1. Sensitivity Equations

In the main text, the sensitivity equation is formulated using matrix notation

\[ \dot{S}(t) = J(t)S(t) + R(t). \] (1)

Here, the time-dependent matrices are obtained by differentiating the vector valued functions with respect to vectors i.e.

\[ S(t) = \begin{bmatrix} \frac{dx_1(t,U)}{du_1} & \frac{dx_1(t,U)}{du_2} & \ldots & \frac{dx_1(t,U)}{du_{MD}} \\ \frac{dx_2(t,U)}{du_1} & \frac{dx_2(t,U)}{du_2} & \ldots & \frac{dx_2(t,U)}{du_{MD}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{dx_D(t,U)}{du_1} & \frac{dx_D(t,U)}{du_2} & \ldots & \frac{dx_D(t,U)}{du_{MD}} \end{bmatrix}^{D \times MD} \] (2)

\[ J(t) = \begin{bmatrix} \frac{\partial f(x_1(t,U))}{\partial x_1} & \frac{\partial f(x_1(t,U))}{\partial x_2} & \ldots & \frac{\partial f(x_1(t,U))}{\partial x_D} \\ \frac{\partial f(x_2(t,U))}{\partial x_1} & \frac{\partial f(x_2(t,U))}{\partial x_2} & \ldots & \frac{\partial f(x_2(t,U))}{\partial x_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(x_D(t,U))}{\partial x_1} & \frac{\partial f(x_D(t,U))}{\partial x_2} & \ldots & \frac{\partial f(x_D(t,U))}{\partial x_D} \end{bmatrix}^{D \times D} \] (3)

\[ R(t) = \begin{bmatrix} \frac{\partial f(x_1(t,U))}{\partial u_1} & \frac{\partial f(x_1(t,U))}{\partial u_2} & \ldots & \frac{\partial f(x_1(t,U))}{\partial u_{MD}} \\ \frac{\partial f(x_2(t,U))}{\partial u_1} & \frac{\partial f(x_2(t,U))}{\partial u_2} & \ldots & \frac{\partial f(x_2(t,U))}{\partial u_{MD}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(x_D(t,U))}{\partial u_1} & \frac{\partial f(x_D(t,U))}{\partial u_2} & \ldots & \frac{\partial f(x_D(t,U))}{\partial u_{MD}} \end{bmatrix}^{D \times MD} \] (4)

2. Optimization

Below is the explicit form of the log posterior. Note that we introduce \( u = \text{vec}(U) \) and \( \Omega = \text{diag}(\omega_1^2, \ldots, \omega_D^2) \) for notational simplicity.
Learning unknown nonparametric ODE’s

\[ \log L = \log p(U|\theta) + \log p(Y|x_0, U, \omega) \]
\[ = \log \mathcal{N}(u|0, K_\theta(Z, Z)) + \sum_{i=1}^{N} \log \mathcal{N}(y_i|x(t_i, U), \Omega) \]
\[ = -\frac{1}{2} u^T K_\theta(Z, Z)^{-1} u - \frac{1}{2} \log |K_\theta(Z, Z)| - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{D} \frac{(y_{ij} - x_j(t_i, U, x_0))^2}{\omega_j^2} - \frac{1}{2} \sum_{i=1}^{N} \log |\Omega| \]
\[ = -\frac{1}{2} u^T K_\theta(Z, Z)^{-1} u - \frac{1}{2} \log |K_\theta(Z, Z)| - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{D} \frac{(y_{ij} - x_j(t_i, U, x_0))^2}{\omega_j^2} - N \sum_{j=1}^{D} \log \omega_j \]

\[ \frac{\partial \log L}{\partial u_k} = \sum_{i=1}^{N} \sum_{j=1}^{D} \frac{y_{ij} - x_j(t_i, U, x_0)}{\omega_j^2} \frac{\partial x_j(t_i, U, x_0)}{\partial u_k} - K_\theta(Z, Z)^{-1} u \]
\[ \frac{\partial \log L}{\partial (x_0)_d} = \sum_{i=1}^{N} \sum_{j=1}^{D} \frac{y_{ij} - x_j(t_i, U, x_0)}{\omega_j^2} \frac{\partial x_j(t_i, U, x_0)}{\partial (x_0)_d} \]
\[ \frac{\partial \log L}{\partial \omega_j} = \frac{1}{\omega_j} \sum_{i=1}^{N} (y_{ij} - x_j(t_i, U, x_0))^2 - \frac{N}{\omega_j} \]

Seemingly hard to compute terms, \( \frac{\partial x_j(t_i, U, x_0)}{\partial u_k} \) and \( \frac{\partial x_j(t_i, U, x_0)}{\partial (x_0)_d} \), are computed using sensitivities. The lengthscale parameter \( \ell \) is considered as a model complexity parameter and is chosen from a grid using cross-validation. We furthermore need the gradient with respect to the other kernel variable, i.e., the signal variance \( \sigma_f^2 \). Because \( K_\theta(Z, Z) \) and \( x(t_i, U) \) are the functions of kernel, computing the gradients with respect to \( \sigma_f^2 \) is not trivial and we make use of partial finite differences:

\[ \frac{\partial \log L}{\partial \sigma_f} = \frac{\log L(\sigma_f + \delta) - \log L(\sigma_f)}{\delta} \]

We use \( \delta = 10^{-4} \) to compute the finite differences.

One problem of using gradient-based optimization techniques is that they do not ensure the positivity of the parameters being optimized. Therefore, we perform the optimization of the noise standard deviations \( \omega = (\omega_1, \ldots, \omega_D) \) and signal variance \( \sigma_f \) with respect to their logarithms:

\[ \frac{\partial \log L}{\partial \log c} = \frac{\partial \log L}{\partial c} \frac{\partial c}{\partial \log c} = \frac{\partial \log L}{\partial c} c \]

where \( c \in (\sigma_f, \omega) \). The training algorithm is given in Algorithm 1.

3. Implementation Details

We initialise the inducing vectors \( U = (u_1, \ldots, u_M) \) by computing the empirical gradients \( \dot{y}_i = y_i - y_{i-1} \), and conditioning as

\[ U_0 = K(Z, Y)K(Y, Y)^{-1} \dot{y}, \]

where we optimize the scale \( c \) against the posterior. The whitened inducing vector is obtained as \( \tilde{U}_0 = L_\theta^{-1} U_0 \). This procedure produces initial vector fields that partially match the trajectory already. We then do 100 restarts of the optimization from random perturbations \( \tilde{U} = \tilde{U}_0 + \varepsilon \).
Algorithm 1: npODE training algorithm

1. Initialize $U, Z, x_0, \theta, \omega$
2. Compute $K_\theta(Z, Z)$ and $L_\theta$
3. while not converged do
   4. Integrate the system to compute the path $x(t)$ and the sensitivities $S(t)$
   5. Compute the gradients by (9), (10), (11), (12) and (13),
   6. Apply the noncenteral parameterization trick: $\nabla_U \log \mathcal{L} = L^T_\theta \nabla_U \log \mathcal{L}$
   7. Update $x_0, \bar{U}, \sigma_f, \omega$ based on L-BFGS update rule
   8. Set $U = L_\theta \bar{U}$

We use L-BFGS gradient optimization routine in Matlab. We initialise the inducing vector locations $Z$ on a equidistant fixed grid on a box containing the observed points. We select the lengthscales $\ell_1, \ldots, \ell_D$ using cross-validation from values $\{0.5, 0.75, 1, 1.25, 1.5\}$. In general large lengthscales induce smoother models, while lower lengthscales cause overfitting.