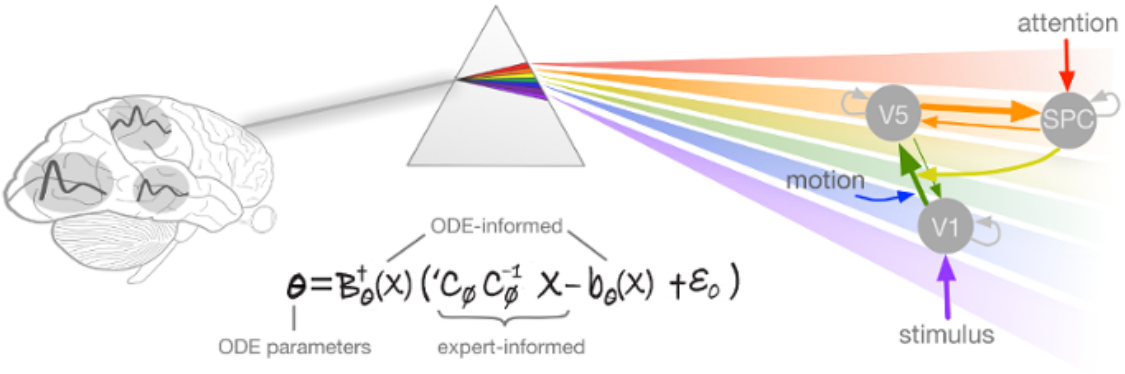
Variational Gradient Matching for Dynamical Systems



Authors: **Nico Stephan Gorbach** and **Stefan Bauer**, email: nico.gorbach@gmail.com

Contents:
Instructional code for the NIPS (2018) paper **Scalable Variational Inference for Dynamical Systems** by Nico S. Gorbach, Stefan Bauer and Joachim M. Buhmann. The paper is available at https://papers.nips.cc/paper/7066-scalable-variational-inference-for-dynamical-systems.pdf. Please cite our paper if you use our program for a further publication. Part of the derivation below is described in Wenk et al. (2018)

Contents

Chapter 1

Introduction

Instructional code for the NIPS (2018) paper " **Scalable Variational Inference for Dynamical Systems** " by Nico S. Gorbach, Stefan Bauer and Joachim M. Buhmann. The paper is available at https://papers.nips.cc/paper/7066-scalable-variational-inference-for-dynamical-systems.pdf. Please cite our paper if you use our program for a further publication. Part of the derivation below is described in Wenk et al. (2018).

1.1 Advantages of Variational Gradient Matching

The essential idea of gradient matching (Calderhead et al., 2002) is to match the gradient governed by the ODEs with that inferred from the observations. In contrast to previous approaches gradient matching introduces a prior over states instead of a prior over ODE parameters. The advantages of gradients matching is two-fold:

1. A prior over the functional form of state dynamics as opposed to ODE parameters facilitates a more expert-aware estimation of ODE parameters since experts can provide a better *a priori* description of state dynamics than ODE parameters.

2. Gradient matching yields a global gradient as opposed to a local one which offers significant computational advantages and provides access to a rich source of sophisticated optimization tools.

Chapter 2

VGM for Lotka-Volterra

Example dynamical system used in this code: Lotka-Volterra system with **half** of the time points **unobserved**.
The ODE parameters are also unobserved.

2.1 Simulation Settings

```
simulation.state_obs_variance = @(mean)(bsxfun(@times,[0.5^2,0.5^2],...
    ones(size(mean))));                                          % observation noise
simulation.ode_param = [2,1,4,1];                               % true ODE parameters [2 1 4 1] is us
simulation.final_time = 2;                                     % end time for integration
simulation.int_interval = 0.01;                                   % integration interval
simulation.time_samp = 0:0.1:simulation.final_time;                  % sample times for observatio
simulation.init_val = [5 3];                                  % state values at first time point
simulation.state_obs_idx = [1,1];                               % indices of states that are directl
```

2.2 User Input

Kernel parameters ϕ:

```
kernel.param = [10,0.2];                                  % set values of rbf kernel parameters
```

Error variance on state derivatives (i.e. γ):

```
state.derivative_variance = [6,6];                           % gamma for gradient matching model
```

```
time.est = 0:0.1:4;                                            % estimation times
```

```
opt_settings.pseudo_inv_type = 'Moore-Penrose';                  % Type of pseudo inverse; optio
opt_settings.coord_ascent_numb_iter = 200;                     % number of coordinate ascent ite
opt_settings.clamp_obs_state_to_GP_fit = false;                  % The observed state trajectori
```

States **x**:

```
symbols.state = {'[prey]','[predator]'};                          % symbols of states in 'ODEs.txt'
```

ODE parameters θ:

```
symbols.param = {'[\theta_1]','[\theta_2]','[\theta_3]','[\theta_4]'};    % symbols of parameters
```

2.3 Import ODEs

```
ode = import_odes(symbols);

disp('ODEs:'); disp(ode.raw)

ODEs:
    '[\theta_1].*[prey] - [\theta_2].*[prey].*[predator]'
    '-[\theta_3].*[predator] + [\theta_4].*[prey].*[predator]'
```

2.4 Mass Action Dynamical Systems

A deterministic dynamical system is represented by a set of K ordinary differential equations (ODEs) with model parameters $\theta \in R^d$ that describe the evolution of K states $\mathbf{x}(t) = [x_1(t), \ldots, x_K(t)]^T$ such that:

$\dot{\mathbf{x}}(t) = \frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(\mathbf{x}(t), \theta) \qquad (1).$

A sequence of observations, $\mathbf{y}(t)$, is usually contaminated by measurement error which we assume to be normally distributed with zero mean and variance for each of the K states, i.e. $\mathbf{E} \sim \mathcal{N}(\mathbf{E}; \mathbf{0}, \mathbf{D})$, with $\mathbf{D}_{ik} = \sigma_k^2 \delta_{ik}$. For N distinct time points the overall system may therefore be summarized as:

$\mathbf{Y} = \mathbf{X} + \mathbf{E}$,

where

$\mathbf{X} = [\mathbf{x}(t_1), \ldots, \mathbf{x}(t_N)] = [\mathbf{x}_1, \ldots, \mathbf{x}_K]^T$,

$\mathbf{Y} = [\mathbf{y}(t_1), \ldots, \mathbf{y}(t_N)] = [\mathbf{y}_1, \ldots, \mathbf{y}_K]^T$,

and $\mathbf{x}_k = [x_k(t_1), \ldots, x_k(t_N)]^T$ is the k'th state sequence and $\mathbf{y}_k = [y_k(t_1), \ldots, y_k(t_N)]^T$ are the observations. Given the observations $\mathbf{Y}$ and the description of the dynamical system (1), the aim is to estimate both state variables $\mathbf{X}$ and parameters θ.

We consider only dynamical systems that are locally linear with respect to ODE parameters θ and individual states $\mathbf{x}_u$. Such ODEs include mass-action kinetics and are given by:

$f_k(\mathbf{x}(t), \theta) = \sum_{i=1} \theta_{ki} \prod_{j \in \mathcal{M}_{ki}} x_j \qquad (2),$

with $\mathcal{M}_{ki} \subseteq \{1, \ldots, K\}$ describing the state variables in each factor of the equation (i.e. the functions are linear in parameters and contain arbitrary large products of monomials of the states).

2.5 Simulate Trajectory Observations

```
[state,time,ode] = generate_ground_truth(time,state,ode,symbols,simulation);
```

```
[state,time,obs_to_state_relation] = generate_state_obs(state,time,simulation);


state.sym.mean = sym('x%d%d',[length(time.est),length(ode.system)]);
state.sym.variance = sym('sigma%d%d',[length(time.est),length(ode.system)]);
ode_param.sym.mean = sym('param%d',[length(symbols.param),1]); assume(ode_param.sym.mean,'real'
```

Only the state dynamics are (partially) observed.

```
[h_states,h_param,p] = setup_plots(state,time,simulation,symbols);

tic; %start timer
```

2.6 Prior on States and State Derivatives

Gradient matching with Gaussian processes assumes a joint Gaussian process prior on states and their derivatives:

$$\begin{pmatrix} \mathbf{X} \\ \dot{\mathbf{X}} \end{pmatrix} \sim \mathcal{N}\left(\begin{matrix} \mathbf{X} \\ \dot{\mathbf{X}} \end{matrix} ; \begin{matrix} \mathbf{0} \\ \mathbf{0} \end{matrix}, \begin{matrix} \mathbf{C}_\phi & \mathbf{C}'_\phi \\ {}'\mathbf{C}_\phi & \mathbf{C}''_\phi \end{matrix} \right) \qquad (3),$$

$\text{cov}(x_k(t), x_k(t)) = C_{\phi_k}(t, t')$

$\text{cov}(\dot{x}_k(t), x_k(t)) = \frac{\partial C_{\phi_k}(t,t')}{\partial t} =: C'_{\phi_k}(t, t')$

$\text{cov}(x_k(t), \dot{x}_k(t)) = \frac{\partial C_{\phi_k}(t,t')}{\partial t'} =: {}'C_{\phi_k}(t, t')$

$\text{cov}(\dot{x}_k(t), \dot{x}_k(t)) = \frac{\partial C_{\phi_k}(t,t')}{\partial t \partial t'} =: C''_{\phi_k}(t, t').$

2.7 Matching Gradients

Given the joint distribution over states and their derivatives (3) as well as the ODEs (2), we therefore have two expressions for the state derivatives:

$\dot{\mathbf{X}} = \mathbf{F} + \epsilon_1, \epsilon_1 \sim \mathcal{N}(\epsilon_1; \mathbf{0}, \mathbf{I}\gamma)$

$\dot{\mathbf{X}} = {}'\mathbf{C}_\phi \mathbf{C}_\phi^{-1} \mathbf{X} + \epsilon_2, \epsilon_2 \sim \mathcal{N}(\epsilon_2; \mathbf{0}, \mathbf{A})$

where $\mathbf{F} := \mathbf{f}(\mathbf{X}, \theta)$, $\mathbf{A} := \mathbf{C}''_\phi - {}'\mathbf{C}_\phi \mathbf{C}_\phi^{-1} \mathbf{C}'_\phi$ and γ is the error variance in the ODEs. Note that, in a deterministic system, the output of the ODEs $\mathbf{F}$ should equal the state derivatives $\dot{\mathbf{X}}$. However, in the first equation above we relax this contraint by adding stochasticity to the state derivatives $\dot{\mathbf{X}}$ in order to compensate for a potential model mismatch. The second equation above is obtained by deriving the conditional distribution for $\dot{\mathbf{X}}$ from the joint distribution in equation (3). Equating the two expressions in the equations above we can eliminate the unknown state derivatives $\dot{\mathbf{X}}$:

$$\mathbf{F} = {}'\mathbf{C}_\phi \mathbf{C}_\phi^{-1} \mathbf{X} + \epsilon_0 \qquad (4),$$

with $\epsilon_0 := \epsilon_2 - \epsilon_1$.

```
[dC_times_invC,inv_C,A_plus_gamma_inv] = kernel_function(kernel,state,time.est);
```

2.8 State Couplings in ODEs

```
coupling_idx = find_state_couplings_in_odes(ode,symbols);
```

2.9 Rewrite ODEs as Linear Combination in Parameters

We rewrite the ODEs in equation (2) as a linear combination in the parameters:

$\mathbf{B}_{\theta k}\theta + \mathbf{b}_{\theta k} \stackrel{!}{=} \mathbf{f}_k(\mathbf{X}, \theta) \qquad (5)$,

where matrices $\mathbf{B}_{\theta k}$ and $\mathbf{b}_{\theta k}$ are defined such that the ODEs $\mathbf{f}_k(\mathbf{X}, \theta)$ are expressed as a linear combination in θ.

```
[ode_param.lin_comb.B,ode_param.lin_comb.b] = rewrite_odes_as_linear_combination_in_parameters(
```

2.10 Posterior over ODE Parameters

Inserting (5) into (4) and solving for θ yields:

$\theta = \mathbf{B}_\theta^+ \left({}'\mathbf{C}_\phi \mathbf{C}_\phi^{-1} \mathbf{X} - \mathbf{b}_\theta + \epsilon_0 \right)$,

where $\mathbf{B}_\theta^+$ denotes the pseudo-inverse of $\mathbf{B}_\theta$.

Since $\mathbf{C}_\phi$ is block diagonal we can rewrite the expression above as:

$\theta = \left(\mathbf{B}_\theta^T \mathbf{B}_\theta \right)^{-1} \mathbf{B}_\theta^T \left(\sum_k {}'\mathbf{C}_{\phi_k} \mathbf{C}_{\phi_k}^{-1} \mathbf{X}_k - \mathbf{b}_{\theta k} + \epsilon_0^{(k)} \right)$,

$= \left(\mathbf{B}_\theta^T \mathbf{B}_\theta \right)^{-1} \left(\sum_k \mathbf{B}_{\theta k}^T \left({}'\mathbf{C}_{\phi_k} \mathbf{C}_{\phi_k}^{-1} \mathbf{X}_k - \mathbf{b}_{\theta k} + \epsilon_0^{(k)} \right) \right)$,

where we subsitute the Moore-Penrose inverse for the pseudo-inverse (i.e. $\mathbf{B}_\theta^+ = \left(\mathbf{B}_\theta^T \mathbf{B}_\theta \right)^{-1} \mathbf{B}_\theta^T$).

We can therefore derive the posterior distribution over ODE parameters:

$p(\theta \mid \mathbf{X}, \phi, \gamma) = \mathcal{N}\left(\theta; \mathbf{B}_\theta^+ \left({}'\mathbf{C}_\phi \mathbf{C}_\phi^{-1} \mathbf{X} - \mathbf{b}_\theta \right),\ \mathbf{B}_\theta^+ \left(\mathbf{A} + \mathbf{I}\gamma \right) \mathbf{B}_\theta^{+T} \right)$

$= \mathcal{N}\left(\theta; \left(\mathbf{B}_\theta^T \mathbf{B}_\theta \right)^{-1} \left(\sum_k \mathbf{B}_{\theta k}^T \left({}'\mathbf{C}_{\phi k} \mathbf{C}_{\phi k}^{-1} \mathbf{X}_k - \mathbf{b}_{\theta k} \right) \right),\ \mathbf{B}_\theta^+ \left(\mathbf{A} + \mathbf{I}\gamma \right) \mathbf{B}_\theta^{+T} \right)$

$= \prod_k \mathcal{N}\left(\theta; \left(\mathbf{B}_\theta^T \mathbf{B}_\theta \right)^{-1} \left(\mathbf{B}_{\theta k}^T \left({}'\mathbf{C}_{\phi k} \mathbf{C}_{\phi k}^{-1} \mathbf{X}_k - \mathbf{b}_{\theta k} \right) \right),\ \mathbf{B}_{\theta k}^+ \left(\mathbf{A}_k + \mathbf{I}\gamma \right) \mathbf{B}_{\theta k}^{+T} \right) \qquad (6)$

2.11 Rewrite ODEs as Linear Combination in Individual States

We rewrite the expression $\mathbf{f}(\mathbf{X}, \theta) - {}'\mathbf{C}_\phi \mathbf{C}_\phi^{-1} \mathbf{X}$ in equation (4) as a linear combination in the individual state $\mathbf{x}_u$:

$\mathbf{R}_{uk} \mathbf{x}_u + \mathbf{r}_{uk} \stackrel{!}{=} \mathbf{f}_k(\mathbf{X}, \theta)$.

where matrices $\mathbf{R}_{uk}$ and $\mathbf{r}_{uk}$ are defined such that the ODE $\mathbf{f}_k(\mathbf{X}, \theta)$ is expressed as a linear combination in the individual state $\mathbf{x}_u$.

```
[state.lin_comb.R,state.lin_comb.r] = rewrite_odes_as_linear_combination_in_ind_states(ode,symb
```

2.12 Posterior over Individual States

Given the linear combination of the ODEs w.r.t. an individual state, we define the matrices $\mathbf{B}_u$ and $\mathbf{b}_u$ such that the expression $\mathbf{f}(\mathbf{X}, \theta) - {}'\mathbf{C}_\phi \mathbf{C}_\phi^{-1} \mathbf{X}$ is rewritten as a linear combination in an individual state:

$$\mathbf{B}_u \mathbf{x}_u + \mathbf{b}_u \stackrel{!}{=} \mathbf{f}(\mathbf{X}, \theta) \qquad (7).$$

Inserting (7) into (4) and solving for $\mathbf{x}_u$ yields:

$$\mathbf{x}_u = \mathbf{B}_u^+ \left(\epsilon_0 - \mathbf{b}_u\right),$$

Since $\mathbf{C}_\phi$ is block diagonal we can rewrite the expression above as:

$$\mathbf{x}_u = \left(\mathbf{B}_u \mathbf{B}_u^T\right)^{-1} \mathbf{B}_u^T \textstyle\sum_k \left(\epsilon_0^{(k)} - \mathbf{b}_{uk}\right)$$

$$= \left(\mathbf{B}_u \mathbf{B}_u^T\right)^{-1} \textstyle\sum_k \mathbf{B}_{uk}^T \left(\epsilon_0^{(k)} - \mathbf{b}_{uk}\right),$$

where $\mathbf{B}_u^+$ denotes the pseudo-inverse of $\mathbf{B}_u$. We can therefore derive the posterior distribution over an individual state $\mathbf{x}_u$:

$$p(\mathbf{x}_u \mid \mathbf{X}_{-u}, \phi, \gamma) = \mathcal{N}\left(\mathbf{x}_u; -\mathbf{B}_u^+ \mathbf{b}_u,\ \mathbf{B}_u^+ \left(\mathbf{A} + \mathbf{I}\gamma\right) \mathbf{B}_u^{+T}\right)$$

$$= \mathcal{N}\left(\mathbf{x}_u; \left(\mathbf{B}_u \mathbf{B}_u^T\right)^{-1} \left(-\textstyle\sum_k \mathbf{B}_{uk}^T \mathbf{b}_{uk}\right),\ \mathbf{B}_u^+ \left(\mathbf{A} + \mathbf{I}\gamma\right) \mathbf{B}_u^{+T}\right) \qquad (8),$$

with $\mathbf{X}_{-u}$ denoting the set of all states except state $\mathbf{x}_u$.

2.13 Mean-field Variational Inference

To infer the parameters θ, we want to find the maximum a posteriori estimate (MAP):

$$\theta^* := arg\max_\theta \ \ln p(\theta \mid \mathbf{Y}, \phi, \gamma, \sigma)$$

$$= arg\max_\theta \ \ln \textstyle\int p(\theta, \mathbf{X} \mid \mathbf{Y}, \phi, \gamma, \sigma) d\mathbf{X}$$

$$= arg\max_\theta \ \ln \textstyle\int p(\theta \mid \mathbf{X}, \phi, \gamma) p(\mathbf{X} \mid \mathbf{Y}, \phi, \sigma) d\mathbf{X} \qquad (9).$$

However, the integral above is intractable due to the strong couplings induced by the nonlinear ODEs $\mathbf{f}$ which appear in the term $p(\theta \mid \mathbf{X}, \phi, \gamma)$.

We use mean-field variational inference to establish variational lower bounds that are analytically tractable by decoupling state variables from the ODE parameters as well as decoupling the state variables from each other. Note that, since the ODEs described by equation (2) are **locally linear**, both conditional distributions $p(\theta \mid \mathbf{X}, \mathbf{Y}, \phi, \gamma, \sigma)$ (equation (6)) and $p(\mathbf{x}_u \mid \theta, \mathbf{X}_{-u}, \mathbf{Y}, \phi, \gamma, \sigma)$ (equation (8)) are analytically tractable and Gaussian distributed as mentioned previously.

The decoupling is induced by designing a variational distribution $Q(\theta, \mathbf{X})$ which is restricted to the family of factorial distributions:

$$\mathcal{Q} := \Big\{ Q : Q(\theta, \mathbf{X}) = q(\theta) \prod_u q(\mathbf{x}_u) \Big\}.$$

The particular form of $q(\theta)$ and $q(\mathbf{x}_u)$ are designed to be Gaussian distributed which places them in the same family as the true full conditional distributions. To find the optimal factorial distribution we minimize the Kullback-Leibler divergence between the variational and the true posterior distribution:

$$\hat{Q} := arg\min_{Q(\theta,\mathbf{X})\in\mathcal{Q}} \mathrm{KL}\left[Q(\theta,\mathbf{X}) \mid\mid p(\theta,\mathbf{X} \mid \mathbf{Y},\phi,\gamma,\sigma)\right] \qquad (10),$$

where $\hat{Q}$ is the proxy distribution. The proxy distribution that minimizes the KL-divergence (10) depends on the true full conditionals and is given by:

$$\hat{q}(\theta) \propto \exp\Big(E_{Q_{-\theta}} \ln p(\theta \mid \mathbf{X}, \mathbf{Y}, \phi, \gamma, \sigma) \Big) \qquad (11)$$

$$\hat{q}(\mathbf{x}_u) \propto \exp\big(E_{Q_{-u}} \ln p(\mathbf{x}_u \mid \theta, \mathbf{X}_{-u}, \mathbf{Y}, \phi, \gamma, \sigma) \big) \qquad (12).$$

2.14 Fitting observations of state trajectories

We fit the observations of state trajectories by standard GP regression. The data-informed distribution $p(\mathbf{X} \mid \mathbf{Y}, \phi, \sigma)$ in euqation (9) can be determined analytically using Gaussian process regression with the GP prior $p(\mathbf{X} \mid \phi) = \prod_k \mathcal{N}(\mathbf{x}_k; \mathbf{0}, \mathbf{C}_\phi)$:

$p(\mathbf{X} \mid \mathbf{Y}, \phi, \gamma) = \prod_k \mathcal{N}(\mathbf{x}_k; \mu_k(\mathbf{y}_k), \Sigma_k)$,

where $\boldsymbol{\mu_k(\mathbf{y}_k) := \sigma_k^{-2}} \left(\boldsymbol{\sigma_k^{-2}\mathbf{I} + \mathbf{C}_{\phi_k}^{-1}}\right)^{-1} \mathbf{y_k}$ and $\boldsymbol{\Sigma_k^{-1} := \sigma_k^{-2}\mathbf{I} + \mathbf{C}_{\phi_k}^{-1}}$.

```
[mu,inv_sigma] = fitting_state_observations(state,inv_C,obs_to_state_relation,simulation);
```

2.15 Coordinate Ascent Variational Gradient Matching

We minimize the KL-divergence in equation (10) by coordinate descent (where each step is analytically tractable) by iterating between determining the proxy for the distribution over ODE parameters $\hat{q}(\theta)$ and the proxies for the distribution over individual states $\hat{q}(\mathbf{x}_u)$.

```
state.proxy.mean = mu;                                    % Initialize the state estimation by the
for i = 1:opt_settings.coord_ascent_numb_iter
```

Expanding the proxy distribution in equation (11) for θ yields:

$$\hat{q}(\theta) \overset{(a)}{\propto} \exp\Big(E_{Q_{-\theta}} \ln p(\theta \mid \mathbf{X}, \mathbf{Y}, \phi, \gamma, \sigma) \Big)$$

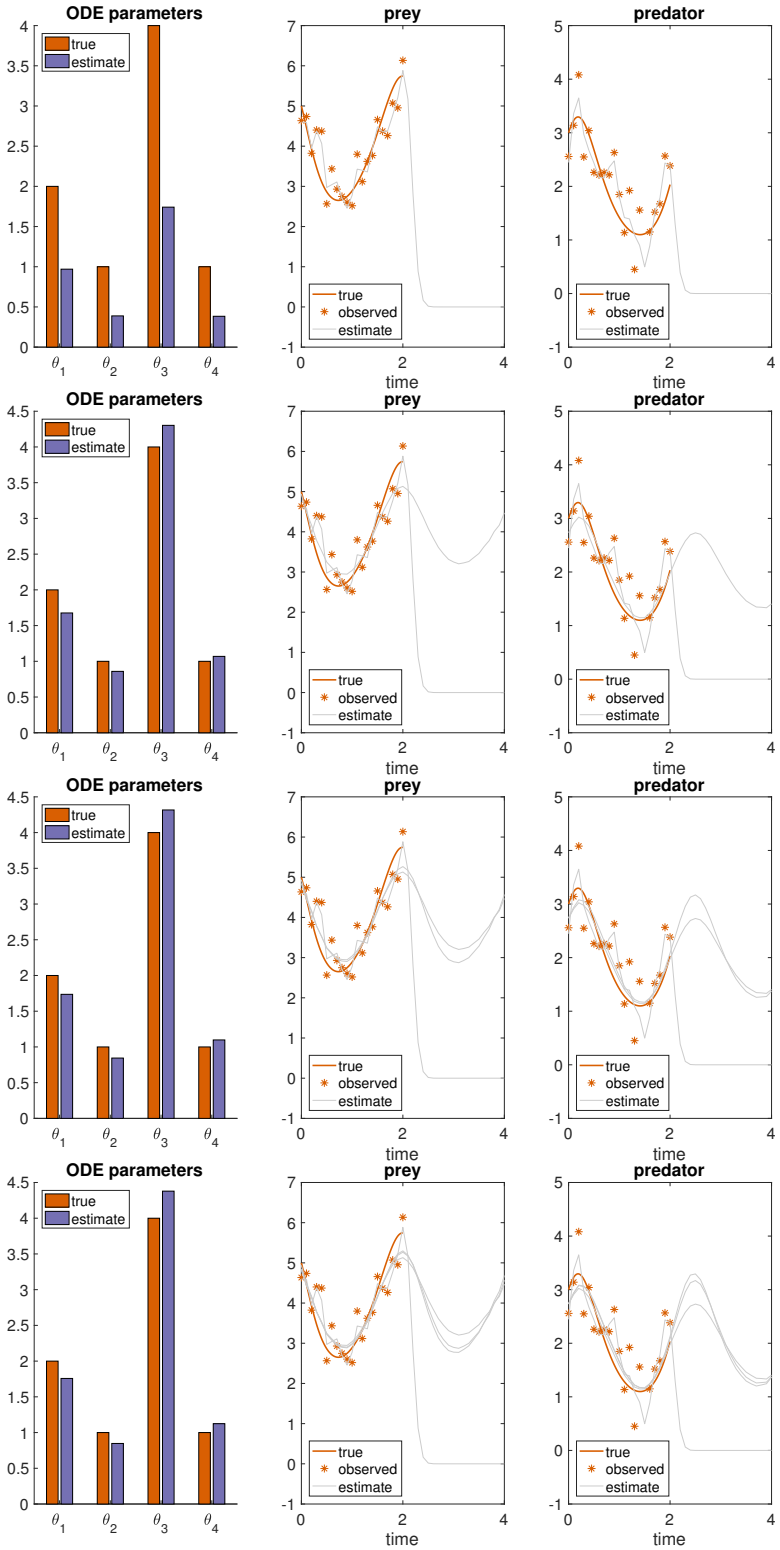
$$\overset{(b)}{\propto} \exp\Big(E_{Q_{-\theta}} \ln \mathcal{N}\left(\theta; \mathbf{B}_\theta^+ \left('\mathbf{C}_\phi \mathbf{C}_\phi^{-1} \mathbf{X} - \mathbf{b}_\theta\right),\ \mathbf{B}_\theta^+ \left(\mathbf{A} + \mathbf{I}\gamma\right) \mathbf{B}_\theta^{+T}\right) \Big)$$

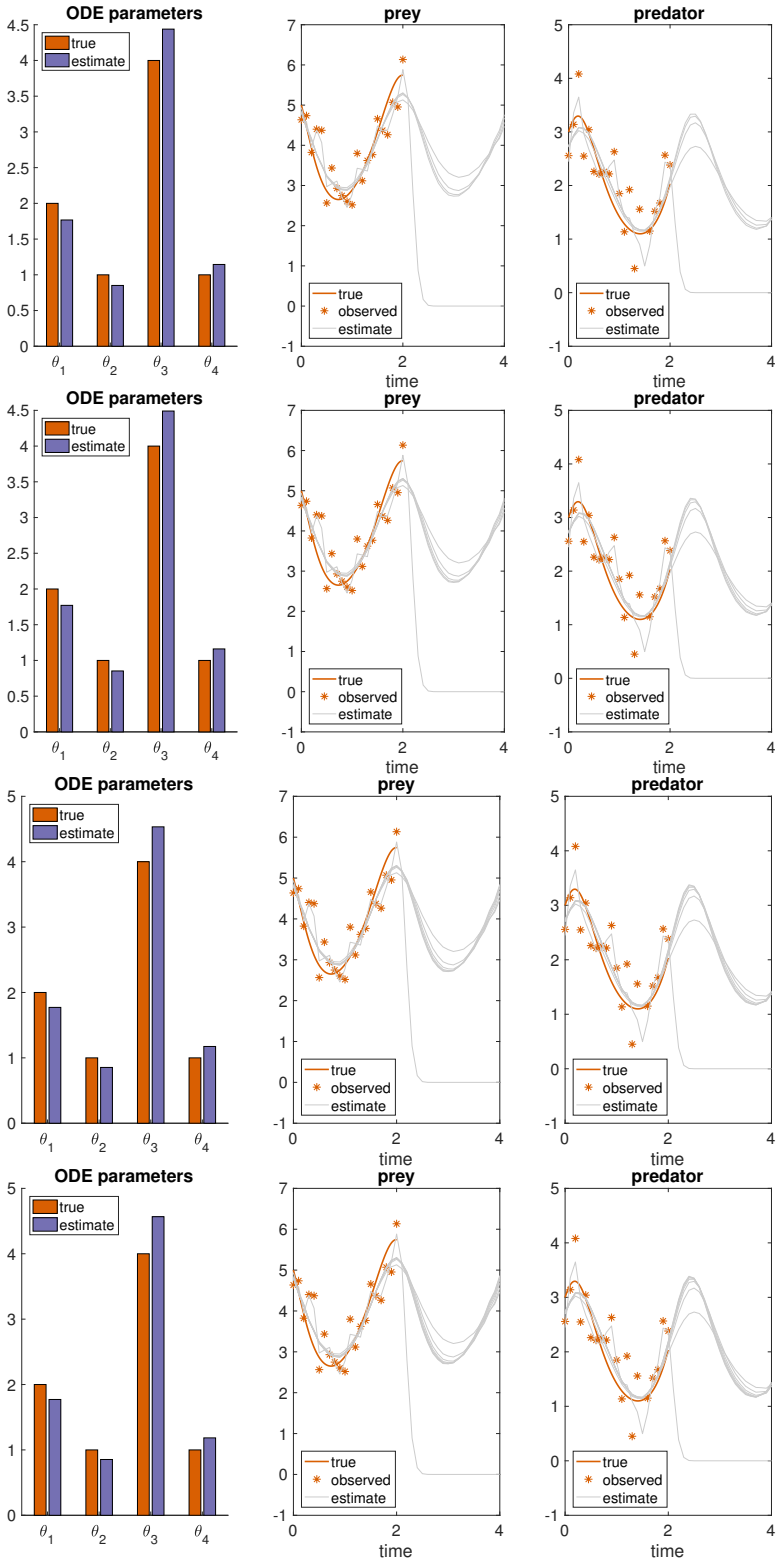
$$= \exp\Big(E_{Q_{-\theta}} \mathcal{N}\left(\theta; \left(\mathbf{B}_\theta^T \mathbf{B}_\theta\right)^{-1} \left(\textstyle\sum_k \mathbf{B}_{\theta k}^T \left('\mathbf{C}_{\phi k} \mathbf{C}_{\phi k}^{-1} \mathbf{X}_k - \mathbf{b}_{\theta k}\right)\right),\ \mathbf{B}_\theta^+ \left(\mathbf{A} + \mathbf{I}\gamma\right) \mathbf{B}_\theta^{+T}\right) \Big),$$

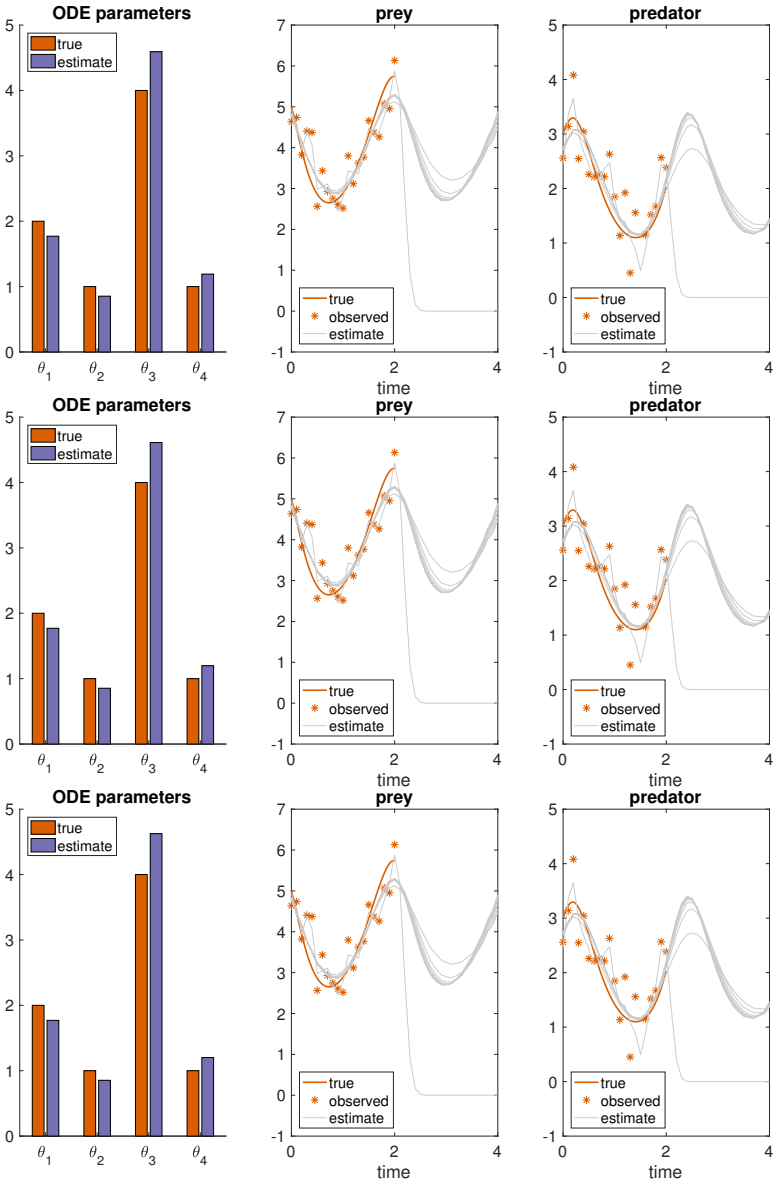
which can be normalized analytically due to its exponential quadratic form. In (a) we recall that the ODE parameters depend only indirectly on the observations $\mathbf{Y}$ through the states $\mathbf{X}$ and in (b) we substitute $p(\theta \mid \mathbf{X}, \phi, \gamma)$ by its density given in equation (6).

```
[param_proxy_mean,param_proxy_inv_cov] = proxy_for_ode_parameters(state.proxy.mean,dC_times_
    ode_param.lin_comb,symbols,A_plus_gamma_inv,opt_settings);

if i==1 || ~mod(i,20)
    plot_results(h_states,h_param,state,time,simulation,param_proxy_mean,...
        p,'not_final');
end
```




ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time



Expanding the proxy distribution in equation (12) over the individual state $\mathbf{x}_u$:

$$\hat{q}(\mathbf{x}_u) \overset{(a)}{\propto} \exp\left(E_{Q_{-u}} \ln(p(\mathbf{x}_u \mid \theta, \mathbf{X}_{-u}, \phi, \gamma) p(\mathbf{x}_u \mid \mathbf{Y}, \phi, \sigma)) \right)$$

$$\overset{(b)}{\propto} \exp\left(E_{Q_{-u}} \ln \mathcal{N}\left(\mathbf{x}_u; -\mathbf{B}_u^{+}\mathbf{b}_u,\ \mathbf{B}_u^{+}\left(\mathbf{A} + \mathbf{I}\gamma\right)\mathbf{B}_u^{+T}\right) + E_{Q_{-u}} \ln \mathcal{N}\left(\mathbf{x}_u; \mu_u(\mathbf{Y}), \Sigma_u\right) \right)$$

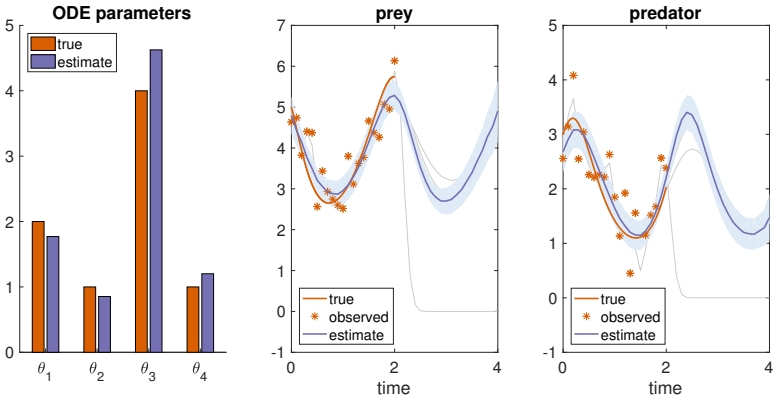
$$= \exp\left(E_{Q_{-u}} \ln \mathcal{N}\left(\mathbf{x}_u; \left(\mathbf{B}_u\mathbf{B}_u^T\right)^{-1}\left(-\textstyle\sum_k \mathbf{B}_{uk}^T\mathbf{b}_{uk}\right),\ \mathbf{B}_u^{+}\left(\mathbf{A} + \mathbf{I}\gamma\right)\mathbf{B}_u^{+T}\right) + E_{Q_{-u}} \ln \mathcal{N}\left(\mathbf{x}_u; \mu_u(\mathbf{Y}), \Sigma_u\right) \right),$$

which, once more, can be normalized analytically due to its exponential quadratic form. In (a) we decompose the full conditional into an ODE-informed distribution and a data-informed distribution and in (b) we substitute the ODE-informed distribution $p(\mathbf{x}_u \mid \theta, \mathbf{X}_{-u}, \phi, \gamma)$ with its density given by equation (8).

```
    [state.proxy.mean,state.proxy.inv_cov] = proxy_for_ind_states(state.lin_comb,state.proxy.mea
      param_proxy_mean',dC_times_invC,coupling_idx.states,symbols,mu,inv_sigma,state.obs_idx,..
        A_plus_gamma_inv,opt_settings);

end
```

```
plot_results(h_states,h_param,state,time,simulation,param_proxy_mean,p,'final');
```



2.16 Time Taken

```
disp(['time taken: ' num2str(toc) ' seconds'])
```

```
time taken: 64.8955 seconds
```

2.17 References

- **Gorbach, N.S.** , **Bauer, S.** and Buhmann, J.M., Scalable Variational Inference for Dynamical Systems. 2017a. Neural Information Processing Systems (NIPS).

  https://papers.nips.cc/paper/7066-scalable-variational-inference-for-dynamical-systems.pdf

- **Bauer, S.** , **Gorbach, N.S.** and Buhmann, J.M., Efficient and Flexible Inference for Stochastic Differential Equations. 2017b. Neural Information Processing Systems (NIPS).

  https://papers.nips.cc/paper/7274-efficient-and-flexible-inference-for-stochastic-systems

- Wenk, P., Gotovos, A., Bauer, S., Gorbach, N.S., Krause, A. and Buhmann, J.M., Fast Gaussian Process Based Gradient Matching for Parameters Identification in Systems of Nonlinear ODEs. 2018. In submission to Conference on Uncertainty in Artificial Intelligence (UAI).
- Calderhead, B., Girolami, M. and Lawrence. N.D., 2002. Accelerating Bayesian inference over nonlinear differential equation models. *In Advances in Neural Information Processing Systems (NIPS)* . 22.

The authors in bold font have contributed equally to their respective papers.

2.18 Subroutines

Gradient matching with Gaussian processes assumes a joint Gaussian process prior on states and their derivatives:

$$\begin{pmatrix} \mathbf{X} \\ \dot{\mathbf{X}} \end{pmatrix} \sim \mathcal{N}\left(\begin{matrix} \mathbf{X} \\ \dot{\mathbf{X}} \end{matrix} ; \begin{matrix} \mathbf{0} \\ \mathbf{0} \end{matrix}, \begin{matrix} \mathbf{C}_\phi & \mathbf{C}'_\phi \\ {}'\mathbf{C}_\phi & \mathbf{C}''_\phi \end{matrix} \right),$$

$$\text{cov}(x_k(t), x_k(t)) = C_{\phi_k}(t, t')$$

$$\text{cov}(\dot{x}_k(t), x_k(t)) = \frac{\partial C_{\phi_k}(t,t')}{\partial t} =: C'_{\phi_k}(t, t')$$

$\text{cov}(x_k(t), \dot{x}_k(t)) = \frac{\partial C_{\phi_k}(t,t')}{\partial t'} =: {}'C_{\phi_k}(t,t')$

$\text{cov}(\dot{x}_k(t), \dot{x}_k(t)) = \frac{\partial C_{\phi_k}(t,t')}{\partial t \partial t'} =: C''_{\phi_k}(t,t').$

```
function [dC_times_invC,inv_C,A_plus_gamma_inv] = kernel_function(kernel,state,time_est)

kernel.param_sym = sym('rbf_param%d',[1,2]); assume(kernel.param_sym,'real');
kernel.time1 = sym('time1'); assume(kernel.time1,'real'); kernel.time2 = sym('time2'); assume(ker
kernel.func = kernel.param_sym(1).*exp(-(kernel.time1-kernel.time2).^2./(kernel.param_sym(2).^2
kernel.name = 'rbf';
```

kernel derivatives

```
for i = 1:length(kernel)
    kernel.func_d = diff(kernel.func,kernel.time1);
    kernel.func_dd = diff(kernel.func_d,kernel.time2);
   GP.fun = matlabFunction(kernel.func,'Vars',{kernel.time1,kernel.time2,kernel.param_sym});
   GP.fun_d = matlabFunction(kernel.func_d,'Vars',{kernel.time1,kernel.time2,kernel.param_sym})
   GP.fun_dd = matlabFunction(kernel.func_dd,'Vars',{kernel.time1,kernel.time2,kernel.param_sym
end
```

populate GP covariance matrix

```
for t=1:length(time_est)
    C(t,:)=GP.fun(time_est(t),time_est,kernel.param);
    dC(t,:)=GP.fun_d(time_est(t),time_est,kernel.param);
    Cd(t,:)=GP.fun_d(time_est,time_est(t),kernel.param);
    ddC(t,:)=GP.fun_dd(time_est(t),time_est,kernel.param);
end
```

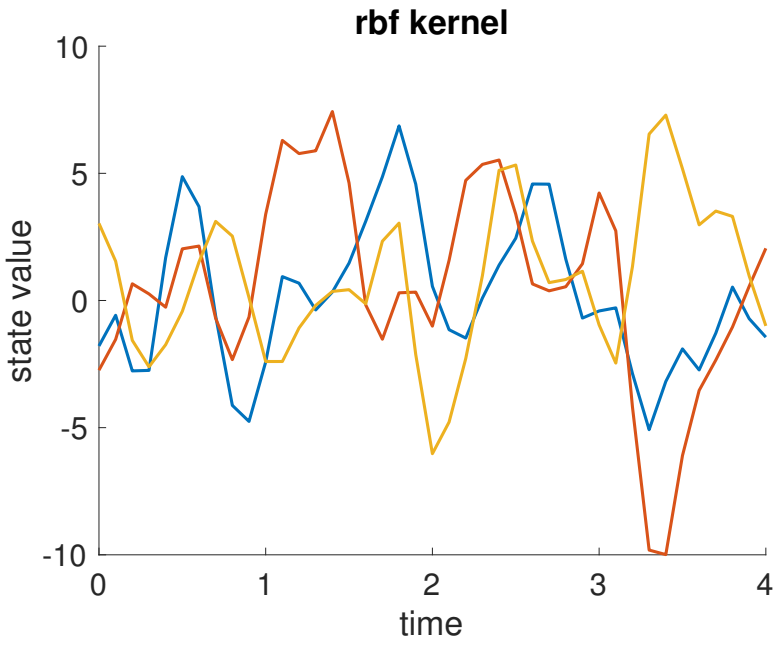
GP covariance scaling

```
[~,D] = eig(C); perturb = abs(max(diag(D))-min(diag(D))) / 10000;
if any(diag(D)<1e-6); C(logical(eye(size(C,1)))) = C(logical(eye(size(C,1)))) + perturb.*rand(si
[~,D] = eig(C);
if any(diag(D)<0); error('C has negative eigenvalues!'); elseif any(diag(D)<1e-6); warning('C is
inv_C = inv_chol(chol(C,'lower'));

dC_times_invC = dC * inv_C;
```

plot samples from GP prior

```
figure(3);
hold on; plot(time_est,mvnrnd(zeros(1,length(time_est)),C(:,:,1),3),'LineWidth',2);
h1 = gca; h1.FontSize = 20; h1.XLabel.String = 'time'; h1.YLabel.String = 'state value';
h1.Title.String = [kernel.name ' kernel'];
```



determine $\mathbf{A} + \mathbf{I}\gamma$:

```
A = ddC - dC_times_invC * Cd;
A_plus_gamma = A + state.derivative_variance(1) .* eye(size(A));
A_plus_gamma = 0.5.*(A_plus_gamma+A_plus_gamma');        % ensure that A plus gamma is symmetric
A_plus_gamma_inv = inv_chol(chol(A_plus_gamma,'lower'));

end
```

We fit the observations of state trajectories by standard GP regression.

$p(\mathbf{X} \mid \mathbf{Y}, \phi, \gamma) = \prod_k \mathcal{N}(\mathbf{x}_k; \mu_k(\mathbf{y}_k), \Sigma_k)$,

where $\boldsymbol{\mu_k}(\mathbf{y_k}) := \boldsymbol{\sigma_k^{-2}} \left(\boldsymbol{\sigma_k^{-2}}\mathbf{I} + \mathbf{C}_{\boldsymbol{\phi_k}}^{-1}\right)^{-1} \mathbf{y_k}$ and $\boldsymbol{\Sigma_k^{-1}} := \boldsymbol{\sigma_k^{-2}}\mathbf{I} + \mathbf{C}_{\boldsymbol{\phi_k}}^{-1}$.

```
function [mu_u,inv_sigma_u,state] = fitting_state_observations(state,inv_C,obs_to_state_relatio
    simulation)
```

Dimensions

```
numb_states = size(state.sym.mean,2);
numb_time_points = size(state.sym.mean,1);
```

Variance of state observations

```
state_obs_variance = simulation.state_obs_variance(state.obs);
```

Form block-diagonal matrix out of $\mathbf{C}_{\phi_k}^{-1}$

```
inv_C_replicas = num2cell(inv_C(:,:,ones(1,numb_states)),[1,2]);
inv_C_blkdiag = sparse(blkdiag(inv_C_replicas{:}));
```

GP posterior inverse covariance matrix: $\boldsymbol{\Sigma_k^{-1}} := \boldsymbol{\sigma_k^{-2}}\mathbf{I} + \mathbf{C}_{\boldsymbol{\phi_k}}^{-1}$

```
dim = size(state_obs_variance,1)*size(state_obs_variance,2);
% covariance matrix of error term (big E):
D = spdiags(reshape(state_obs_variance.^(-1),[],1),0,dim,dim) * speye(dim);
A_times_D_times_A = obs_to_state_relation' * D * obs_to_state_relation;
inv_sigma = A_times_D_times_A + inv_C_blkdiag;
```

GP posterior mean: $\boldsymbol{\mu_k}(\mathbf{y_k}) := \boldsymbol{\sigma_k^{-2}} \left(\boldsymbol{\sigma_k^{-2}}\mathbf{I} + \mathbf{C}_{\boldsymbol{\phi_k}}^{\mathbf{-1}}\right)^{\mathbf{-1}} \mathbf{y_k}$

```
mu = inv_sigma \ obs_to_state_relation' * D * reshape(state.obs,[],1);
```

Reshape GP mean

```
mu_u = zeros(numb_time_points,numb_states);
for u = 1:numb_states
    idx = (u-1)*numb_time_points+1:(u-1)*numb_time_points+numb_time_points;
    mu_u(:,u) = mu(idx);
end
```

Reshape GP inverse covariance matrix

```
inv_sigma_u = zeros(numb_time_points,numb_time_points,numb_states);
for i = 1:numb_states
    idx = [(i-1)*numb_time_points+1:(i-1)*numb_time_points+numb_time_points];
    inv_sigma_u(:,:,i) = inv_sigma(idx,idx);
end

end
```

```
function coupling_idx = find_state_couplings_in_odes(ode,symbols)

state_sym = sym('state%d',[1,length(ode.system)]); assume(state_sym,'real');
for k = 1:length(ode.system)
    tmp_idx = ismember(state_sym,symvar(ode.system_sym(k))); tmp_idx(:,k) = 1;
    ode_couplings_states(k,tmp_idx) = 1;
end

for u = 1:length(symbols.state)
    coupling_idx.states{u} = find(ode_couplings_states(:,u));
end

end
```

$\mathbf{B}_{\theta k}\theta + \mathbf{b}_{\theta k} \stackrel{!}{=} \mathbf{f}_k(\mathbf{X}, \theta)$,

where matrices $\mathbf{B}_{\theta k}$ and $\mathbf{b}_{\theta k}$ are defined such that the ODEs $\mathbf{f}_k(\mathbf{X}, \theta)$ are expressed as a linear combination in θ.

```
function [B,b] = rewrite_odes_as_linear_combination_in_parameters(ode,symbols)
```

Initialization of symbolic variables

```
param_sym = sym('param%d',[1,length(symbols.param)]); assume(param_sym,'real');
state_sym = sym('state%d',[1,length(symbols.state)]); assume(state_sym,'real');
state0_sym = sym('state0'); assume(state0_sym,'real');
state_const_sym = sym('state_const'); assume(state_const_sym,'real');
```

Rewrite ODEs as linear combinations in parameters (global)

```
[B_sym,b_sym] = equationsToMatrix(ode.system_sym,param_sym);
b_sym = -b_sym; % See the documentation of the function "equationsToMatrix"
```

Operations locally w.r.t. ODEs

```
for k = 1:length(ode.system)
    B_sym(k,B_sym(k,:)=='0') = state0_sym;
    for i = 1:length(B_sym(k,:))
        sym_var = symvar(B_sym(k,i));
        if isempty(sym_var)
            B_sym(k,i) = B_sym(k,i) + state0_sym;
        end
    end
    B{k} = matlabFunction(B_sym(k,:),'Vars',{state_sym,state0_sym,state_const_sym});
    b{k} = matlabFunction(b_sym(k,:),'Vars',{state_sym,state0_sym,state_const_sym});
end

end
```

$\mathbf{R}_{uk}\mathbf{x}_u + \mathbf{r}_{uk} \stackrel{!}{=} \mathbf{f}_k(\mathbf{X}, \theta)$.

where matrices $\mathbf{R}_{uk}$ and $\mathbf{r}_{uk}$ are defined such that the ODEs $\mathbf{f}_k(\mathbf{X}, \theta)$ is rewritten as a linear combination in the individual state $\mathbf{x}_u$.

```
function [R,r] = rewrite_odes_as_linear_combination_in_ind_states(ode,symbols,coupling_idx)
```

Initialization of symbolic variables

```
param_sym = sym('param%d',[1,length(symbols.param)]); assume(param_sym,'real');
state_sym = sym('state%d',[1,length(symbols.state)]); assume(state_sym,'real');
state0_sym = sym('state0'); assume(state0_sym,'real');
state_const_sym = sym('state_const'); assume(state_const_sym,'real');
```

Rewrite ODEs as linear combinations in parameters (locally)

```
for u = 1:length(symbols.state)
    for k = coupling_idx{u}'
      [R_sym,r_sym] = equationsToMatrix(ode.system{k}(state_sym,param_sym'),state_sym(:,u));
        r_sym = -r_sym; % See the documentation of the function "equationsToMatrix"

        R{u,k} = matlabFunction(R_sym,'Vars',{state_sym,param_sym});
        r{u,k} = matlabFunction(r_sym,'Vars',{state_sym,param_sym});
    end
end
```

```
end
```

$$\hat{q}(\theta) \propto \exp\left(E_{Q_{-\theta}} \ln \mathcal{N}\left(\theta; \left(\mathbf{B}_\theta^T \mathbf{B}_\theta\right)^{-1} \left(\sum_k \mathbf{B}_{\theta k}^T \left('\mathbf{C}_{\phi k} \mathbf{C}_{\phi k}^{-1} \mathbf{X}_k - \mathbf{b}_{\theta k}\right)\right),\ \mathbf{B}_\theta^+ \left(\mathbf{A} + \mathbf{I}\gamma\right) \mathbf{B}_\theta^{+T}\right)\right),$$

```
function [param_proxy_mean,param_proxy_inv_cov] = proxy_for_ode_parameters(state_proxy_mean,...
    dC_times_invC,lin_comb,symbols,A_plus_gamma_inv,opt_settings)
```

Initialization

```
state0 = zeros(size(dC_times_invC,1),1);
param_proxy_inv_cov = zeros(length(symbols.param));
global_scaling = zeros(length(symbols.param));
global_mean = zeros(length(symbols.param),1);
```

Iteratate through ODEs

```
for k = 1:length(symbols.state)
```

unpack matrices **B** and **b**

```
    B = lin_comb.B{k}(state_proxy_mean,state0,ones(size(state_proxy_mean,1),1));
    b = lin_comb.b{k}(state_proxy_mean,state0,ones(size(state_proxy_mean,1),1));
```

Local operations

```
    if strcmp(opt_settings.pseudo_inv_type,'Moore-Penrose')
```

The Moore-Penrose inverse of $\mathbf{B}_\theta$ is given by: $\mathbf{B}_\theta$ is given by: $\mathbf{B}_\theta^+ := \left(\mathbf{B}_\theta^T \mathbf{B}_\theta\right)^{-1} \mathbf{B}_\theta^T$

local mean: $\mathbf{B}_{\theta k}^T \left('\mathbf{C}_{\phi_k} \mathbf{C}_{\phi k}^{-1} \mathbf{X}_k - \mathbf{b}_{\theta k}\right)$

```
        local_mean = B' * (dC_times_invC * state_proxy_mean(:,k) - b);
        local_scaling = B' * B;
        local_inv_cov = B' * A_plus_gamma_inv * B;

    elseif strcmp(opt_settings.pseudo_inv_type,'modified Moore-Penrose')
```

The modified Moore-Penrose inverse of $\mathbf{B}_\theta$ is given by: $\mathbf{B}_\theta$ is given by: $\mathbf{B}_\theta^+ := \left(\mathbf{B}_\theta^T (\mathbf{A} + \mathbf{I}\gamma) \mathbf{B}_\theta\right)^{-1} \mathbf{B}_\theta^T (\mathbf{A} + \mathbf{I}\gamma)$

local mean: $\mathbf{B}_{\theta k}^T (\mathbf{A} + \mathbf{I}\gamma) \left('\mathbf{C}_{\phi_k} \mathbf{C}_{\phi k}^{-1} \mathbf{X}_k - \mathbf{b}_{\theta k}\right)$

```
      local_mean = B' * A_plus_gamma_inv * (dC_times_invC * state_proxy_mean(:,k) - b);
        local_scaling = B' * A_plus_gamma_inv * B;
        local_inv_cov = local_scaling;

    end
```

Global operations

```
    global_mean = global_mean + local_mean;
    global_scaling = global_scaling + local_scaling;

    % Inverse covariance of ODE param proxy distribution
    param_proxy_inv_cov = param_proxy_inv_cov + local_inv_cov;

end
```

Check scaling of covariance matrix

```
[~,D] = eig(param_proxy_inv_cov);
if any(diag(D)<0)
    warning('param_proxy_inv_cov has negative eigenvalues!');
elseif any(diag(D)<1e-3)
    warning('param_proxy_inv_cov is badly scaled')
    disp('perturbing diagonal of param_proxy_inv_cov')
    perturb = abs(max(diag(D))-min(diag(D))) / 10000;
   param_proxy_inv_cov(logical(eye(size(param_proxy_inv_cov,1)))) = param_proxy_inv_cov(logical
        + perturb.*rand(size(param_proxy_inv_cov,1),1);
end
```

Mean of parameter proxy distribution (option: Moore-penrose inverse example): $\left(\mathbf{B}_\theta^T \mathbf{B}_\theta\right)^{-1} \left(\sum_k \mathbf{B}_{\theta k}^T \left('\mathbf{C}_{\phi k} \mathbf{C}_{\phi k}^{-1} \mathbf{X}_k - \mathbf{b}_{\theta k}\right)\right)$

```
param_proxy_mean = global_scaling \ global_mean;
param_proxy_mean = abs(param_proxy_mean);      % mirroring to preserve magnitude

end
```

$$\hat{q}(\mathbf{x}_u) \propto \exp\big(\, E_{Q_{-u}} \ln \mathcal{N}\left(\mathbf{x}_u; \left(\mathbf{B}_u \mathbf{B}_u^T\right)^{-1} \left(-\textstyle\sum_k \mathbf{B}_{uk}^T \mathbf{b}_{uk}\right),\ \mathbf{B}_u^+ \left(\mathbf{A} + \mathbf{I}\gamma\right) \mathbf{B}_u^{+T}\right)$$
$$+ E_{Q_{-u}} \ln \mathcal{N}\left(\mathbf{x}_u; \mu_u(\mathbf{Y}), \Sigma_u\right)\big),$$

```
function [state_proxy_mean,state_proxy_inv_cov] = proxy_for_ind_states(lin_comb,state_proxy_mea
    ode_param,dC_times_invC,coupling_idx,symbols,mu,inv_sigma,state_obs_idx,...
    A_plus_gamma_inv,opt_settings)
```

Clamp observed states to GP fit

```
if opt_settings.clamp_obs_state_to_GP_fit
    state_enumeration = find(~state_obs_idx);
else
    state_enumeration = 1:length(symbols.state);
end

for u = state_enumeration
```

Initialization

```
    state_proxy_inv_cov(:,:,u) = zeros(size(dC_times_invC));
    global_scaling = zeros(size(dC_times_invC));
    global_mean = zeros(size(dC_times_invC,1),1);
```

Iteratate through ODEs

```
    for k = coupling_idx{u}'
```

unpack matrices **R** and **r**

```
        R = diag(lin_comb.R{u,k}(state_proxy_mean,ode_param));
        r = lin_comb.r{u,k}(state_proxy_mean,ode_param);
        if size(R,1) == 1; R = R.*eye(size(dC_times_invC,1)); end
        if length(r)==1; r = zeros(length(global_mean),1); end
```

Define matrices B and b such that $\mathbf{B}_{uk}\mathbf{x}_u + \mathbf{b}_{uk} \stackrel{!}{=} \mathbf{f}_k(\mathbf{X},\theta) - {}'\mathbf{C}_{\phi_k}\mathbf{C}_{\phi_k}^{-1}\mathbf{X}$

```
        if k~=u
            B = R;
            b = r - dC_times_invC * state_proxy_mean(:,k);
        else
            B = R - dC_times_invC;
            b = r;
        end
```

Local operations

```
        if strcmp(opt_settings.pseudo_inv_type,'Moore-Penrose')
            % local mean: $\mathbf{B}_{uk}^T \left(\epsilon_0^{(k)}
            % -\mathbf{b}_{uk}
            local_mean = -B' * b;
            local_scaling = B' * B;
            local_inv_cov = B' * A_plus_gamma_inv * B;
        elseif strcmp(opt_settings.pseudo_inv_type,'modified Moore-Penrose')
            local_mean = -B' * A_plus_gamma_inv * b;
            local_scaling = B' * A_plus_gamma_inv * B;
            local_inv_cov = local_scaling;
        end
```

Global operations

```
        global_mean = global_mean + local_mean;
        global_scaling = global_scaling + local_scaling;
```

Inverse covariance for state proxy distribution

```
        state_proxy_inv_cov(:,:,u) = state_proxy_inv_cov(:,:,u) + local_inv_cov;

    end
```

Mean of state proxy distribution (option: Moore-penrose inverse example): $\left(\mathbf{B}_u\mathbf{B}_u^T\right)^{-1}\sum_k \mathbf{B}_{uk}^T\left(\epsilon_0^{(k)} - \mathbf{b}_{uk}\right)$

```
   state_proxy_mean(:,u) = (global_scaling + inv_sigma(:,:,u)) \ (global_mean + (inv_sigma(:,:,u)
end

end


function ode = import_odes(symbols)
```

Path to system of ODEs

```
path_ode = './Lotka_Volterra_ODEs.txt';
```

Import ODEs

```
ode.raw = importdata(path_ode);
ode.refined = ode.raw;
```

Refine ODEs

```
for k = 1:length(ode.refined)
for u = 1:length(symbols.state); ode.refined{k} = strrep(ode.refined{k},[symbols.state{u}],['sta
for j = 1:length(symbols.param); ode.refined{k} = strrep(ode.refined{k},symbols.param{j},['param
end
for k = 1:length(ode.refined); ode.system{k} = str2func(['@(state,param)(' ode.refined{k} ')']);

end


function [state,time,ode] = generate_ground_truth(time,state,ode,symbols,simulation)
```

Integration times

```
time.true=0:simulation.int_interval:simulation.final_time;              % ture times
Tindex=length(time.true);                                               % index time
TTT=length(simulation.time_samp);                                % number of sampled points
itrue=round(simulation.time_samp./simulation.int_interval+ones(1,TTT));   % Index of sample time
```

Symbolic computations

```
param_sym = sym('param%d',[1,length(symbols.param)]); assume(param_sym,'real');
state_sym = sym('state%d',[1,length(symbols.state)]); assume(state_sym,'real');
for i = 1:length(ode.system)
    ode.system_sym(i) = ode.system{i}(state_sym,param_sym);
end
```

Fourth order Runge-Kutta (numerical) integration

```
ode_system_mat = matlabFunction(ode.system_sym','Vars',{state_sym',param_sym'});
[~,OutX_solver]=ode45(@(t,x) ode_system_mat(x,simulation.ode_param'), time.true, simulation.ini
state.true_all=OutX_solver;
state.true=state.true_all(itrue,:);
```

Pack

```
state.obs_idx = simulation.state_obs_idx;

end


function [state,time,obs_to_state_relation] = generate_state_obs(state,time,simulation)
```

State observations

```
state_obs_variance = simulation.state_obs_variance(state.true);
state.obs = state.true + sqrt(state_obs_variance) .* randn(size(state.true));
```

Mapping between states and observations

```
if length(simulation.time_samp) < length(time.est)
    time.idx = munkres(pdist2(simulation.time_samp',time.est'));
   time.ind = sub2ind([length(simulation.time_samp),length(time.est)],1:length(simulation.time_
else
    time.idx = munkres(pdist2(time.est',simulation.time_samp'));
   time.ind = sub2ind([length(time.est),length(simulation.time_samp)],1:length(time.est),time.i
end
time.obs_time_to_state_time_relation = zeros(length(simulation.time_samp),length(time.est)); ti
state_mat = eye(size(state.true,2));
obs_to_state_relation = sparse(kron(state_mat,time.obs_time_to_state_time_relation));
time.samp = simulation.time_samp;

end


function [h_states,h_param,p] = setup_plots(state,time,simulation,symbols)
```

Refine ODE parameter symbols

```
for i = 1:length(symbols.param); symbols.param{i} = symbols.param{i}(2:end-1); end
```

Figure size and position setup

```
figure(1); set(1, 'Position', [0, 200, 1200, 500]);
```

ODE parameters

```
h_param = subplot(1,3,1); h_param.FontSize = 20; h_param.Title.String = 'ODE parameters';
set(gca,'XTick',[1:length(symbols.param)]); set(gca,'XTickLabel',symbols.param);
hold on; drawnow
```

States

```
for u = 1:2
   h_states{u} = subplot(1,3,u+1); cla; p.true = plot(time.true,state.true_all(:,u),'LineWidth',2
   hold on; p.obs = plot(simulation.time_samp,state.obs(:,u),'*','Color',[217,95,2]./255,'Marker
   h_states{u}.FontSize = 20; h_states{u}.Title.String = symbols.state{u}(2:end-1); h_states{u}.X
    h_states{u}.XLabel.String = 'time'; hold on;
end

end



function plot_results(h_states,h_param,state,time,simulation,param_proxy_mean,p,plot_type)

for u = 1:2
    if strcmp(plot_type,'final')

        % State proxy variance
        state_proxy_variance = diag(state.proxy.inv_cov(:,:,u)^(-1));
     shaded_region = [state.proxy.mean(:,u)+1*sqrt(state_proxy_variance); flip(state.proxy.mean
     f = fill(h_states{u},[time.est'; flip(time.est',1)], shaded_region, [222,235,247]/255); set

        % Replot true states
     p.true = plot(h_states{u},time.true,state.true_all(:,u),'LineWidth',2,'Color',[217,95,2]./

        % Replot state obbservations
     p.obs = plot(h_states{u},simulation.time_samp,state.obs(:,u),'*','Color',[217,95,2]./255,

        % State proxy mean (final)
     hold on; p.est = plot(h_states{u},time.est,state.proxy.mean(:,u),'Color',[117,112,179]./25
    else
        % state proxy mean (not final)
     hold on; p.est = plot(h_states{u},time.est,state.proxy.mean(:,u),'LineWidth',0.1,'Color',[
    end
    % Specify legend entries
   legend(h_states{u},[p.true,p.obs,p.est],{'true','observed','estimate'},'Location','southwest
end

% ODE parameters
cla(h_param); b = bar(h_param,1:length(param_proxy_mean),[simulation.ode_param',param_proxy_mea
b(1).FaceColor = [217,95,2]./255; b(2).FaceColor = [117,112,179]./255;
h_param.XLim = [0.5,length(param_proxy_mean)+0.5]; h_param.YLimMode = 'auto';
legend(h_param,{'true','estimate'},'Location','northwest');
drawnow

end
```