta = (\sigma^2, l)$.

The likelihood function is given by

$$\log p(\mathbf{y} \mid \theta) = -\frac{1}{2}\mathbf{y}^\top \mathbf{K}^{-1}\mathbf{y} - \frac{1}{2}\log |\mathbf{K}| - \frac{n}{2}\log 2\pi. \tag{29}$$

Using maximum-likelihood estimation, we can learn the parameters θ which best fit the data $\mathbf{y}$.

D. Experimental details

We provide below details of the datasets generation procedure and the hyperparameters used by our models.

Lotka–Volterra The Lotka–Volterra system for two-dimensional inputs is defined by the following ODEs

$$\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2), \tag{30}$$

$$\mathrm{d}\mathbf{x}_1 = 2\mathbf{x}_1 - \mathbf{x}_1\mathbf{x}_2, \quad \mathrm{d}\mathbf{x}_2 = -4\mathbf{x}_2 + \mathbf{x}_1\mathbf{x}_2. \tag{31}$$

We sample initial values from the uniform distribution over the square $[1, 7] \times [1, 7]$, and evolve 400 trajectories. The trajectories are run until $T = 5$, by solving the ODE at 500 random points within $(0, 5)$. We collect observations from each trajectory at 50 randomly sampled points (different for each trajectory).

1D Sine ODE We consider the ODE system initialized at the distribution $p_0 = \mathcal{N}(0, 1)$, evolving according to

$$\mathrm{d}\mathbf{x}_t = \sin(\mathbf{x}_t)dt. \tag{32}$$

We observe 400 trajectories from the system at 50 randomly sampled points in $t \in (0, 2)$.

1D Pendulum ODE We model the position and velocity of a pendulum, where $\mathbf{x} \in \mathbb{R}^2$ represents the angle and angular velocity. The initial angle is set to 90 degrees with velocity 0.

$$\frac{\mathrm{d}^2\mathbf{x}(t)}{\mathrm{d}t^2} = -9.81 \sin(\mathbf{x}(t)) \tag{33}$$

We only generate a single trajectory using this ODE. We observe the position every 0.1 seconds for $t \in (0, 16)$ (i.e. we use $n = 160$ observations).

NN hyperparameters For all experiments, we use a neural network with an input layer of 64 dimensions, and 3 hidden layers each with 258 dimensions. We use `LeakyRelu` as the activation functions and `Adam` as the optimizer. We train the CFM models for 400 to 600 epochs and the NODE model for 10 epochs. We did not notice improvements for longer training of NODE.

Training resources All models are trained on a Mac M2 chip with a batch size of 256 and a learning rate of 0.01. The Lotka-Volterra with the GP-Flows model takes approximately 30 minutes to train. The BB-Flows model takes at most 15 minutes for all datasets, while the NODE model takes at most 20 minutes.

Hyperparameters for the GP-Flows conditional model The GP we use for all datasets has an RBF kernel with hyperparameters learned via MLE, after about 400 epochs of training per trajectory. It takes less than a minute to fit the kernel for each trajectory, and this is done only once per dataset. We sample 10 functions from every trajectory GP and compute the conditional derivative mean corresponding to every function.

Hyperparameters for the BB-Flows conditional model We set the process noise to 0.005 and the observation noise to 0.001 for the Kalman Smoother.

Hyperparamers for the basic neural ODE model We use the `dopri5` solver for integrating the learned ODE. We compare the obtained trajectories to the training trajectories using MSE.