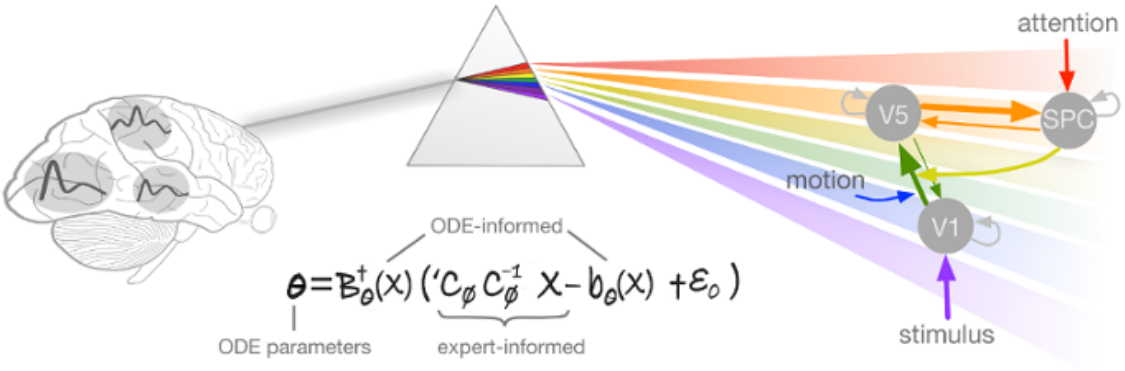
.

Code Documentation of Variational Gradient Matching for Dynamical Systems



Authors:

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

Contents:

Code documentation 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

Theory

2.1 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).

2.2 Prior on States and State Derivatives

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

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

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

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

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

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

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

2.4 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:

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

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

2.5 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:

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

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

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

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

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

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

2.6 Mean-field Variational Inference

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

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

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

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

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:

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

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:

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

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:

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

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

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

2.8 Proxy for ODE Parameters

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

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

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

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

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).

2.9 Proxy for Individual States

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

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

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

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

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).

Chapter 3

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.

3.1 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;
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 observations
simulation.init_val = [5 3];        % initial state values

%symbols of observed states that appear in the 'ODEs.txt' file.
simulation.observed_states = {'[prey]','[predator]'};
```

3.2 User Input

3.2.1 Path to ODEs

```
odes_path = 'Lotka_Volterra_ODEs.txt';
```

3.2.2 Symbols

symbols of states and parameters in the '_ODEs.txt' file

States **x**:

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

ODE parameters θ (symbols of parameters in 'ODEs.txt' file):

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

3.2.3 Kernel

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*ones(1,length(symbols.state)); % gamma for gradient matching model
```

3.2.4 Estimation times

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

3.2.5 Type of pseudo-inverse

Type of pseudo inverse; options: 'Moore-Penrose' or 'modified Moore-Penrose'

```
opt_settings.pseudo_inv_type = 'Moore-Penrose';
```

3.2.6 Optimization settings

```
opt_settings.coord_ascent_numb_iter = 40;  % number of coordinate ascent iterations

% The observed state trajectories are clamped to the trajectories
% determined by standard GP regression (Boolean)
opt_settings.clamp_obs_state_to_GP_fit = false;
```

Plot settings: layout and size

```
plot_settings.size = [1200, 500]; plot_settings.layout = [1,3];
```

3.3 Import ODEs

```
ode = import_odes(symbols,odes_path);
```

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

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

3.4 Simulate Trajectory Observations

3.4.1 Generate ground truth by numerical integration

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

3.4.2 Generate state observations

```
if ~iscell(simulation.observed_states)
    ratio_observed = simulation.observed_states;
    state_obs_idx = zeros(1,simulation.numb_odes,'logical');
    idx = randperm(simulation.numb_odes);
    idx = idx(1:floor(simulation.numb_odes * ratio_observed));
    state_obs_idx(idx) = 1;
    simulation.observed_states = symbols.state(state_obs_idx);
end

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

3.4.3 Symbols

```
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');
```

3.4.4 Setup plots

Only the state dynamics are (partially) observed.

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

tic; %start timer
```

3.5 Prior over States and State Derivatives

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

3.6 State Couplings in ODEs

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

3.7 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(ode,symbols);
```

3.8 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,symbols,coupling_idx.states);
```

3.9 Fitting observations of state trajectories

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

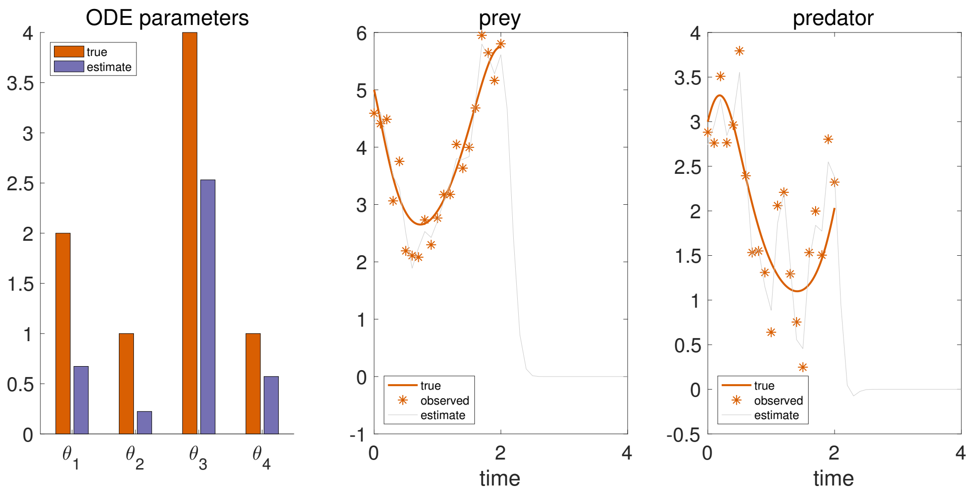
3.10 Coordinate Ascent Variational Gradient Matching

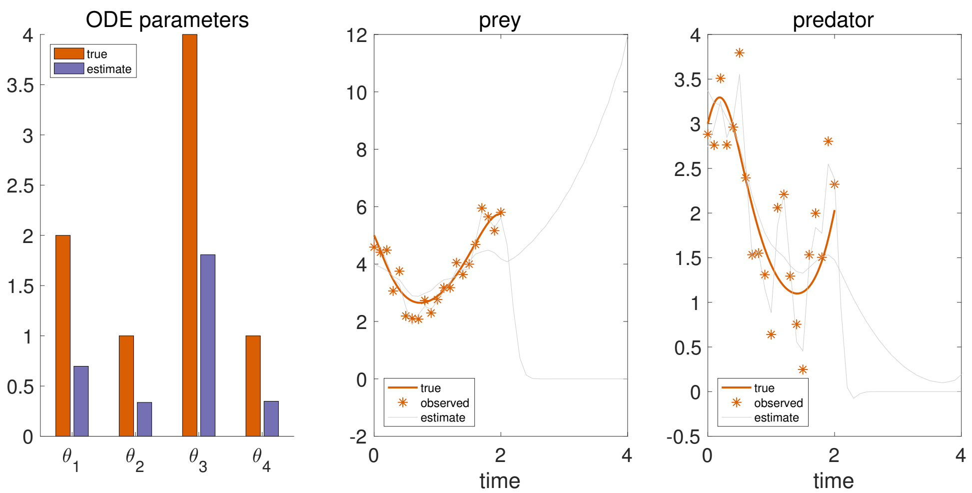
We minimize the KL-divergence in equation (10) by coordinate descent (where each step is analytically tractable) by iterating between determining the proxy for the distribution over ODE parameters $\hat{q}(\theta)$ and the proxies for the distribution over individual states $\hat{q}(\mathbf{x}_u)$.

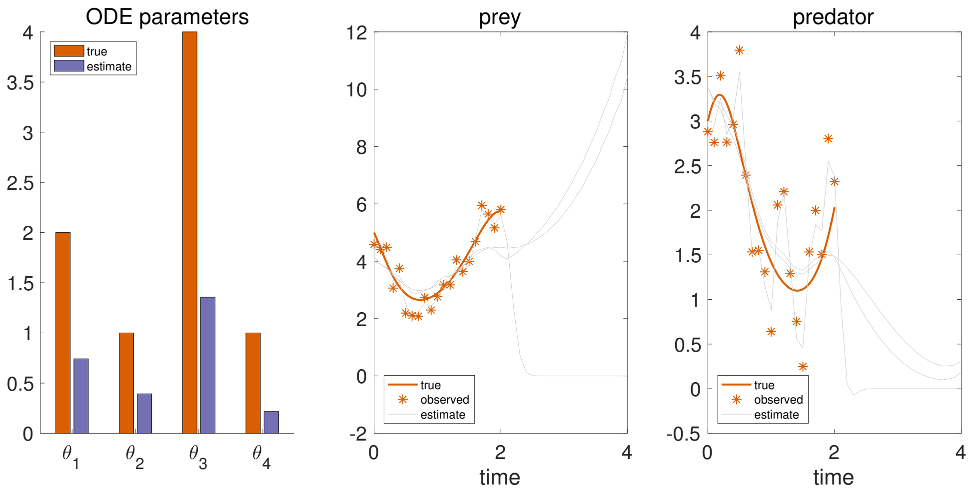
```
state.proxy.mean = mu;  % Initialize the state estimation by the GP regression posterior
for i = 1:opt_settings.coord_ascent_numb_iter
```

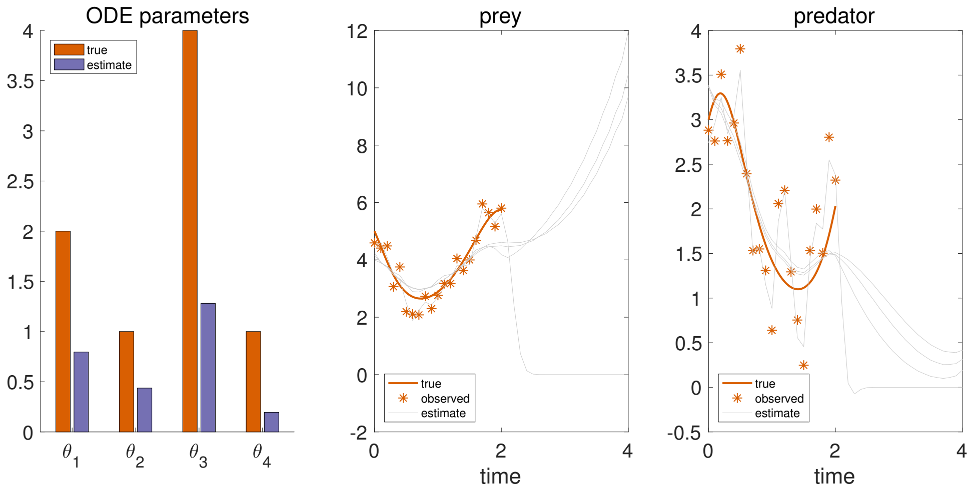
3.10.1 Proxy for ODE parameters

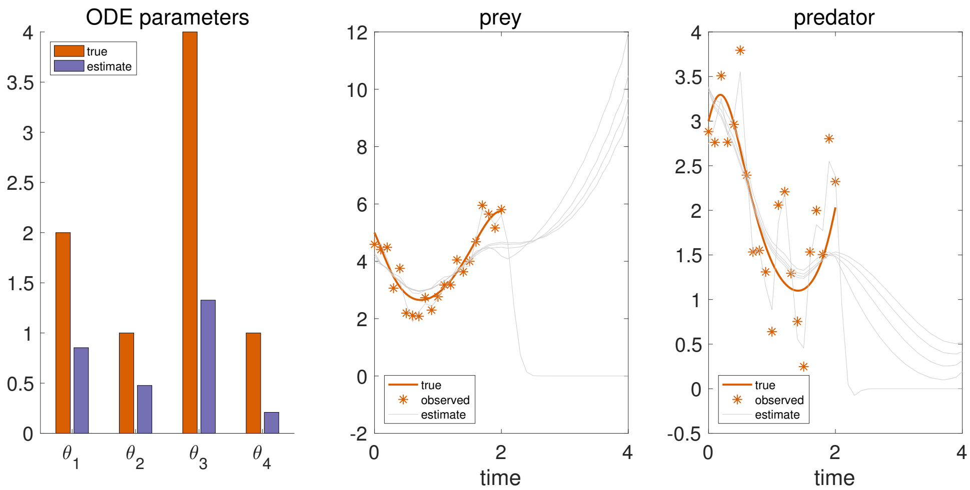
```
  [param_proxy_mean,param_proxy_inv_cov] = proxy_for_ode_parameters(state.proxy.mean,...
       dC_times_invC,ode_param.lin_comb,symbols,A_plus_gamma_inv,opt_settings);

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

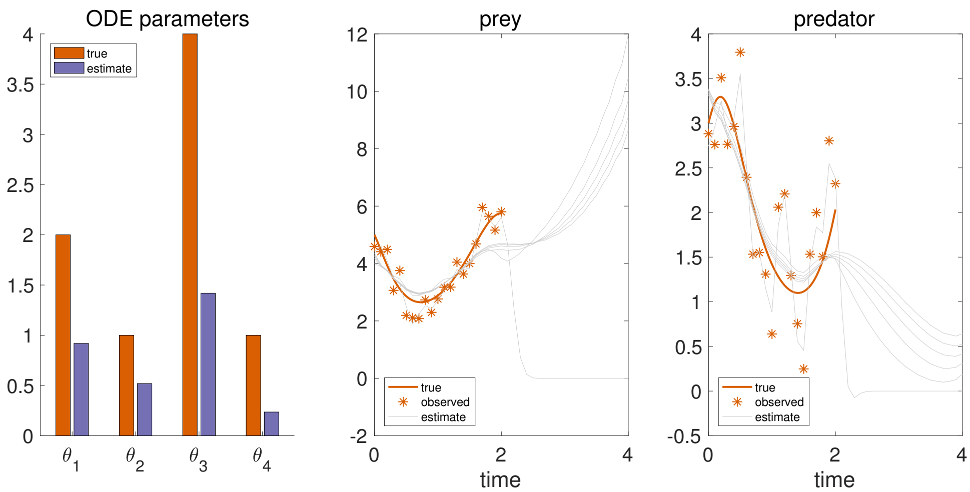

ODE parameters
true
estimate
θ1 θ2 θ3 θ4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

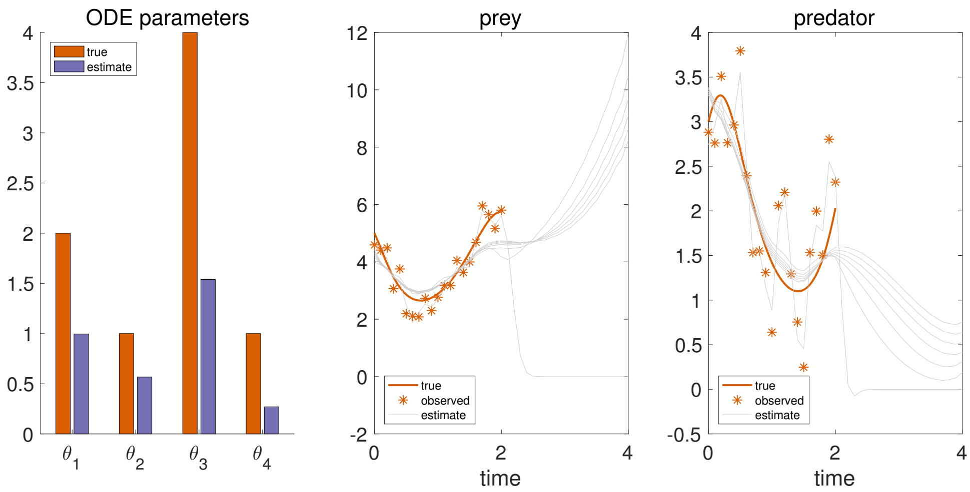

ODE parameters
true
estimate
θ1 θ2 θ3 θ4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

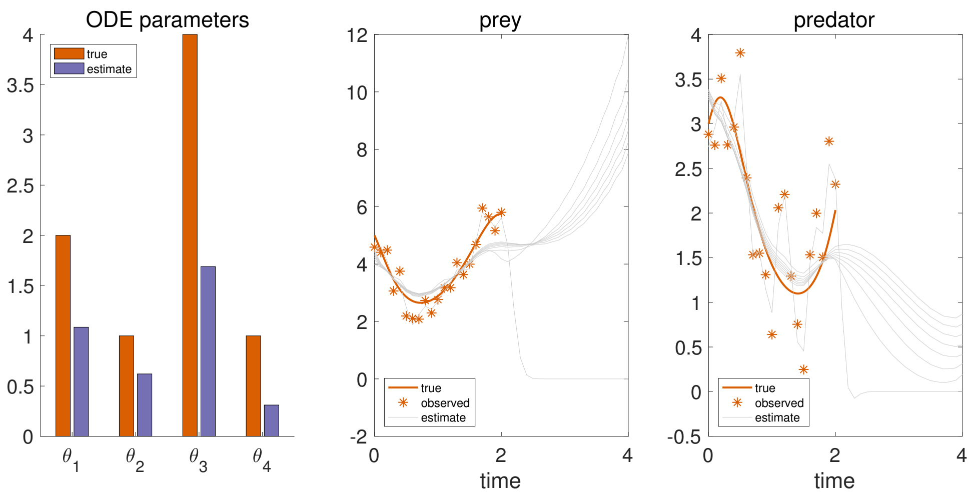

ODE parameters
true
estimate
θ1 θ2 θ3 θ4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

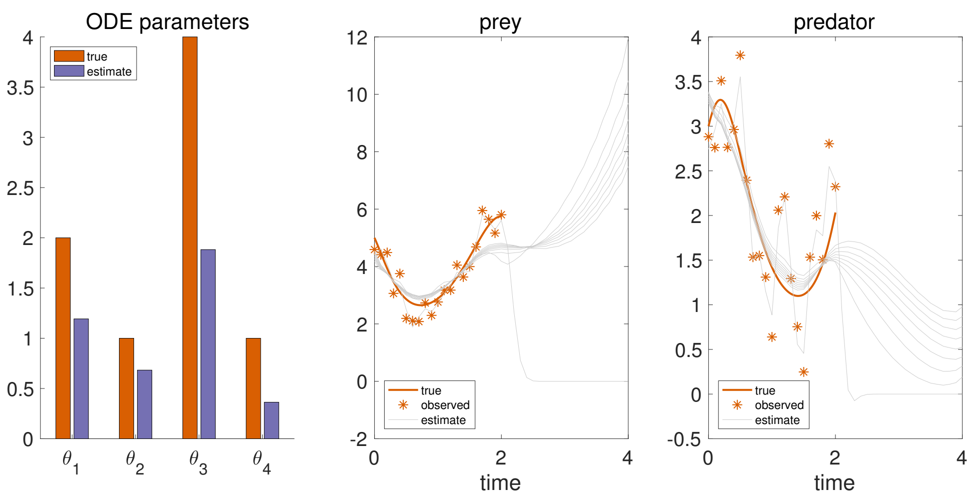

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

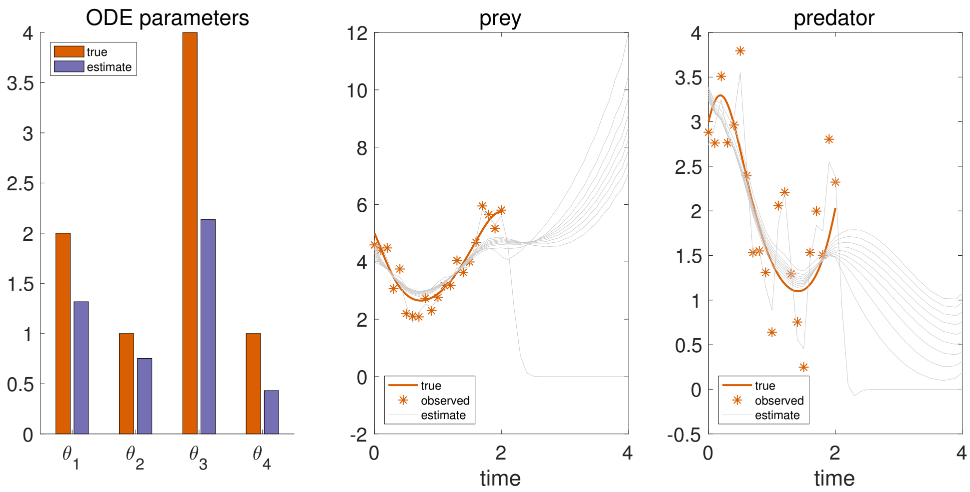

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

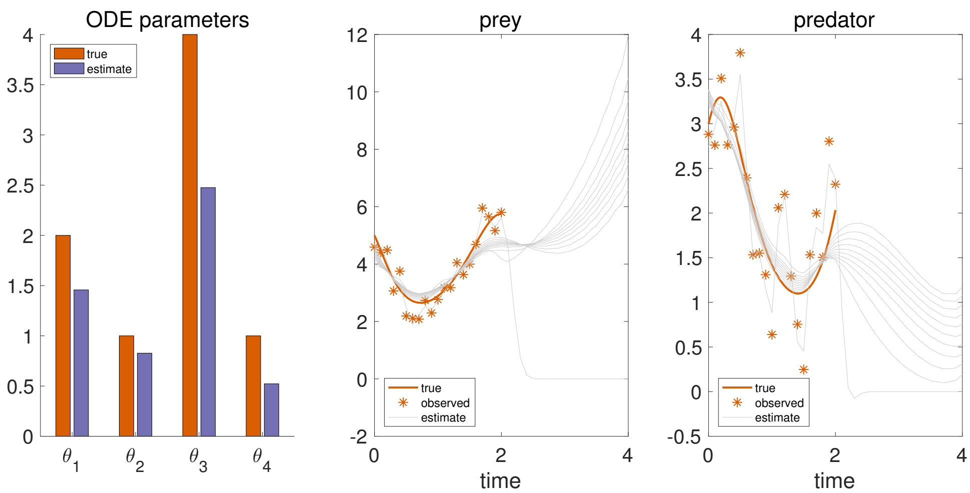

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

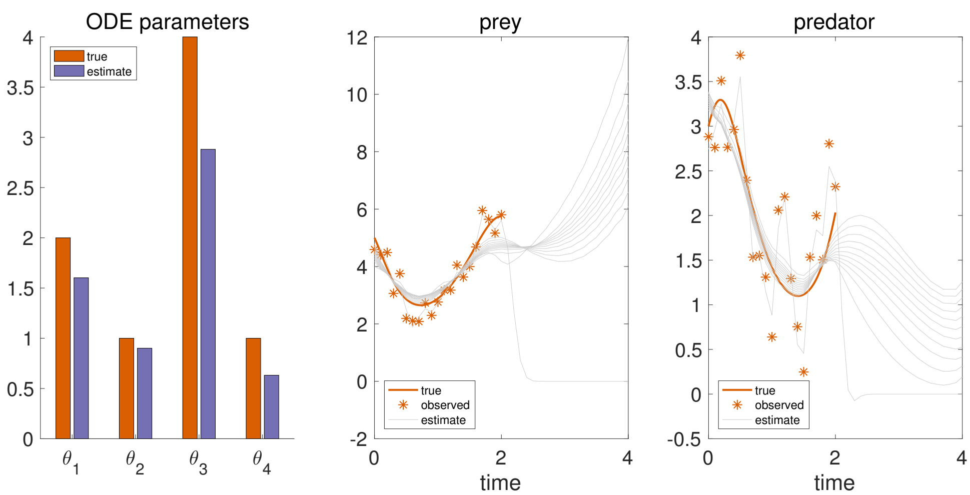

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

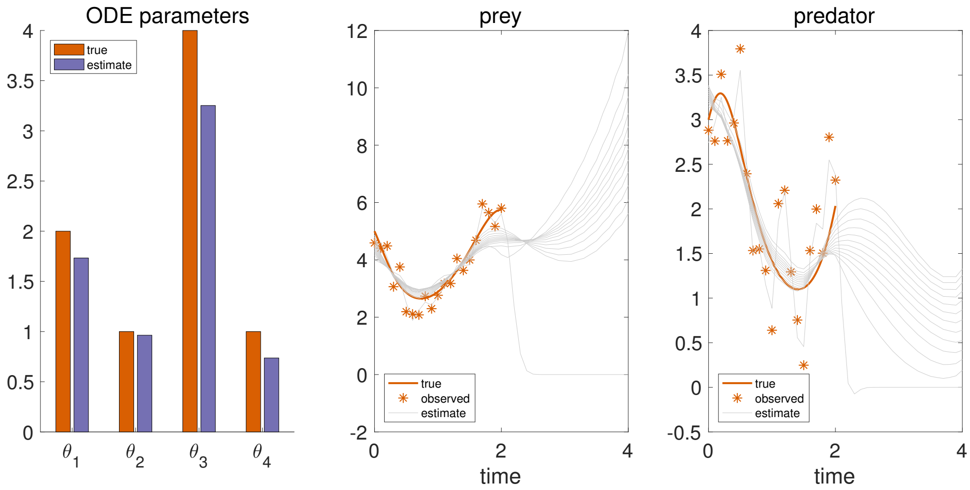

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

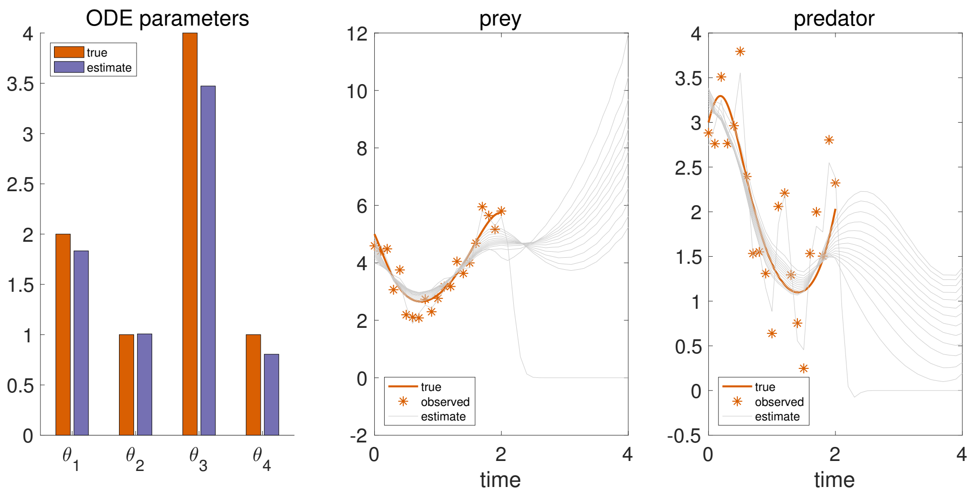

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

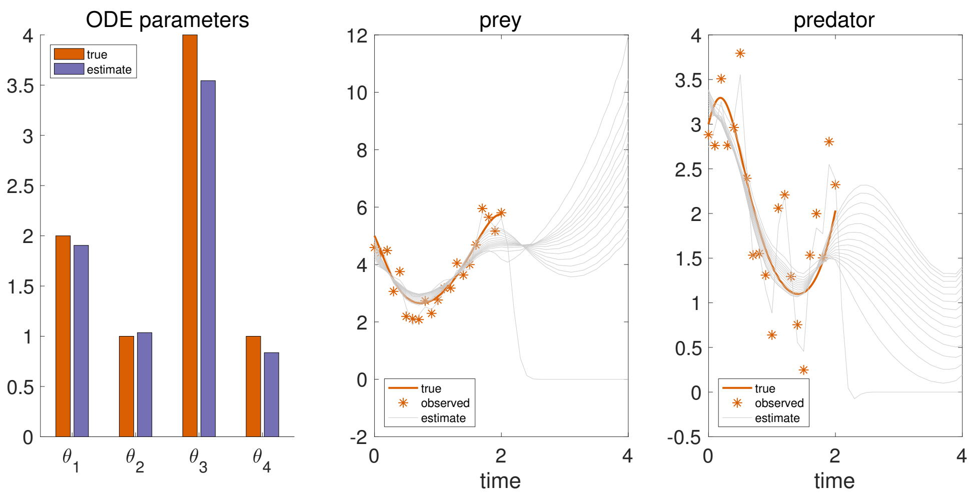

ODE parameters
true
estimate
θ1
θ2
θ3
θ4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

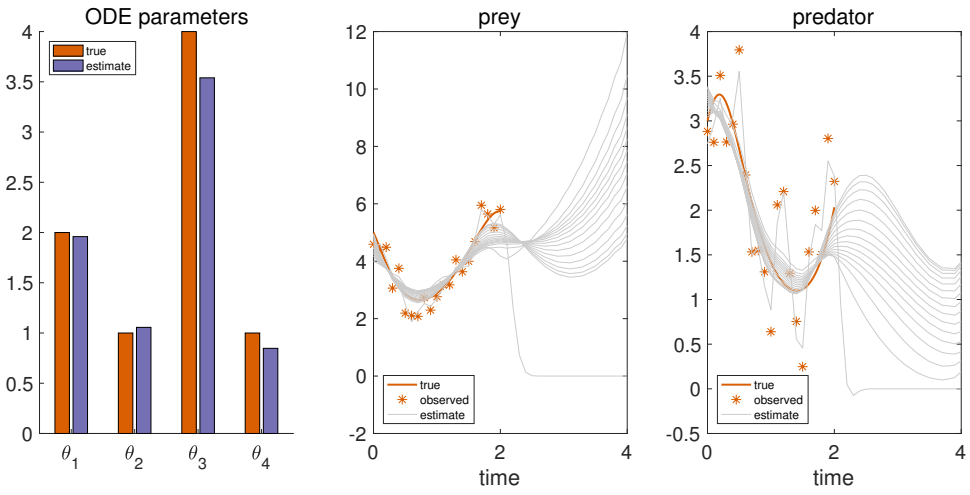

ODE parameters
true
estimate
θ1
θ2
θ3
θ4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

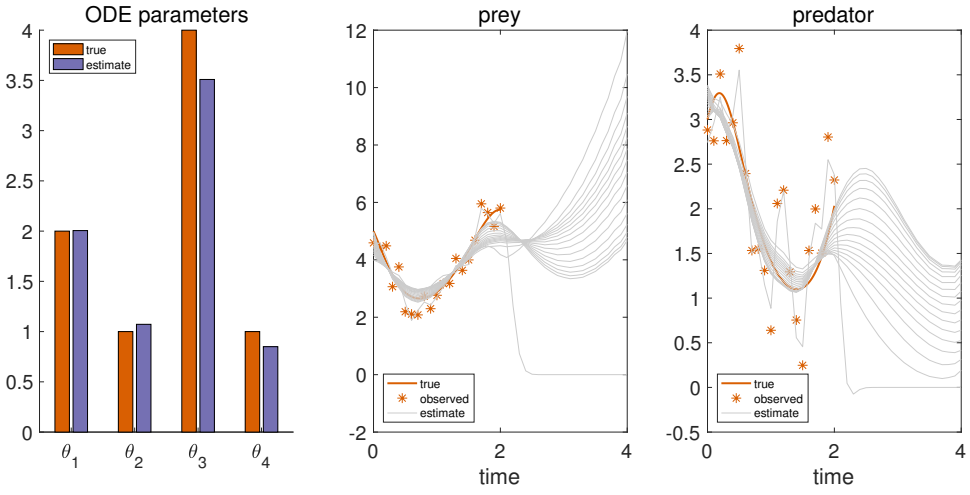

ODE parameters
true
estimate
θ1
θ2
θ3
θ4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

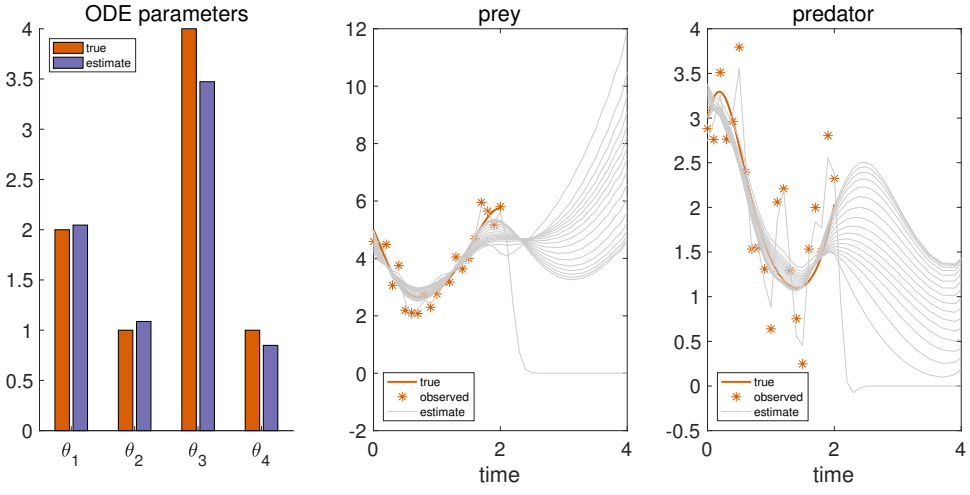

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

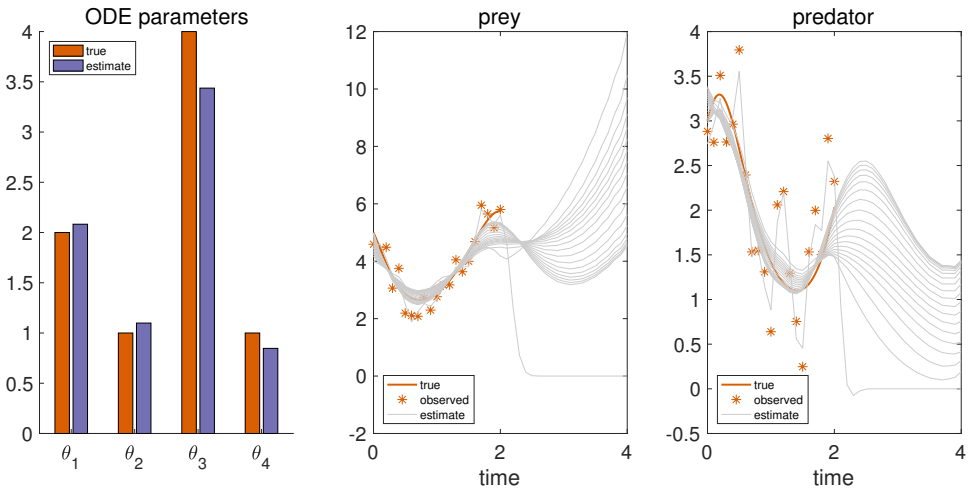

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

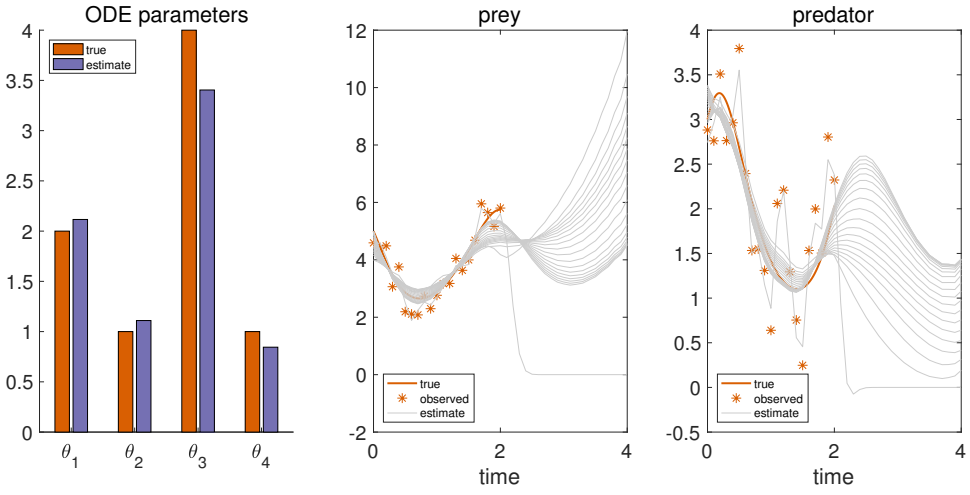

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

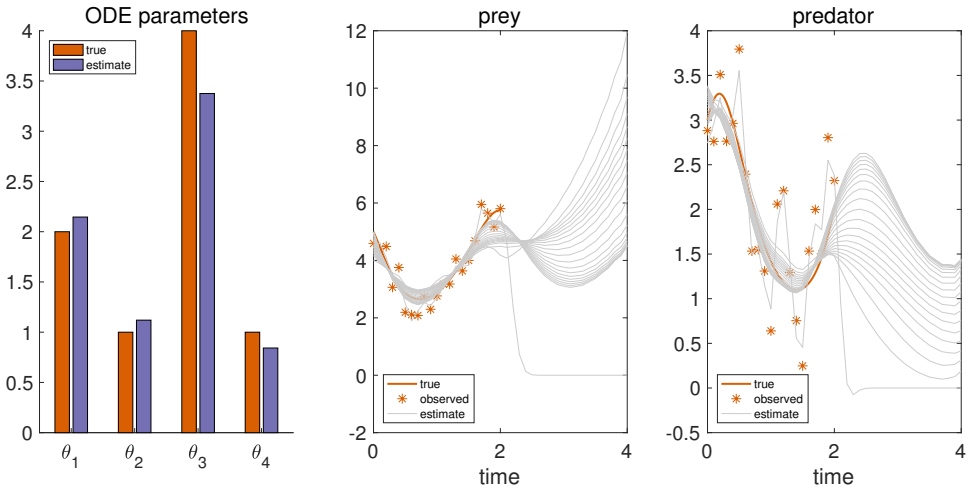

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

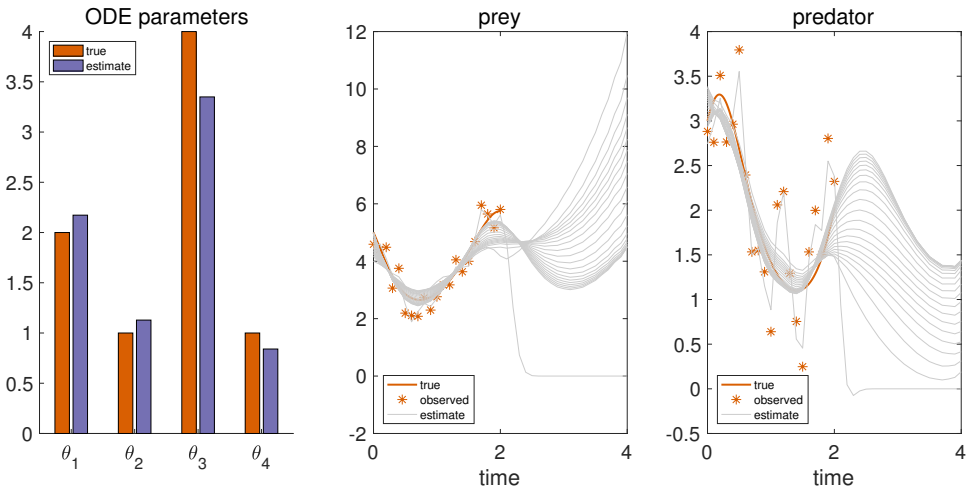

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

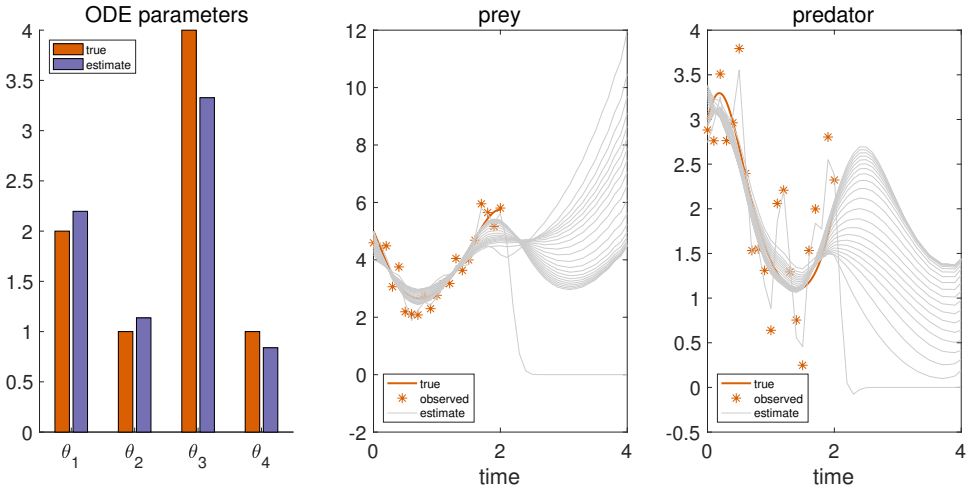

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

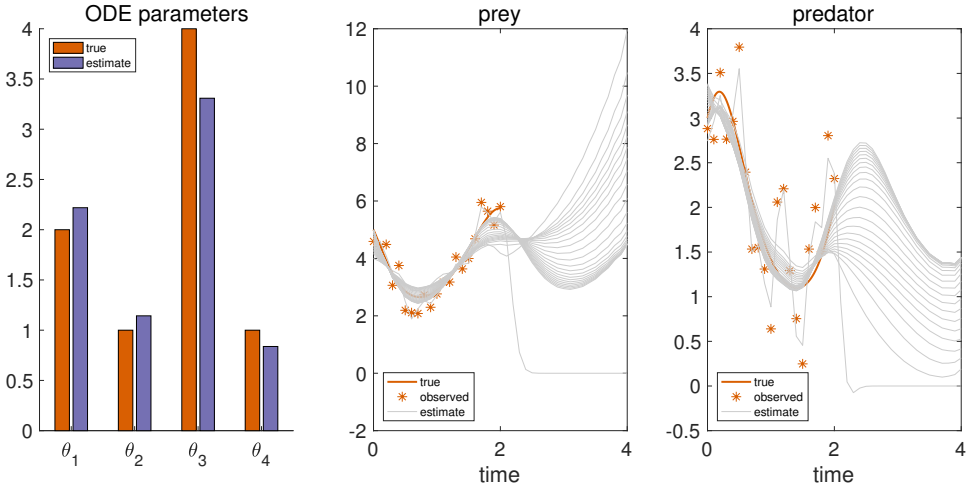

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

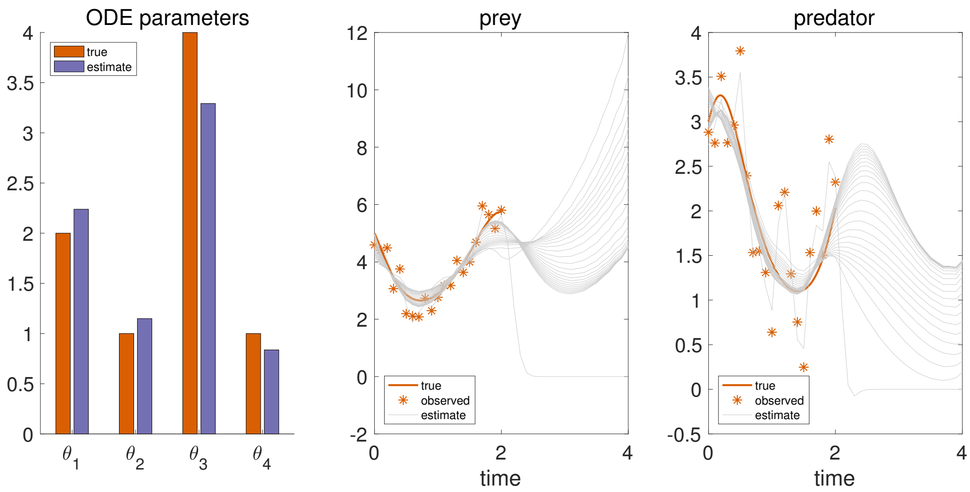

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

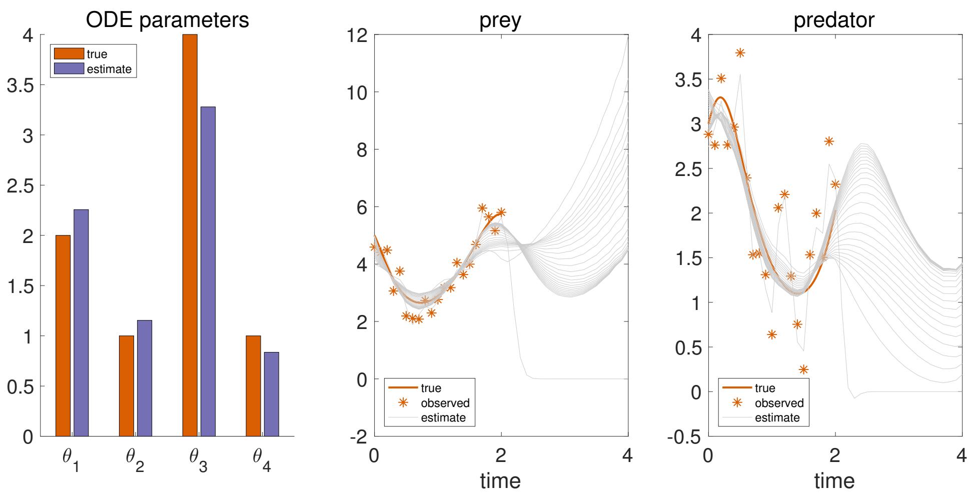

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

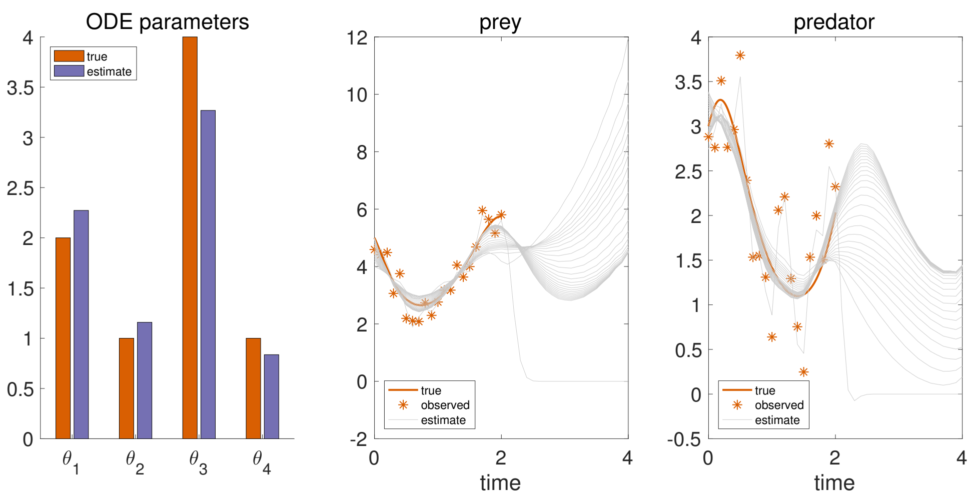

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

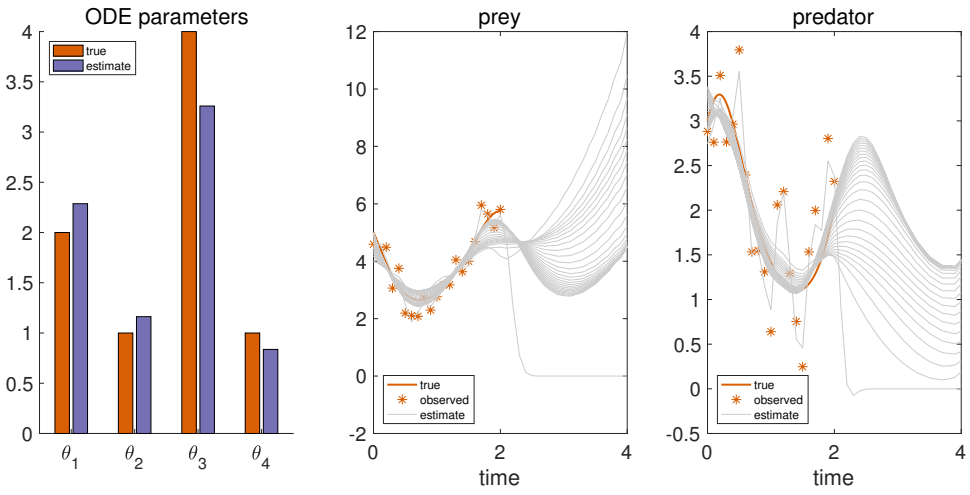

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

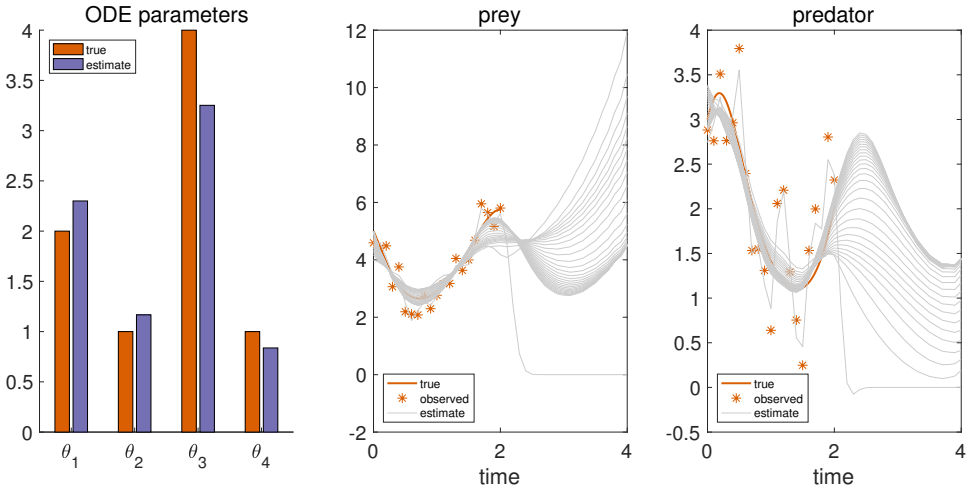

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

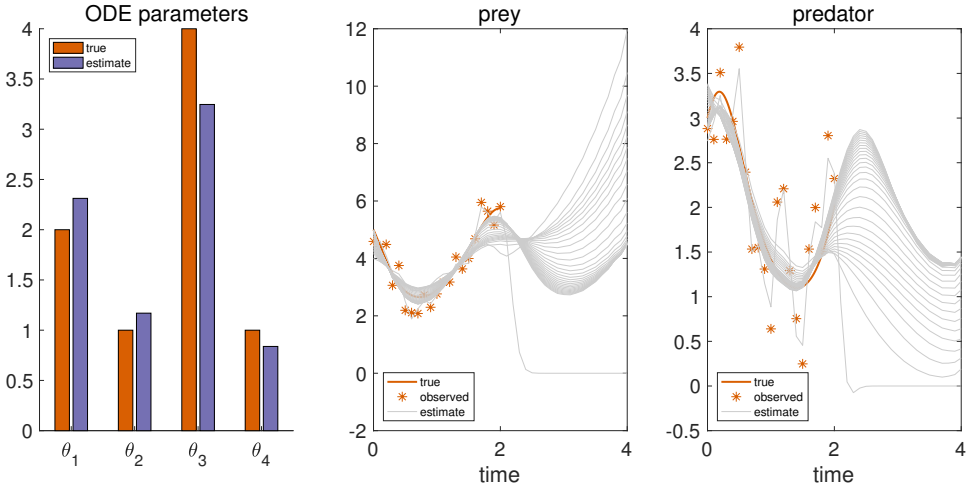

ODE parameters
true
estimate
θ1
θ2
θ3
θ4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

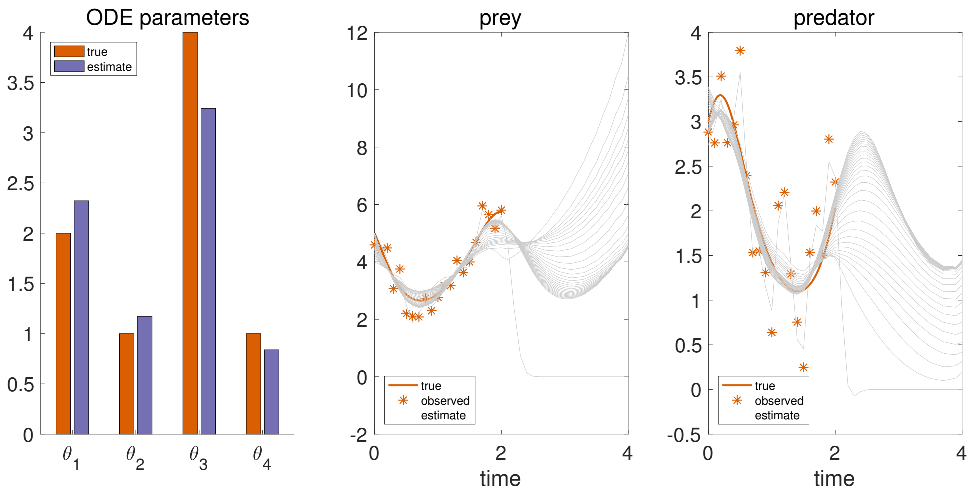

ODE parameters
true
estimate
θ1
θ2
θ3
θ4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

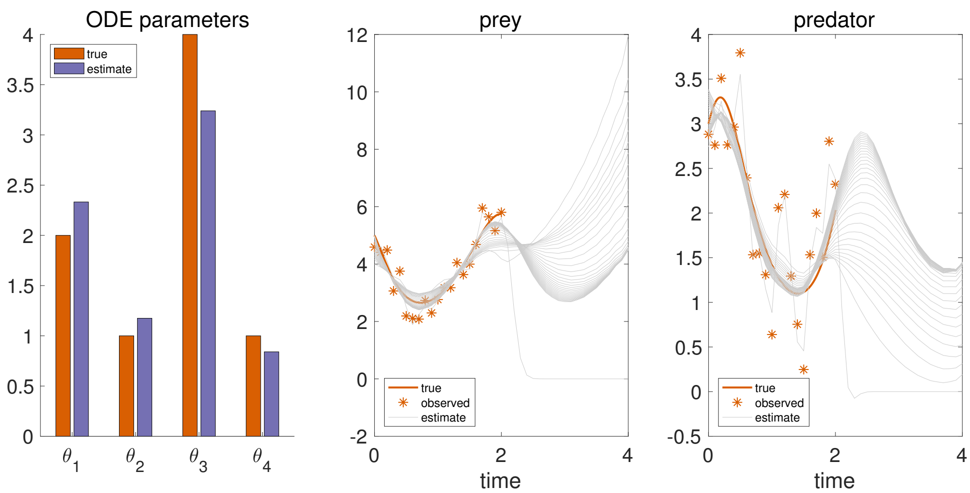

ODE parameters
true
estimate
θ1
θ2
θ3
θ4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

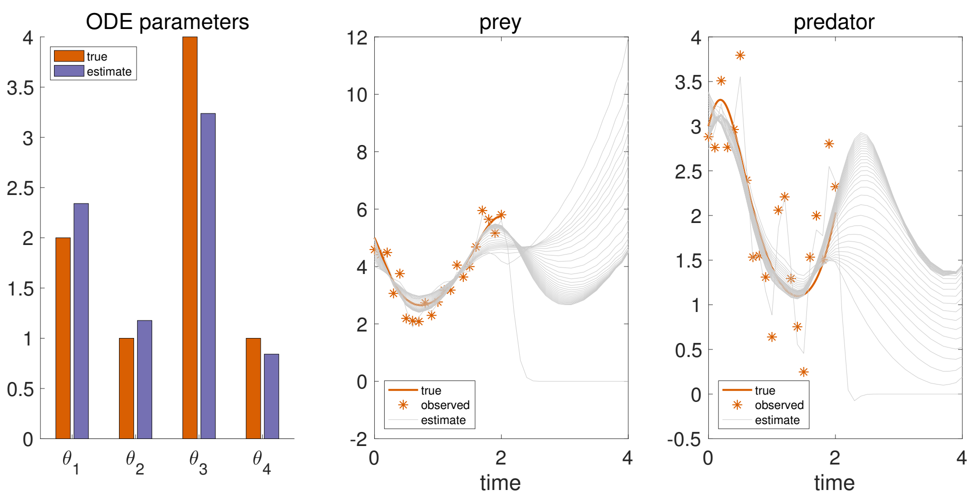

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

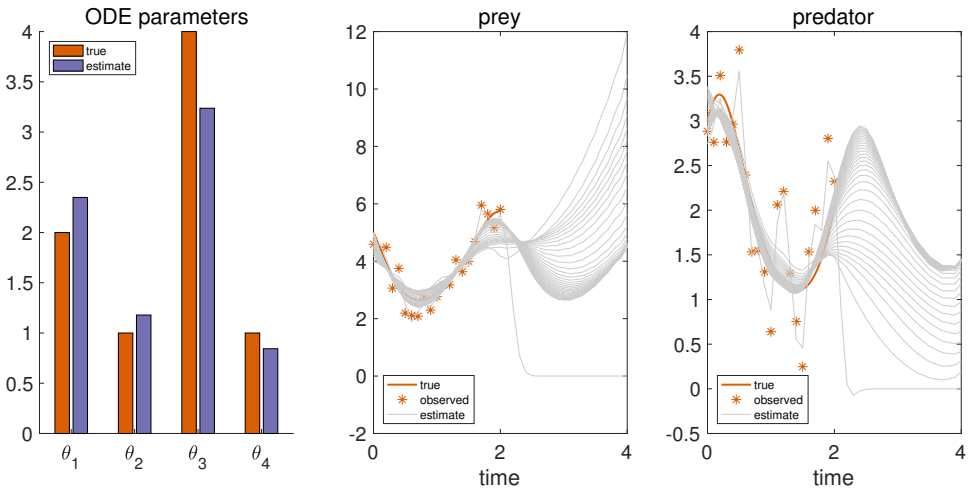

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

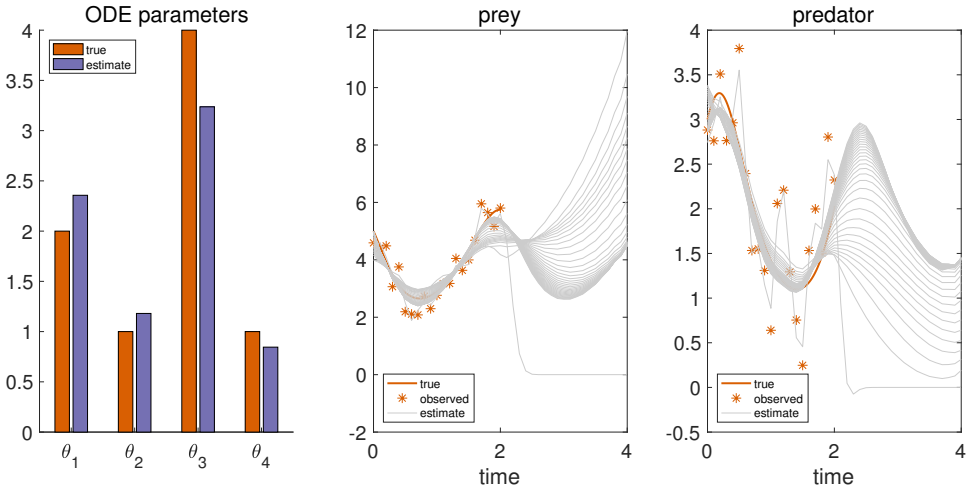

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

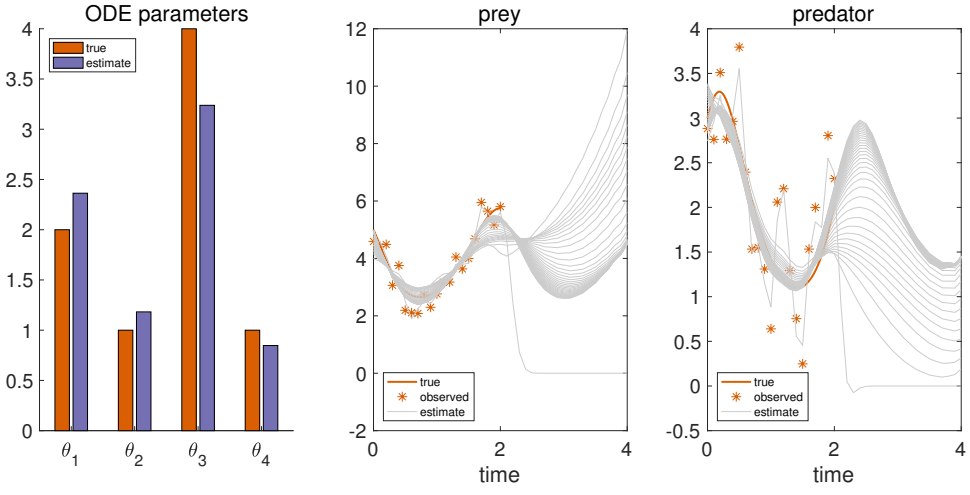

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

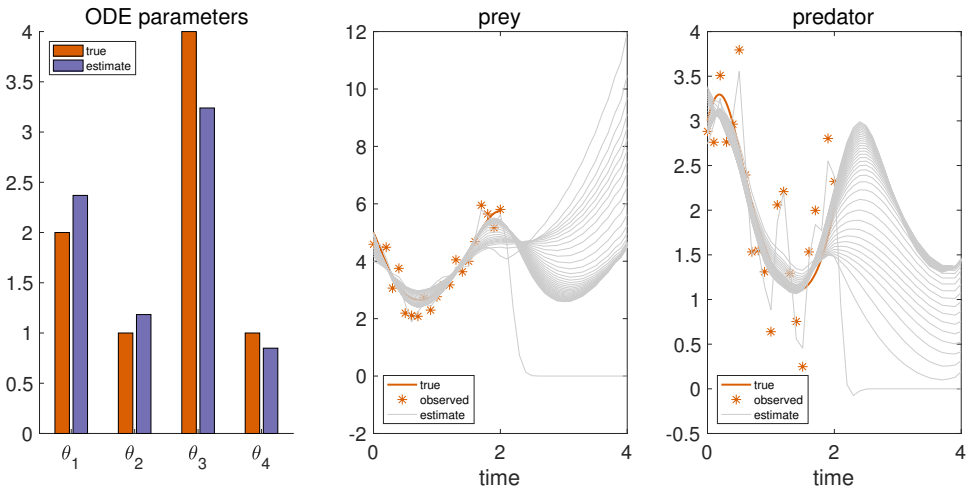

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

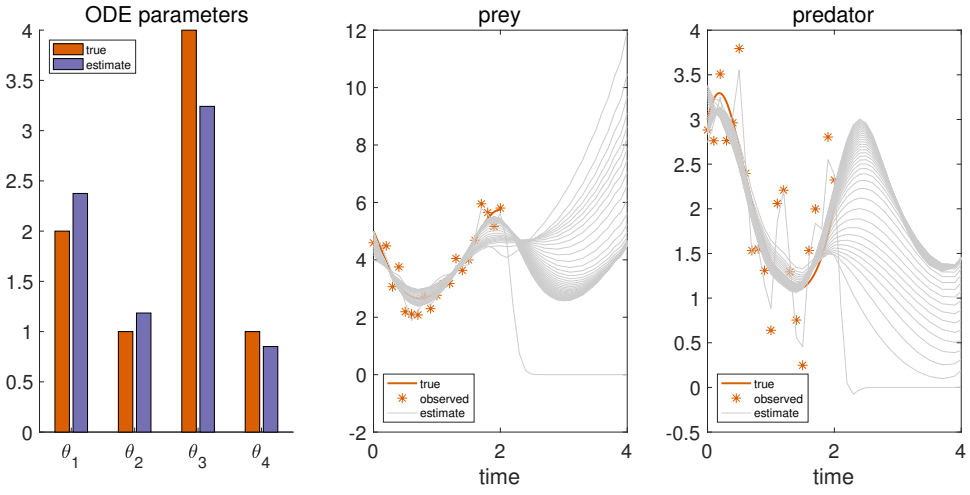

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
true
observed
estimate
time
predator
true
observed
estimate
time

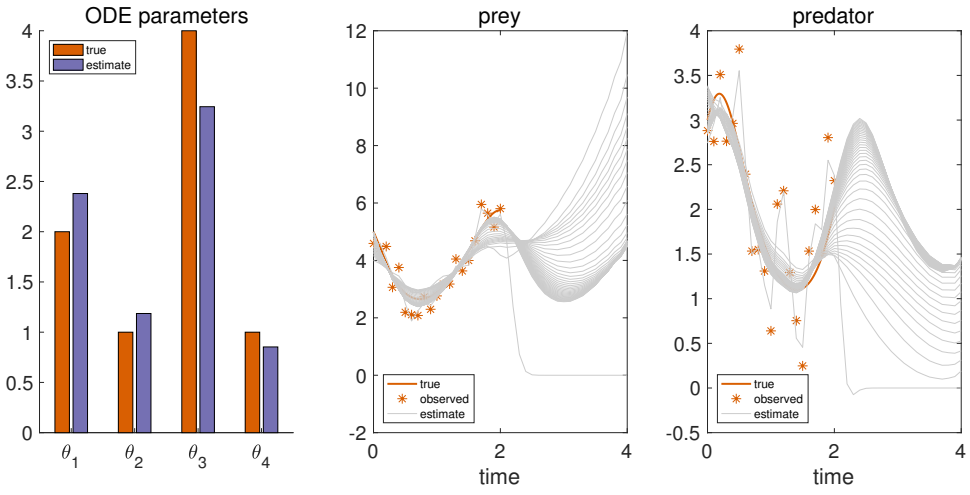

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time

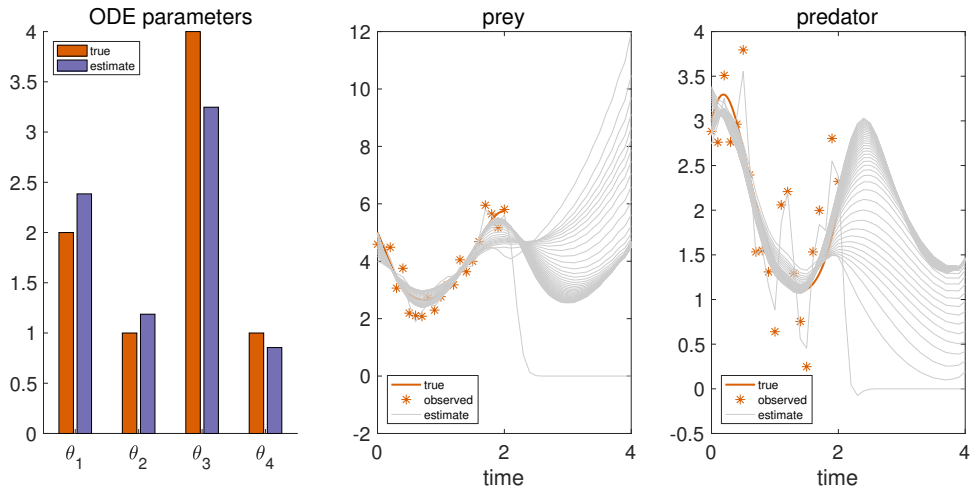

ODE parameters
true
estimate
θ_1 θ_2 θ_3 θ_4
prey
predator
true
observed
estimate
time


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


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


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


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



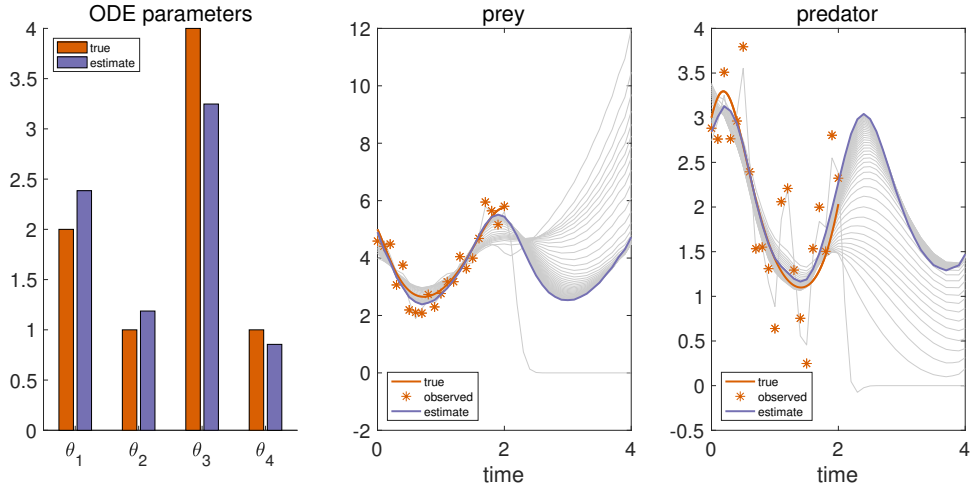
3.10.2 Proxy for individual states

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

```
end
```

3.10.3 Final result

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



3.11 Time Taken

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

```
time taken: 51.19 seconds
```

Chapter 4

VGM for Lorenz 96 with arbitrarily many ODEs

4.1 Simulation Settings

```
simulation.numb_odes = 100;          % number of ODEs
% observation noise
simulation.state_obs_variance = @(mean)(bsxfun(@times,0.1,ones(size(mean))));
simulation.ode_param = 8;            % true ODE parameter;
simulation.final_time = 4;           % end time for integration
simulation.int_interval = 0.01;      % integration interval
simulation.time_samp = 0:0.125:simulation.final_time; % sample times for observations
simulation.init_val = zeros(1,simulation.numb_odes); % initial state values

% ratio of number of observed states over number of unobserved states.
simulation.observed_states = 0.5;    % only 50% of the states are observed
```

4.2 User Input

4.2.1 Path to ODEs

```
odes_path = 'Lorenz96_ODEs.txt';
```

4.2.2 Symbols

symbols of states and parameters in the '_ODEs.txt' file

States **x**:

```
for u = 1:simulation.numb_odes; symbols.state(u) = {['[x_' num2str(u) ']']}; end
```

ODE parameters θ (symbols of parameters in 'ODEs.txt' file):

```
symbols.param = {'[\theta]'};
% \section{ Kernel }
%
% Kernel parameters $\phi$:
kernel.param = [10,0.2];             % set values of rbf kernel parameters
```

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

```
state.derivative_variance = 6*ones(1,length(symbols.state)); % gamma for gradient matching model
```

4.2.3 Estimation times

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

4.2.4 Type of pseudo-inverse

Type of pseudo inverse; options: 'Moore-Penrose' or 'modified Moore-Penrose'

```
opt_settings.pseudo_inv_type = 'Moore-Penrose';
```

4.2.5 Optimization settings

```
opt_settings.coord_ascent_numb_iter = 10;  % number of coordinate ascent iterations

% The observed state trajectories are clamped to the trajectories
% determined by standard GP regression (Boolean)
opt_settings.clamp_obs_state_to_GP_fit = true;
```

Plot settings: layout and size

```
plot_settings.size = [1600, 800]; plot_settings.layout = [3,3];
```

4.3 Import ODEs

```
generate_Lorenz96_ODEs(simulation.numb_odes)
ode = import_odes(symbols,odes_path);


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

```
ODEs:
    '([x_2] - [x_99]) .* [x_100] - [x_1] + [\theta]'
    '([x_3] - [x_100]) .* [x_1] - [x_2] + [\theta]'
    '([x_4] - [x_1]) .* [x_2] - [x_3] + [\theta]'
    '([x_5] - [x_2]) .* [x_3] - [x_4] + [\theta]'
    '([x_6] - [x_3]) .* [x_4] - [x_5] + [\theta]'
    '([x_7] - [x_4]) .* [x_5] - [x_6] + [\theta]'
    '([x_8] - [x_5]) .* [x_6] - [x_7] + [\theta]'
    '([x_9] - [x_6]) .* [x_7] - [x_8] + [\theta]'
    '([x_10] - [x_7]) .* [x_8] - [x_9] + [\theta]'
    '([x_11] - [x_8]) .* [x_9] - [x_10] + [\theta]'
    '([x_12] - [x_9]) .* [x_10] - [x_11] + [\theta]'
```

```
'([x_13] - [x_10]) .* [x_11] - [x_12] + [\theta]'
'([x_14] - [x_11]) .* [x_12] - [x_13] + [\theta]'
'([x_15] - [x_12]) .* [x_13] - [x_14] + [\theta]'
'([x_16] - [x_13]) .* [x_14] - [x_15] + [\theta]'
'([x_17] - [x_14]) .* [x_15] - [x_16] + [\theta]'
'([x_18] - [x_15]) .* [x_16] - [x_17] + [\theta]'
'([x_19] - [x_16]) .* [x_17] - [x_18] + [\theta]'
'([x_20] - [x_17]) .* [x_18] - [x_19] + [\theta]'
'([x_21] - [x_18]) .* [x_19] - [x_20] + [\theta]'
'([x_22] - [x_19]) .* [x_20] - [x_21] + [\theta]'
'([x_23] - [x_20]) .* [x_21] - [x_22] + [\theta]'
'([x_24] - [x_21]) .* [x_22] - [x_23] + [\theta]'
'([x_25] - [x_22]) .* [x_23] - [x_24] + [\theta]'
'([x_26] - [x_23]) .* [x_24] - [x_25] + [\theta]'
'([x_27] - [x_24]) .* [x_25] - [x_26] + [\theta]'
'([x_28] - [x_25]) .* [x_26] - [x_27] + [\theta]'
'([x_29] - [x_26]) .* [x_27] - [x_28] + [\theta]'
'([x_30] - [x_27]) .* [x_28] - [x_29] + [\theta]'
'([x_31] - [x_28]) .* [x_29] - [x_30] + [\theta]'
'([x_32] - [x_29]) .* [x_30] - [x_31] + [\theta]'
'([x_33] - [x_30]) .* [x_31] - [x_32] + [\theta]'
'([x_34] - [x_31]) .* [x_32] - [x_33] + [\theta]'
'([x_35] - [x_32]) .* [x_33] - [x_34] + [\theta]'
'([x_36] - [x_33]) .* [x_34] - [x_35] + [\theta]'
'([x_37] - [x_34]) .* [x_35] - [x_36] + [\theta]'
'([x_38] - [x_35]) .* [x_36] - [x_37] + [\theta]'
'([x_39] - [x_36]) .* [x_37] - [x_38] + [\theta]'
'([x_40] - [x_37]) .* [x_38] - [x_39] + [\theta]'
'([x_41] - [x_38]) .* [x_39] - [x_40] + [\theta]'
'([x_42] - [x_39]) .* [x_40] - [x_41] + [\theta]'
'([x_43] - [x_40]) .* [x_41] - [x_42] + [\theta]'
'([x_44] - [x_41]) .* [x_42] - [x_43] + [\theta]'
'([x_45] - [x_42]) .* [x_43] - [x_44] + [\theta]'
'([x_46] - [x_43]) .* [x_44] - [x_45] + [\theta]'
'([x_47] - [x_44]) .* [x_45] - [x_46] + [\theta]'
'([x_48] - [x_45]) .* [x_46] - [x_47] + [\theta]'
'([x_49] - [x_46]) .* [x_47] - [x_48] + [\theta]'
'([x_50] - [x_47]) .* [x_48] - [x_49] + [\theta]'
'([x_51] - [x_48]) .* [x_49] - [x_50] + [\theta]'
'([x_52] - [x_49]) .* [x_50] - [x_51] + [\theta]'
'([x_53] - [x_50]) .* [x_51] - [x_52] + [\theta]'
'([x_54] - [x_51]) .* [x_52] - [x_53] + [\theta]'
'([x_55] - [x_52]) .* [x_53] - [x_54] + [\theta]'
'([x_56] - [x_53]) .* [x_54] - [x_55] + [\theta]'
'([x_57] - [x_54]) .* [x_55] - [x_56] + [\theta]'
'([x_58] - [x_55]) .* [x_56] - [x_57] + [\theta]'
'([x_59] - [x_56]) .* [x_57] - [x_58] + [\theta]'
'([x_60] - [x_57]) .* [x_58] - [x_59] + [\theta]'
'([x_61] - [x_58]) .* [x_59] - [x_60] + [\theta]'
'([x_62] - [x_59]) .* [x_60] - [x_61] + [\theta]'
'([x_63] - [x_60]) .* [x_61] - [x_62] + [\theta]'
```

```
    '([x_64] - [x_61]) .* [x_62] - [x_63] + [\theta]'
    '([x_65] - [x_62]) .* [x_63] - [x_64] + [\theta]'
    '([x_66] - [x_63]) .* [x_64] - [x_65] + [\theta]'
    '([x_67] - [x_64]) .* [x_65] - [x_66] + [\theta]'
    '([x_68] - [x_65]) .* [x_66] - [x_67] + [\theta]'
    '([x_69] - [x_66]) .* [x_67] - [x_68] + [\theta]'
    '([x_70] - [x_67]) .* [x_68] - [x_69] + [\theta]'
    '([x_71] - [x_68]) .* [x_69] - [x_70] + [\theta]'
    '([x_72] - [x_69]) .* [x_70] - [x_71] + [\theta]'
    '([x_73] - [x_70]) .* [x_71] - [x_72] + [\theta]'
    '([x_74] - [x_71]) .* [x_72] - [x_73] + [\theta]'
    '([x_75] - [x_72]) .* [x_73] - [x_74] + [\theta]'
    '([x_76] - [x_73]) .* [x_74] - [x_75] + [\theta]'
    '([x_77] - [x_74]) .* [x_75] - [x_76] + [\theta]'
    '([x_78] - [x_75]) .* [x_76] - [x_77] + [\theta]'
    '([x_79] - [x_76]) .* [x_77] - [x_78] + [\theta]'
    '([x_80] - [x_77]) .* [x_78] - [x_79] + [\theta]'
    '([x_81] - [x_78]) .* [x_79] - [x_80] + [\theta]'
    '([x_82] - [x_79]) .* [x_80] - [x_81] + [\theta]'
    '([x_83] - [x_80]) .* [x_81] - [x_82] + [\theta]'
    '([x_84] - [x_81]) .* [x_82] - [x_83] + [\theta]'
    '([x_85] - [x_82]) .* [x_83] - [x_84] + [\theta]'
    '([x_86] - [x_83]) .* [x_84] - [x_85] + [\theta]'
    '([x_87] - [x_84]) .* [x_85] - [x_86] + [\theta]'
    '([x_88] - [x_85]) .* [x_86] - [x_87] + [\theta]'
    '([x_89] - [x_86]) .* [x_87] - [x_88] + [\theta]'
    '([x_90] - [x_87]) .* [x_88] - [x_89] + [\theta]'
    '([x_91] - [x_88]) .* [x_89] - [x_90] + [\theta]'
    '([x_92] - [x_89]) .* [x_90] - [x_91] + [\theta]'
    '([x_93] - [x_90]) .* [x_91] - [x_92] + [\theta]'
    '([x_94] - [x_91]) .* [x_92] - [x_93] + [\theta]'
    '([x_95] - [x_92]) .* [x_93] - [x_94] + [\theta]'
    '([x_96] - [x_93]) .* [x_94] - [x_95] + [\theta]'
    '([x_97] - [x_94]) .* [x_95] - [x_96] + [\theta]'
    '([x_98] - [x_95]) .* [x_96] - [x_97] + [\theta]'
    '([x_99] - [x_96]) .* [x_97] - [x_98] + [\theta]'
    '([x_100] - [x_97]) .* [x_98] - [x_99] + [\theta]'
    '([x_1] - [x_98]) .* [x_99] - [x_100] + [\theta]'
```

4.4 Simulate Trajectory Observations

4.4.1 Generate ground truth by numerical integration

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

4.4.2 Generate state observations

```
if ~iscell(simulation.observed_states)
    ratio_observed = simulation.observed_states;
    state_obs_idx = zeros(1,simulation.numb_odes,'logical');
    idx = randperm(simulation.numb_odes);
    idx = idx(1:floor(simulation.numb_odes * ratio_observed));
    state_obs_idx(idx) = 1;
    simulation.observed_states = symbols.state(state_obs_idx);
end

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

4.4.3 Symbols

```
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');
```

4.4.4 Setup plots

Only the state dynamics are (partially) observed.

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

tic; %start timer

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

4.5 State Couplings in ODEs

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

4.6 Prior over State and State Derivatives

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

4.7 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(ode,symbols);
```

4.8 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,symbols,coupling_idx.states);
```

4.9 Fitting observations of state trajectories

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

4.10 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 $\hat{q}(\mathbf{x}_u)$.

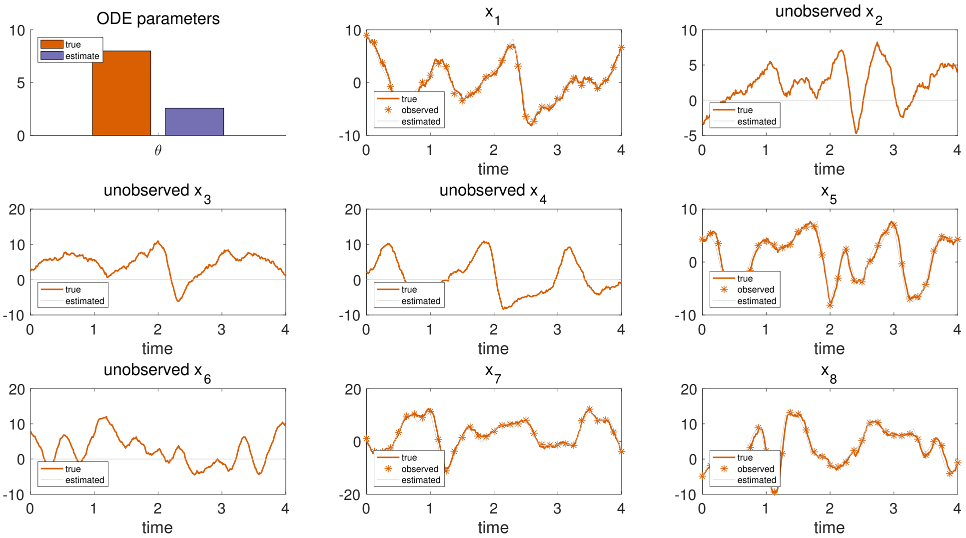
```
state.proxy.mean = mu;  % Initialize the state estimation by the GP regression posterior
for i = 1:opt_settings.coord_ascent_numb_iter
```

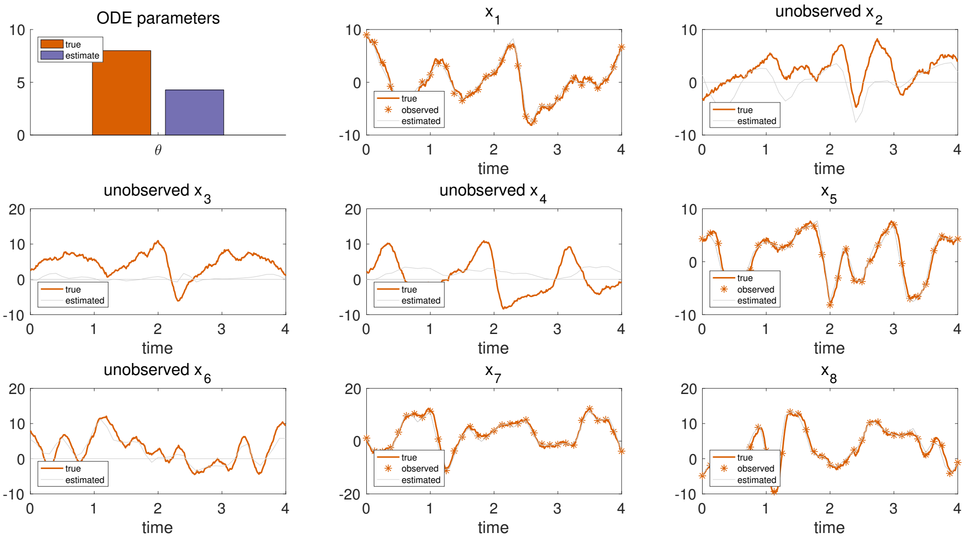
4.10.1 Proxy for ODE parameters

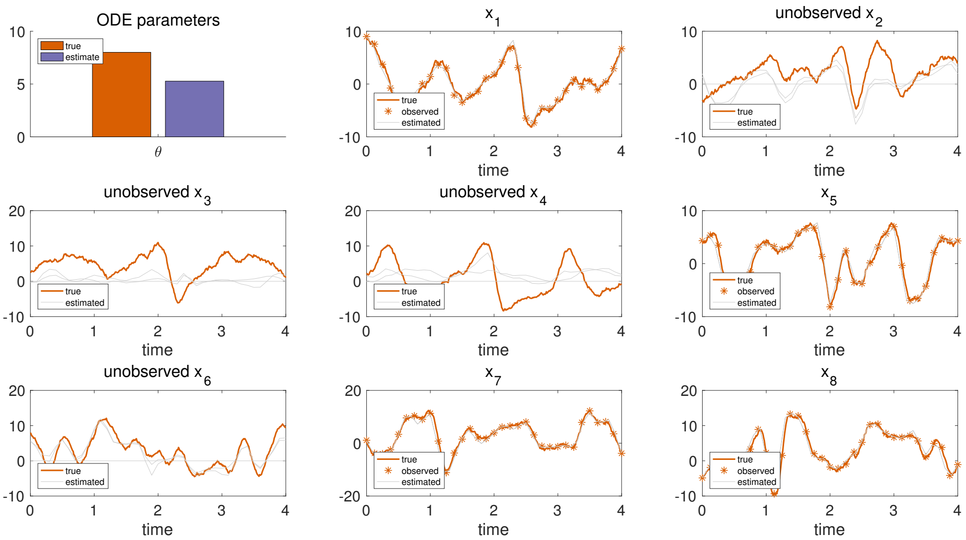
```
  [param_proxy_mean,param_proxy_inv_cov] = proxy_for_ode_parameters(state.proxy.mean,...
      dC_times_invC,ode_param.lin_comb,symbols,A_plus_gamma_inv,opt_settings);

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

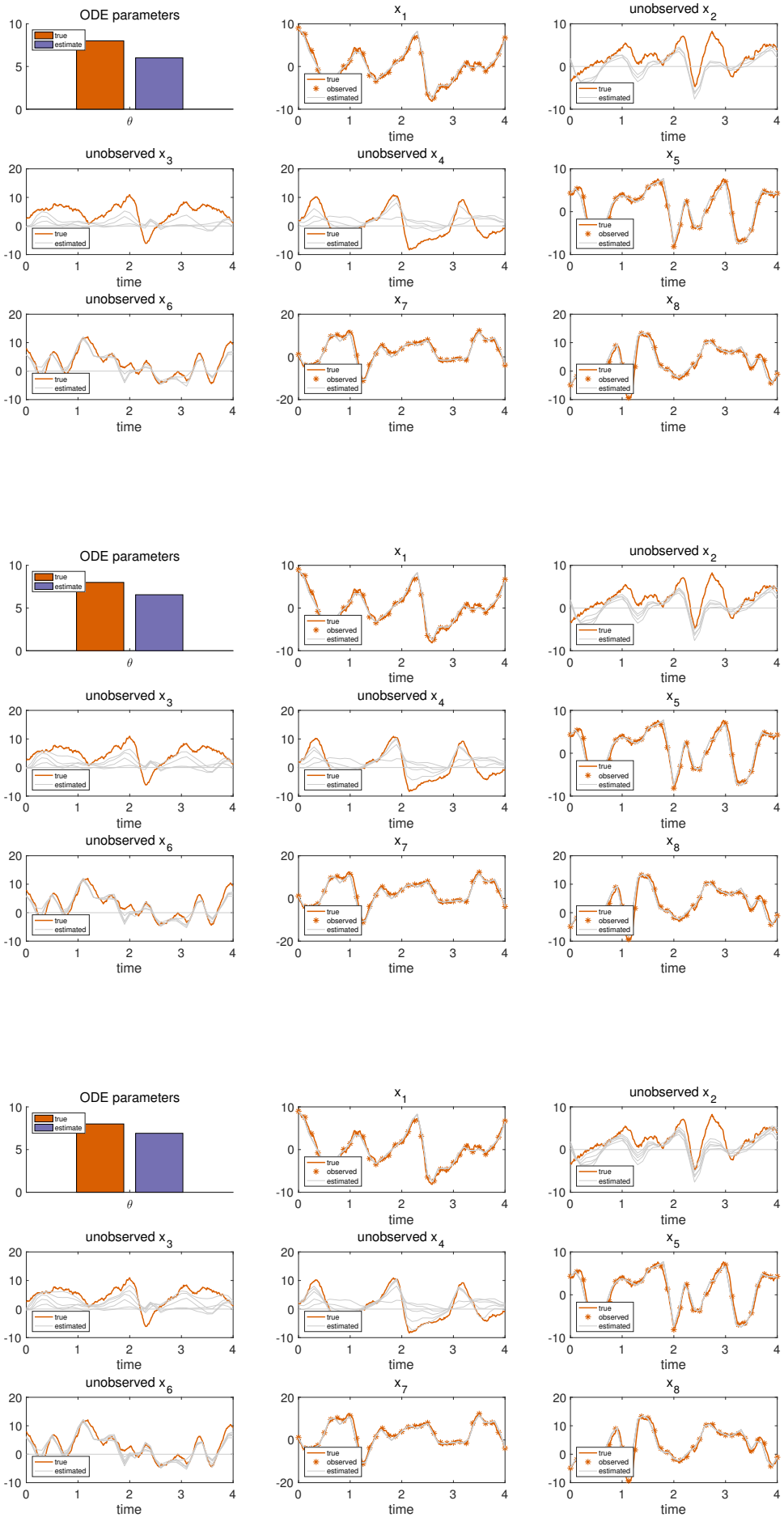
4.10.2 Proxy for individual states

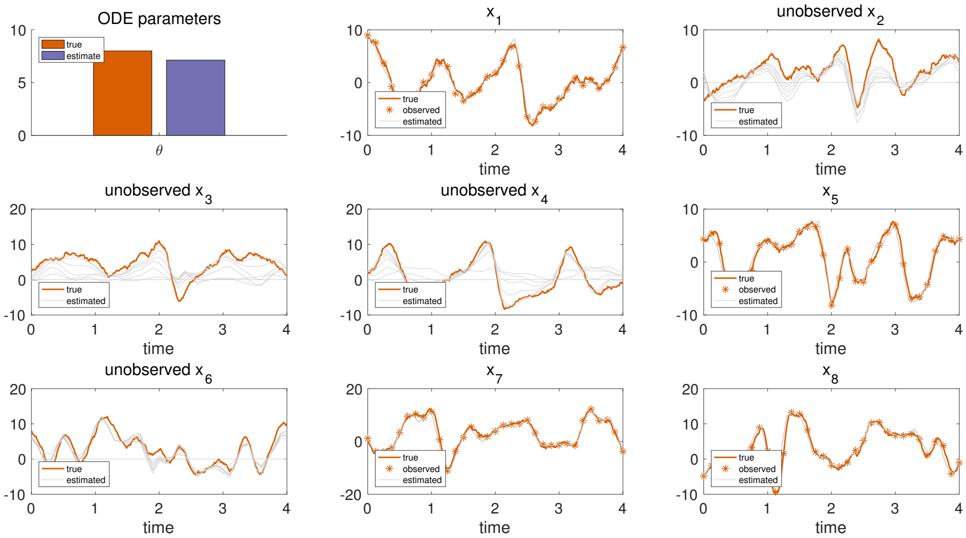
```
    [state.proxy.mean,state.proxy.inv_cov] = proxy_for_ind_states(state.lin_comb,...
     state.proxy.mean,param_proxy_mean',dC_times_invC,coupling_idx.states,symbols,...
       mu,inv_sigma,simulation.observed_states,A_plus_gamma_inv,opt_settings);
```

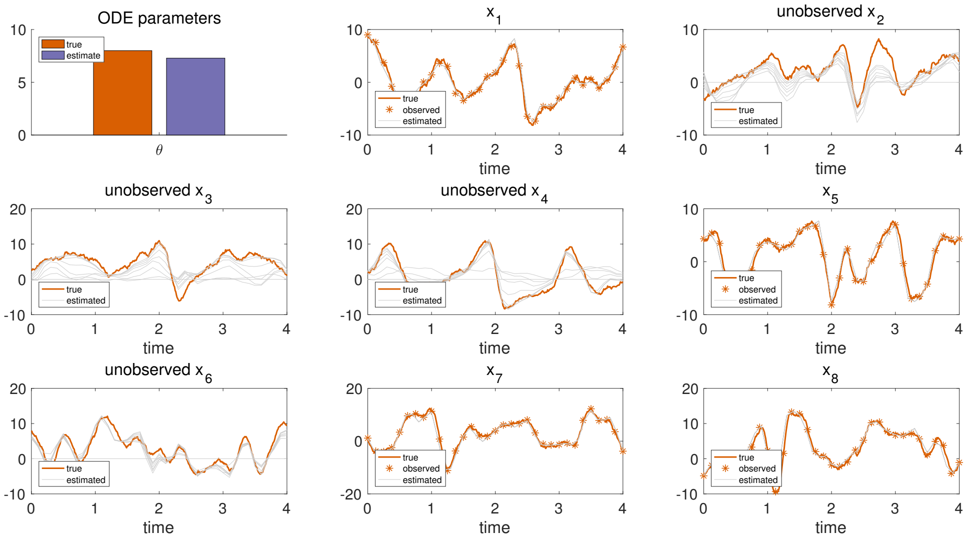

ODE parameters
true
estimate
θ
x1
unobserved x2
unobserved x3
unobserved x4
x5
unobserved x6
x7
x8
true
observed
estimated
time

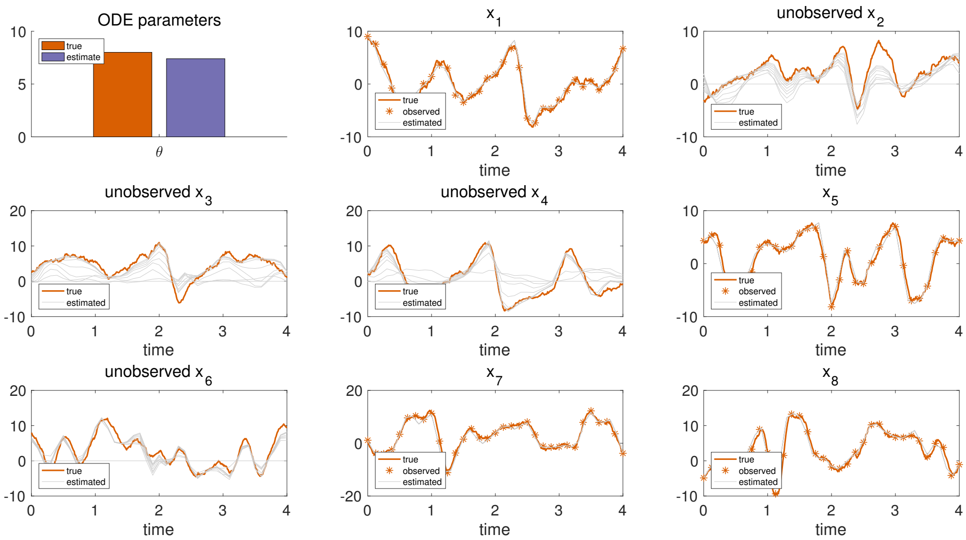

ODE parameters
true
estimate
θ
x1
unobserved x2
unobserved x3
unobserved x4
x5
unobserved x6
x7
x8
true
observed
estimated
time

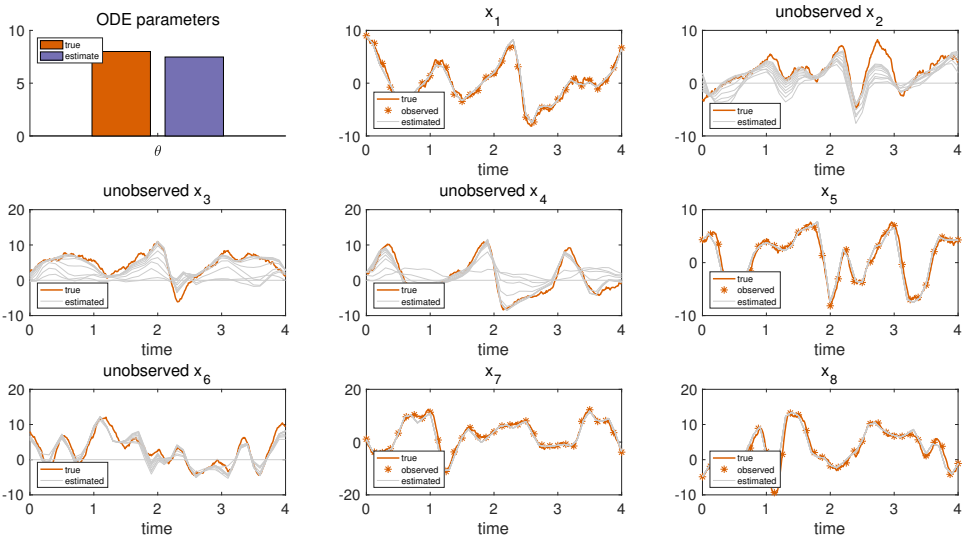

ODE parameters
true
estimate
θ
x1
unobserved x2
unobserved x3
unobserved x4
x5
unobserved x6
x7
x8
true
observed
estimated
time


ODE parameters
true
estimate
θ
x1
unobserved x2
unobserved x3
unobserved x4
x5
unobserved x6
x7
x8
time
true
observed
estimated


ODE parameters
true
estimate
θ
x 1
unobserved x 2
unobserved x 3
unobserved x 4
x 5
unobserved x 6
x 7
x 8
true
observed
estimated
time


ODE parameters
true
estimate
θ
x 1
unobserved x 2
unobserved x 3
unobserved x 4
x 5
unobserved x 6
x 7
x 8
true
observed
estimated
time

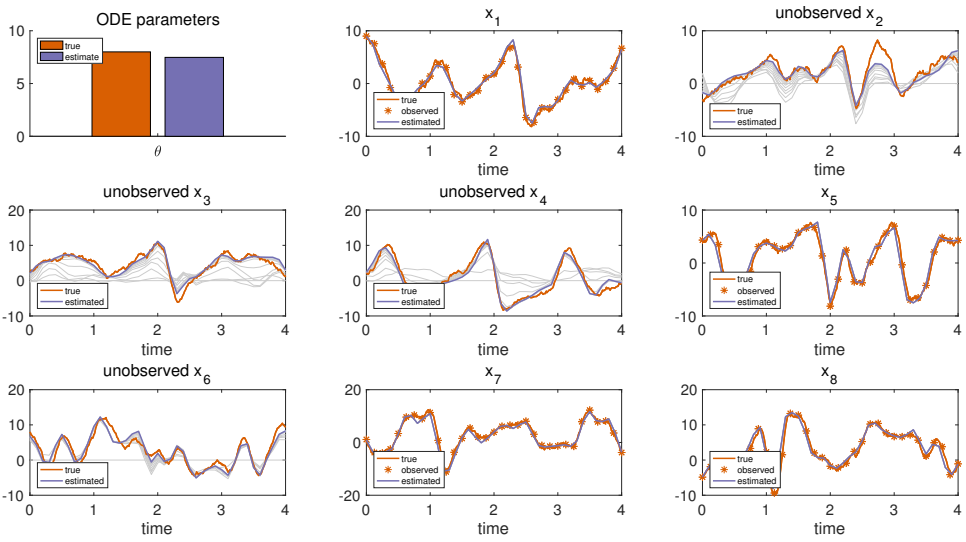

ODE parameters
true
estimate
θ
x 1
unobserved x 2
unobserved x 3
unobserved x 4
x 5
unobserved x 6
x 7
x 8
true
observed
estimated
time




```
end
```

4.10.3 Final result

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



4.11 Time Taken

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

```
time taken: 388.2428 seconds
```

Chapter 5

Lorenz Attractor

5.1 Simulation Settings

```
simulation.state_obs_variance = @(mean)(bsxfun(@times,[2,2],...
    ones(size(mean))));              % observation noise
simulation.ode_param = [10,28,8/3];  % true ODE parameters;
simulation.final_time = 20;          % end time for integration
simulation.int_interval = 0.01;      % integration interval
simulation.time_samp = 0:0.1:simulation.final_time; % sample times for observations
simulation.init_val = -7.*ones(1,3); % initial state values

%symbols of observed states that appear in the 'ODEs.txt' file.
simulation.observed_states = {'[x]','[z]'};
```

5.2 User Input

5.2.1 Path to ODEs

```
odes_path = 'Lorenz_attractor_ODEs.txt';
```

5.2.2 Symbols

symbols of states and parameters in the '_ODEs.txt' file

States **x**:

```
symbols.state = {'[x]','[y]','[z]'}; % symbols of states in 'ODEs.txt' file
```

ODE parameters θ (symbols of parameters in 'ODEs.txt' file):

```
symbols.param = {'[\sigma]','[\rho]','[\beta]'};
```

5.2.3 Kernel

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*ones(1,length(symbols.state)); % gamma for gradient matching model
```

5.2.4 Estimation times

```
time.est = 0:0.1:20;                % estimation times
```

5.2.5 Type of pseudo-inverse

Type of pseudo inverse; options: 'Moore-Penrose' or 'modified Moore-Penrose'

```
opt_settings.pseudo_inv_type = 'Moore-Penrose';
```

5.2.6 Optimization settings

```
opt_settings.coord_ascent_numb_iter = 40;  % number of coordinate ascent iterations

% The observed state trajectories are clamped to the trajectories
% determined by standard GP regression (Boolean)
opt_settings.clamp_obs_state_to_GP_fit = true;
```

Plot settings: layout and size

```
plot_settings.size = [1600, 800]; plot_settings.layout = [2,2];
```

5.3 Import ODEs

```
ode = import_odes(symbols,odes_path);


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


ODEs:
    '[\sigma] .* ([y] - [x])'
    '[\rho] .* [x] - [y] - [x] .* [z]'
    '[x] .* [y] - [\beta] .* [z]'
```

5.4 Simulate Trajectory Observations

5.4.1 Generate ground truth by numerical integration

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

5.4.2 Generate state observations

```
if ~iscell(simulation.observed_states)
    ratio_observed = simulation.observed_states;
    state_obs_idx = zeros(1,simulation.numb_odes,'logical');
    idx = randperm(simulation.numb_odes);
    idx = idx(1:floor(simulation.numb_odes * ratio_observed));
    state_obs_idx(idx) = 1;
    simulation.observed_states = symbols.state(state_obs_idx);
end

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

5.4.3 Symbols

```
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');
```

5.4.4 Setup plots

Only the state dynamics are (partially) observed.

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

tic; %start timer
```

5.4.5 Prior on States and State Derivatives

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

5.5 State Couplings in ODEs

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

5.6 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(ode,symbols);
```

5.7 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,symbols,coupling_idx.states);
```

5.8 Fitting observations of state trajectories

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

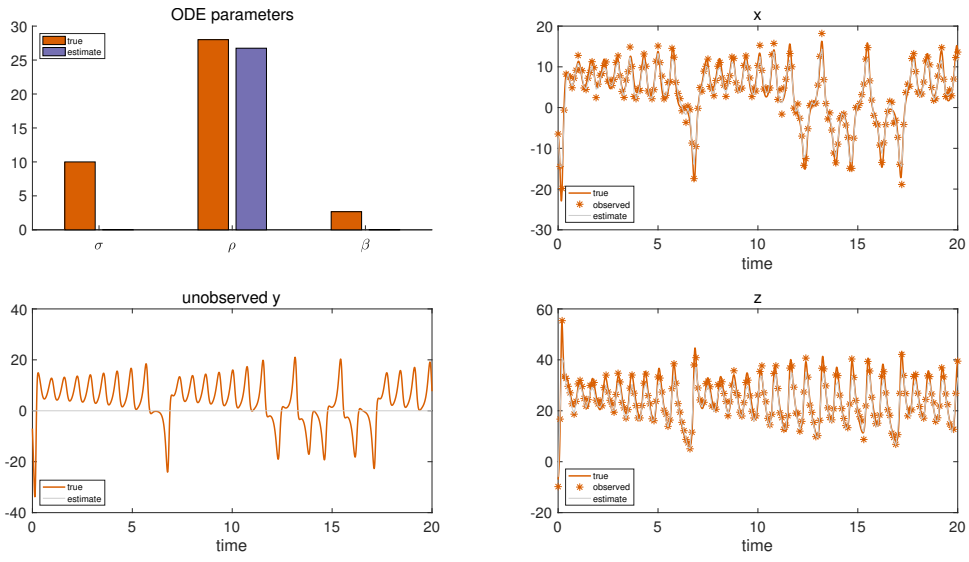
5.9 Coordinate Ascent Variational Gradient Matching

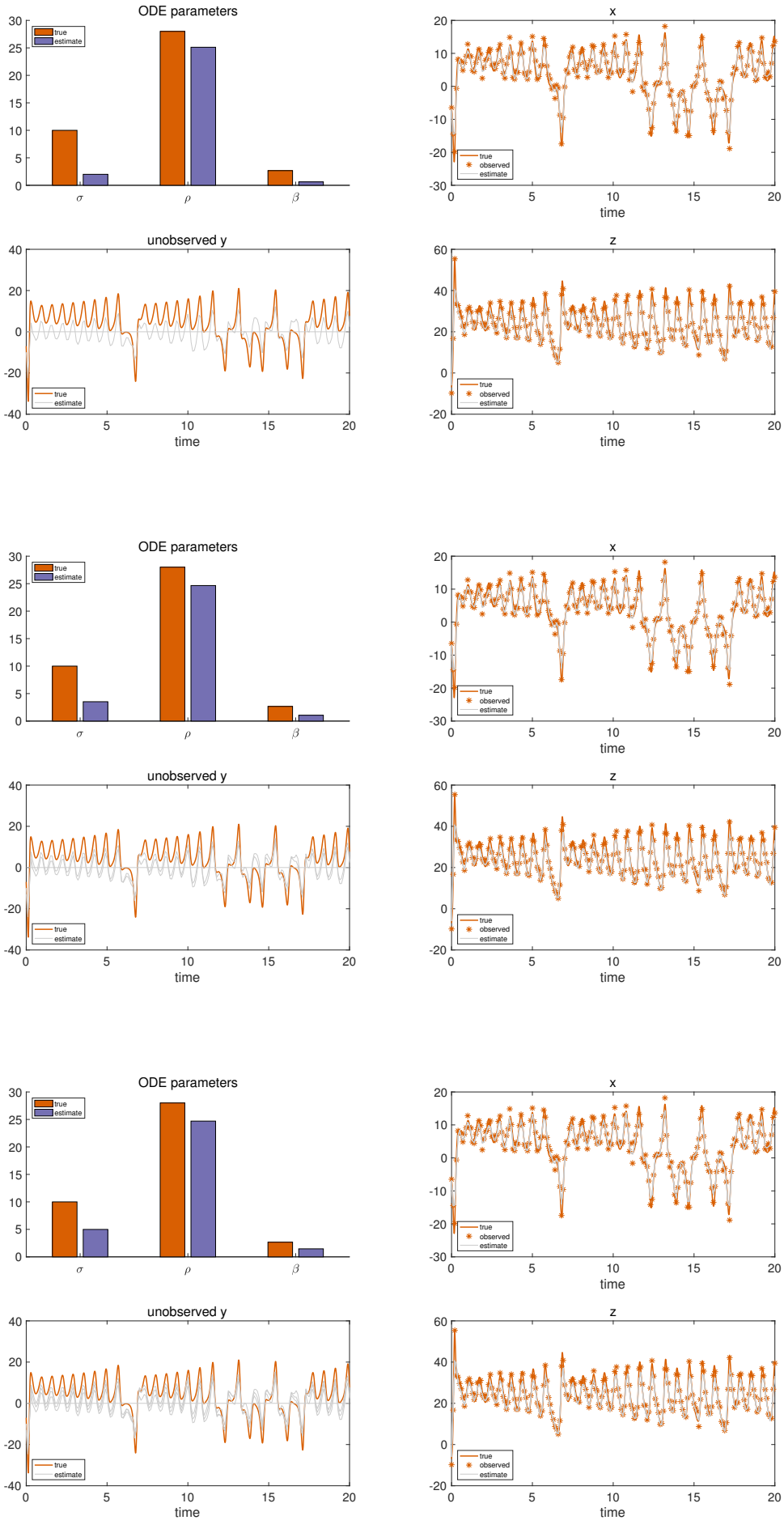
We minimize the KL-divergence in equation (10) by coordinate descent (where each step is analytically tractable) by iterating between determining the proxy for the distribution over ODE parameters $\hat{q}(\theta)$ and the proxies for the distribution over individual states $\hat{q}(\mathbf{x}_u)$.

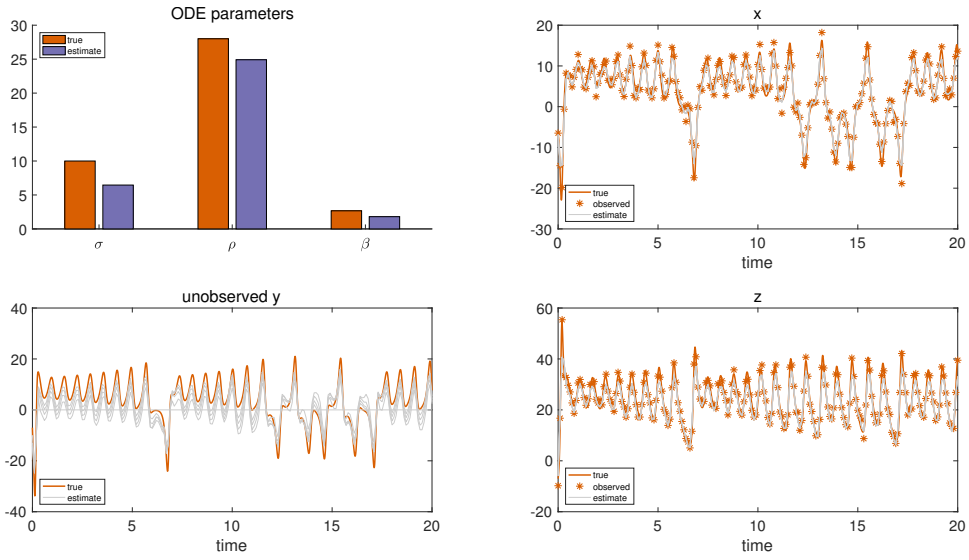
```
state.proxy.mean = mu;  % Initialize the state estimation by the GP regression posterior
for i = 1:opt_settings.coord_ascent_numb_iter
```

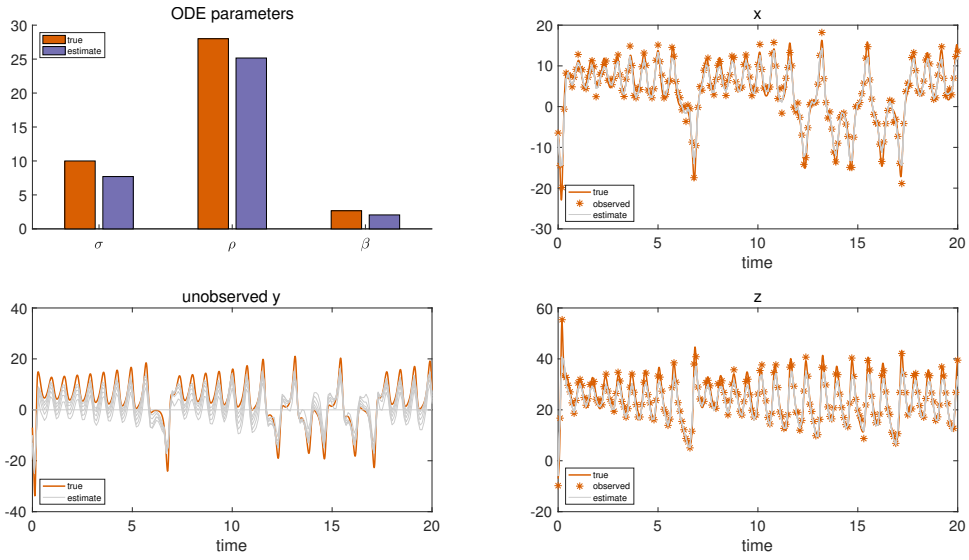
5.9.1 Proxy for ODE parameters

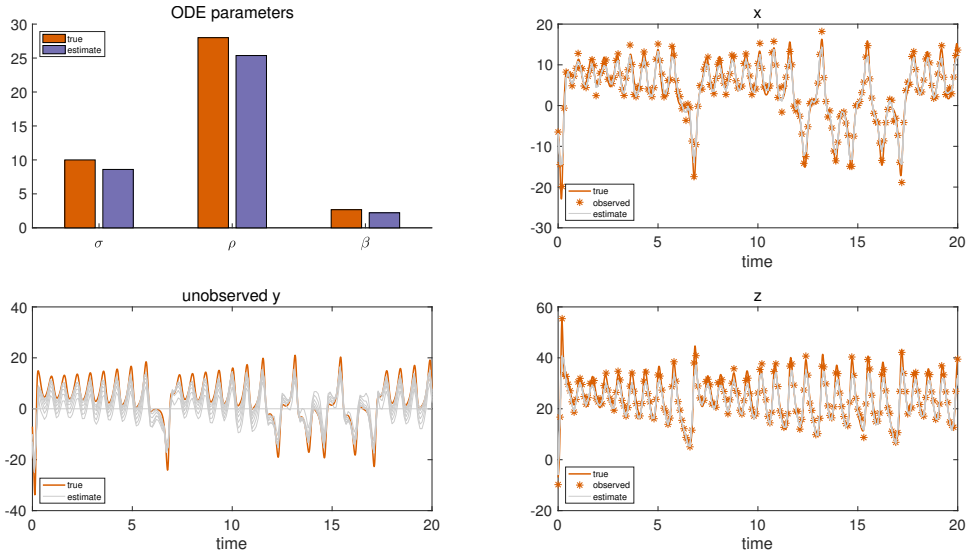
```
    [param_proxy_mean,param_proxy_inv_cov] = proxy_for_ode_parameters(state.proxy.mean,...
         dC_times_invC,ode_param.lin_comb,symbols,A_plus_gamma_inv,opt_settings);

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

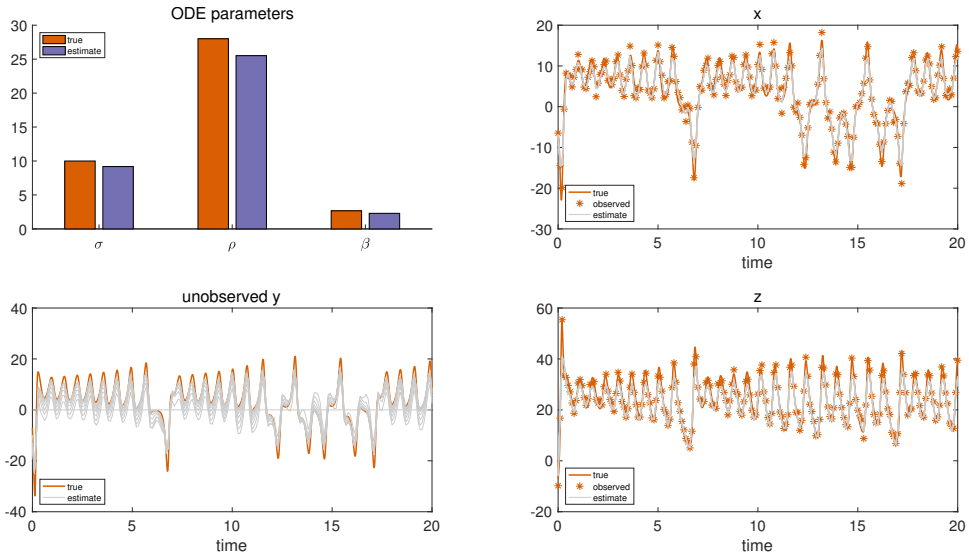


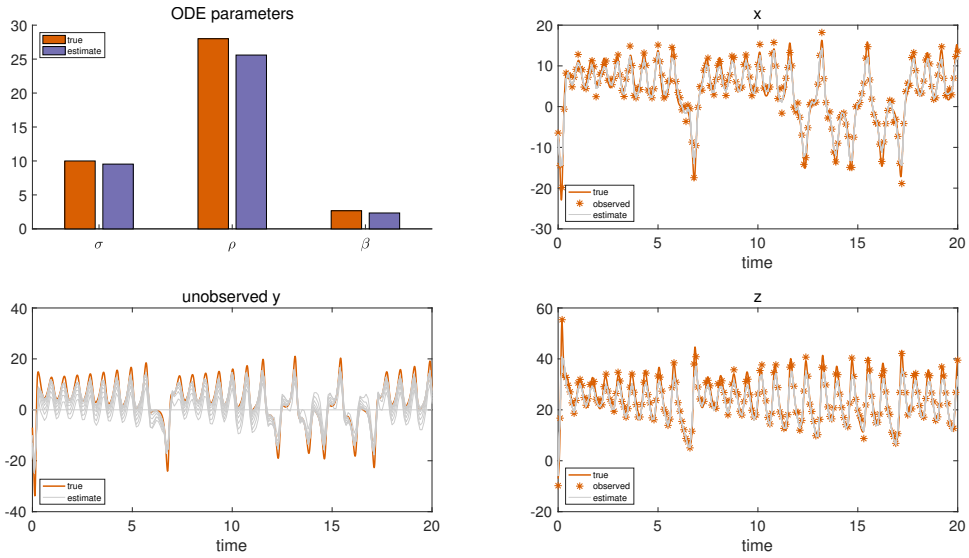

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z

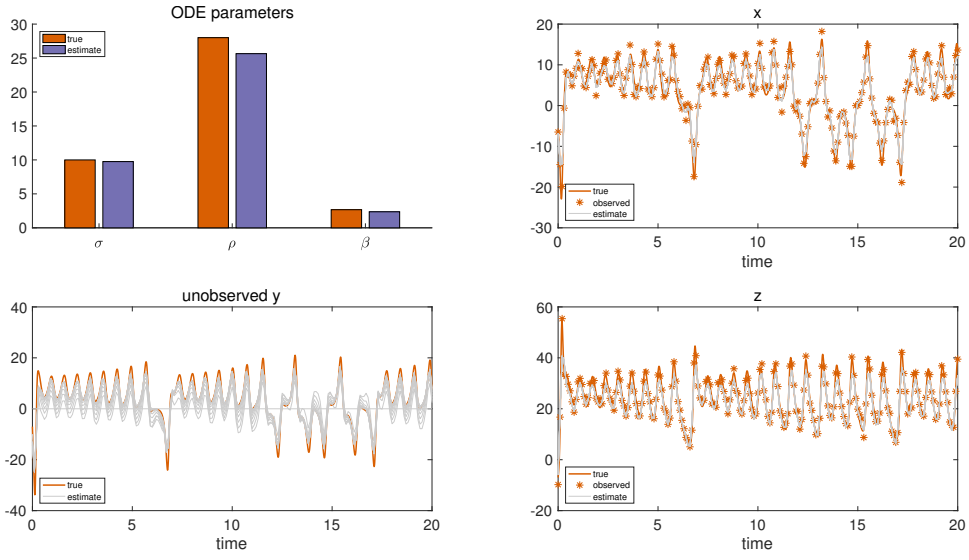

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z

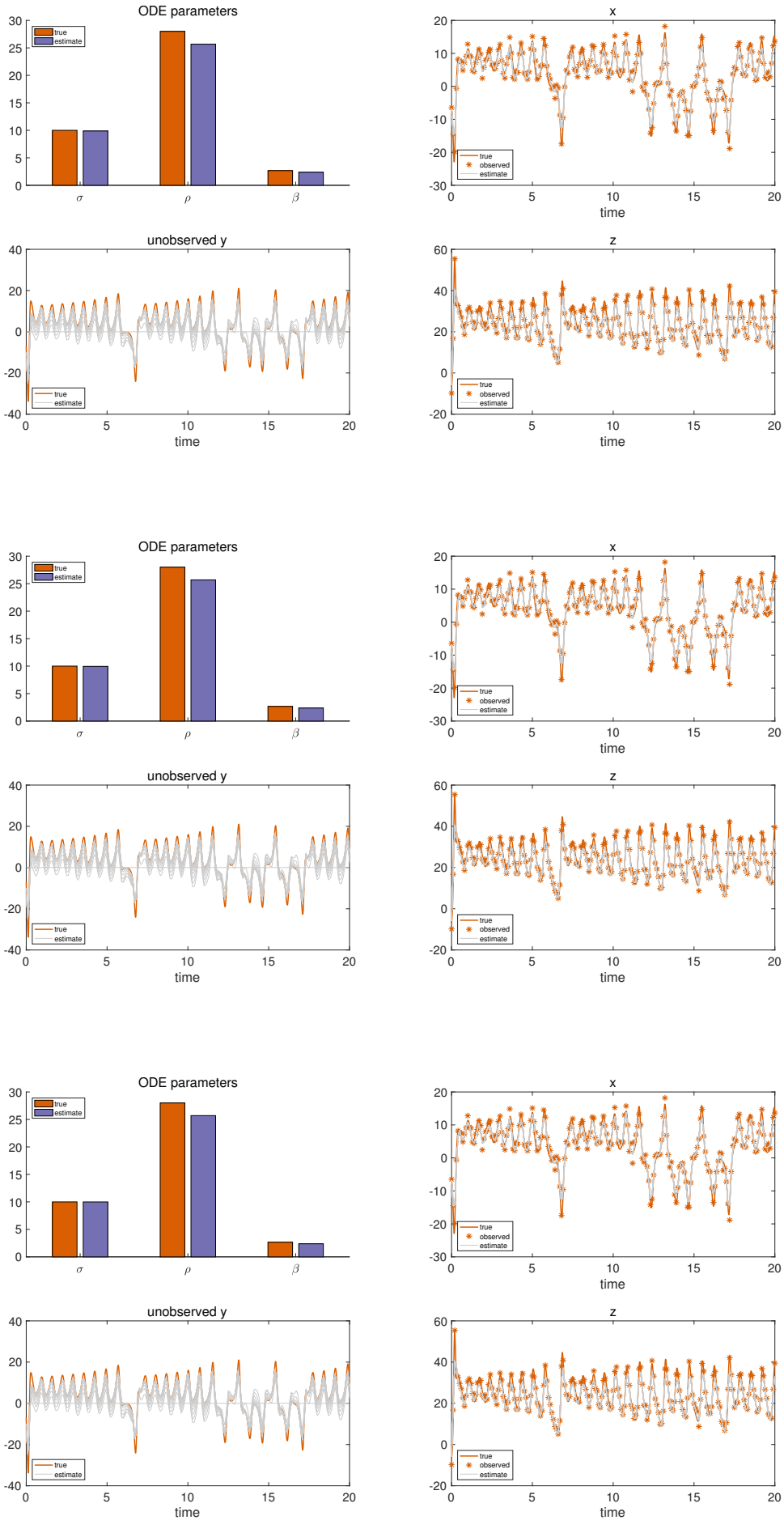

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z

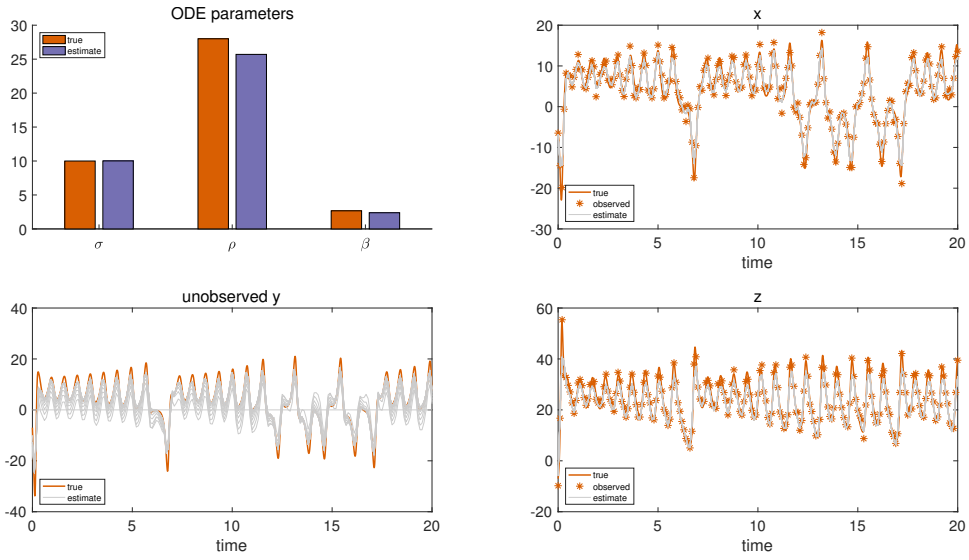

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z

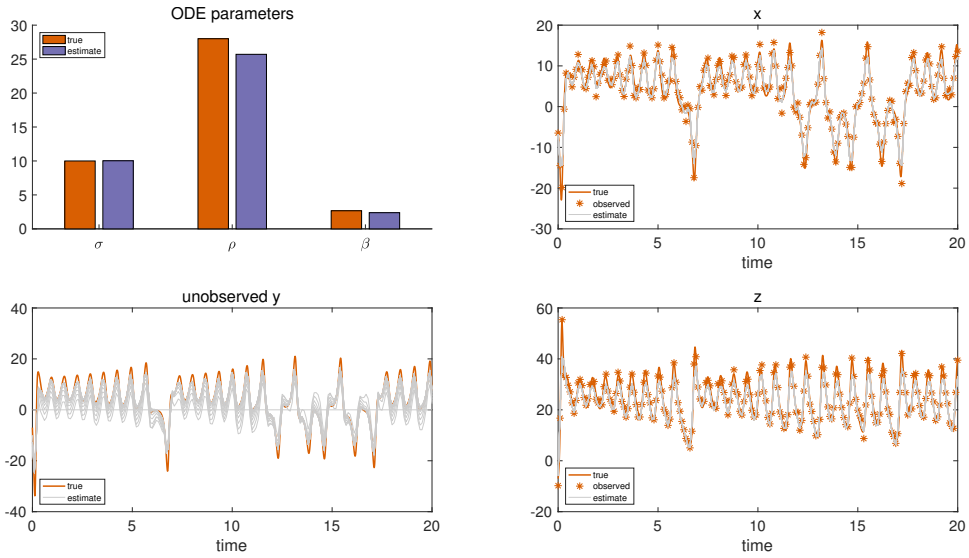

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z

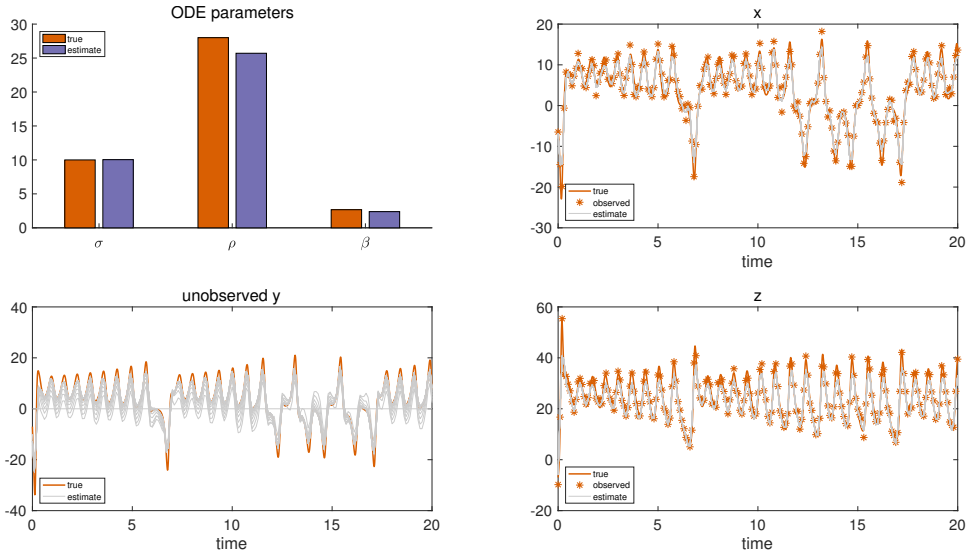

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z

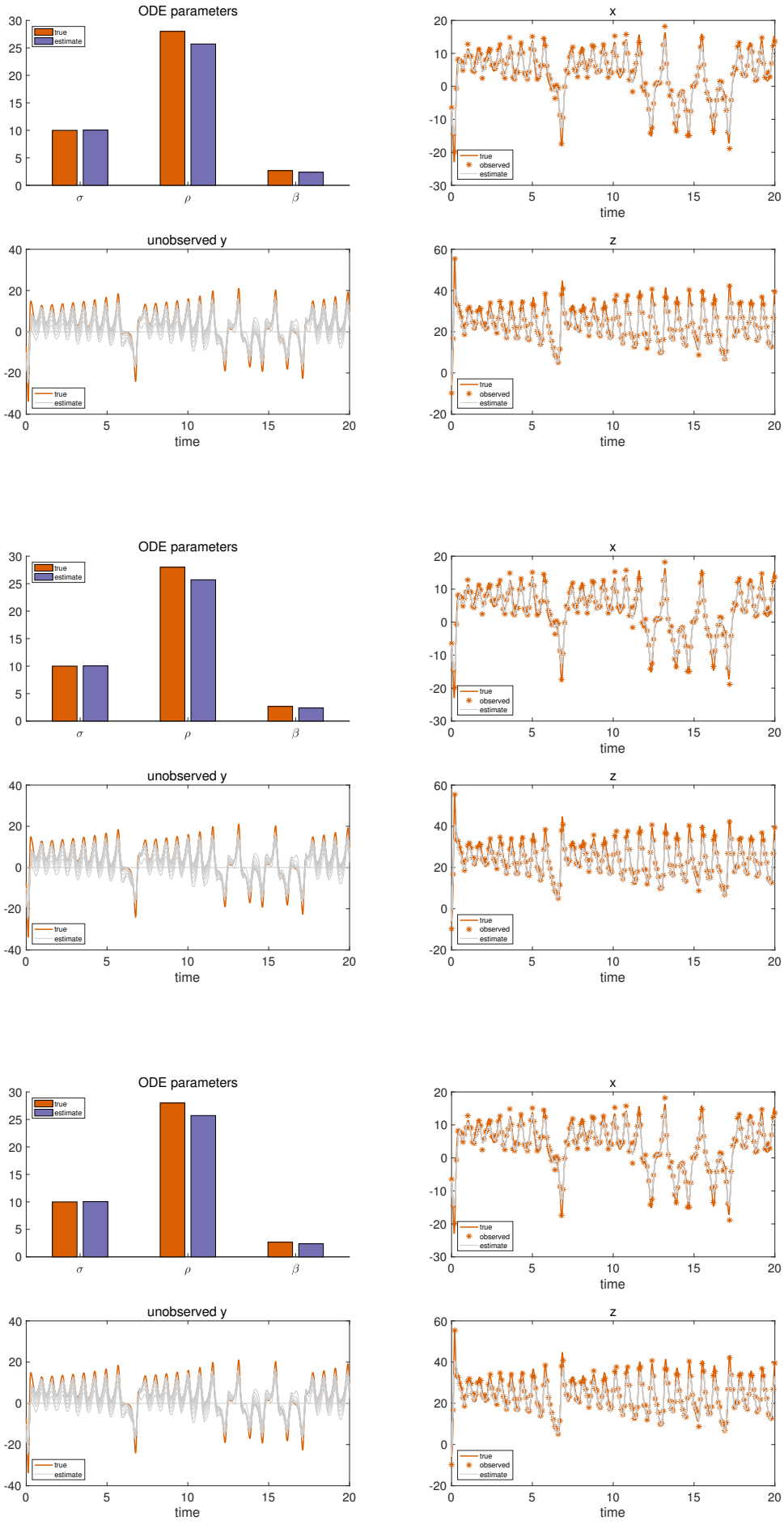

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z

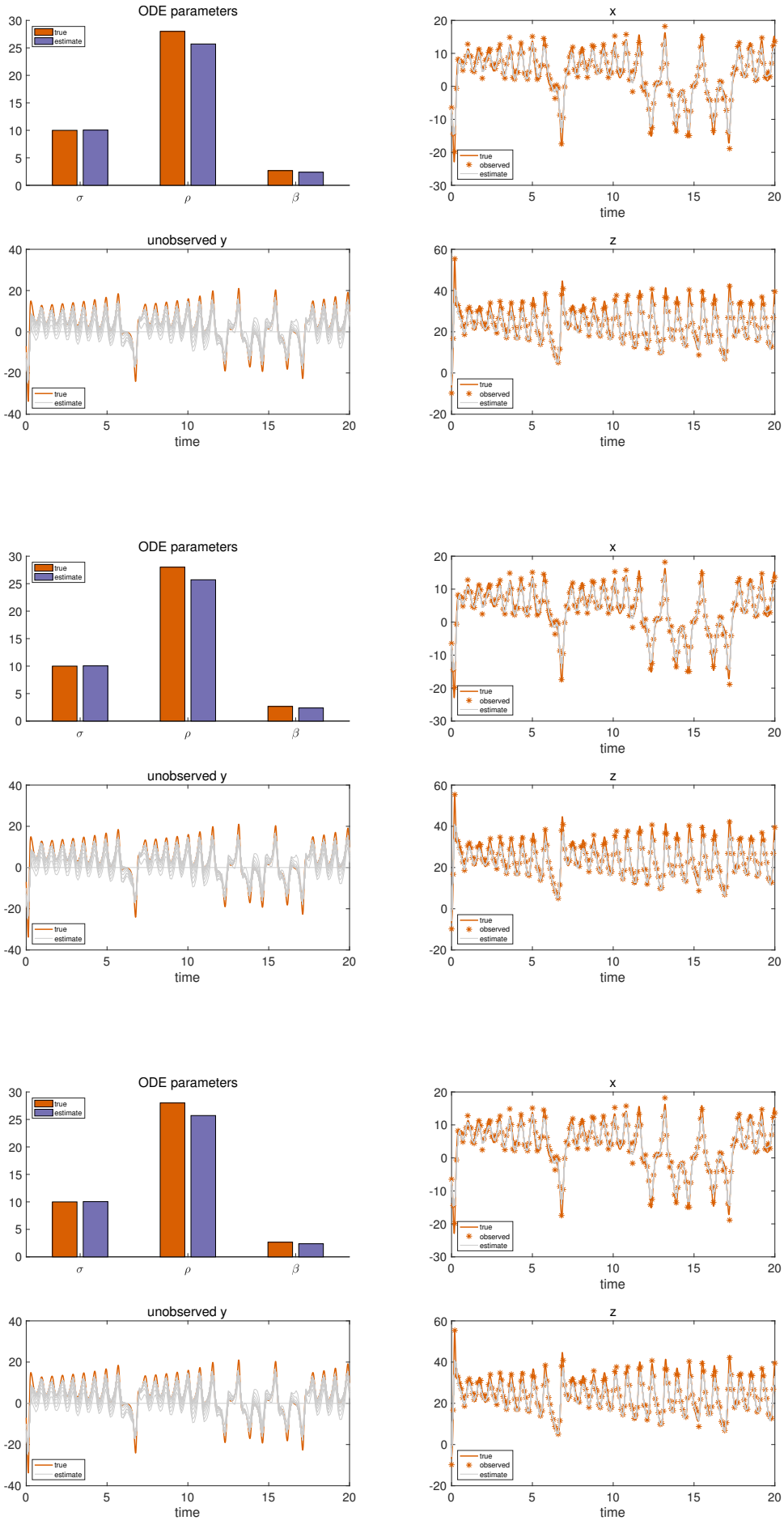

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z

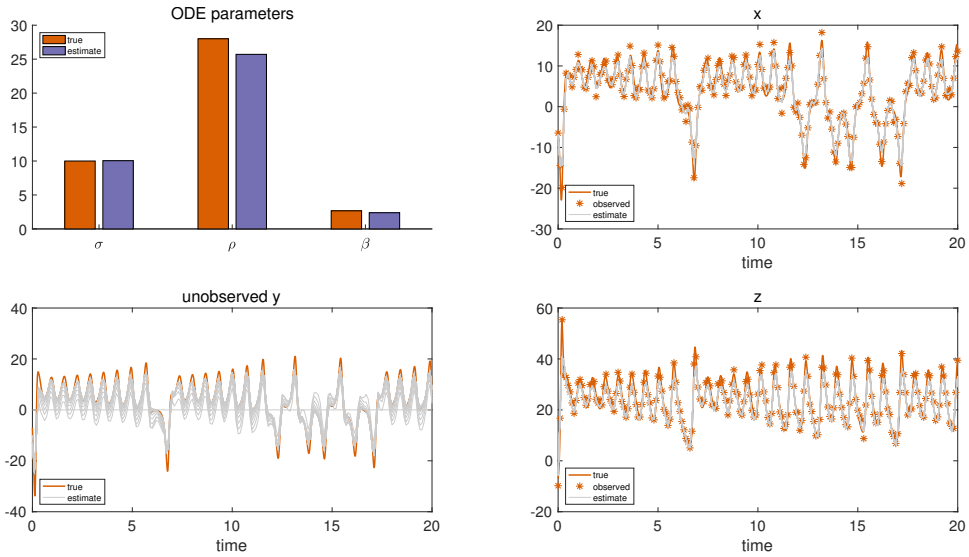

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z

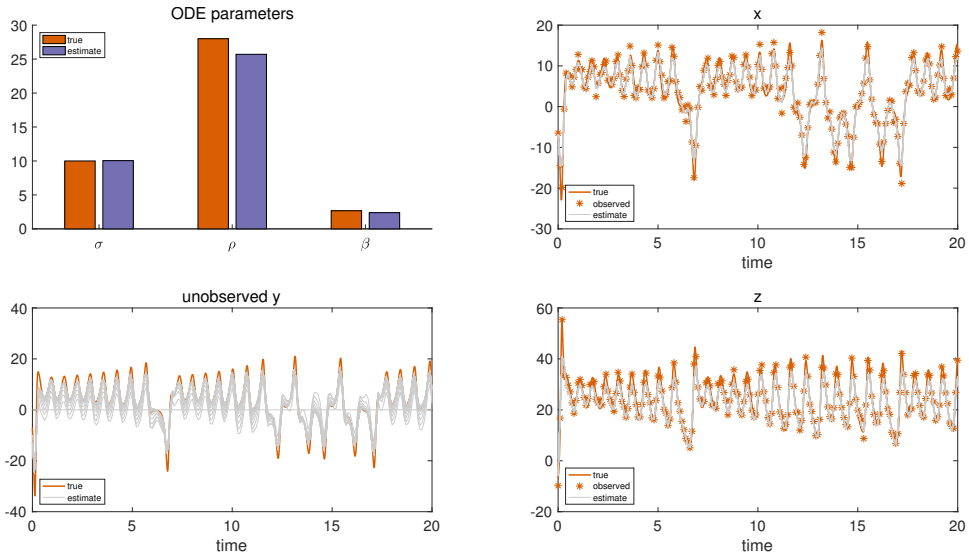

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z

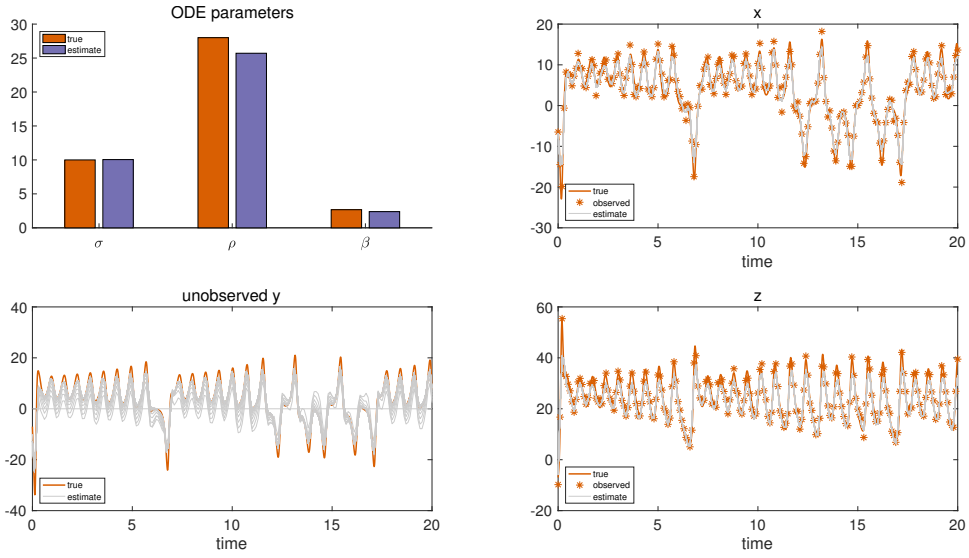

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z

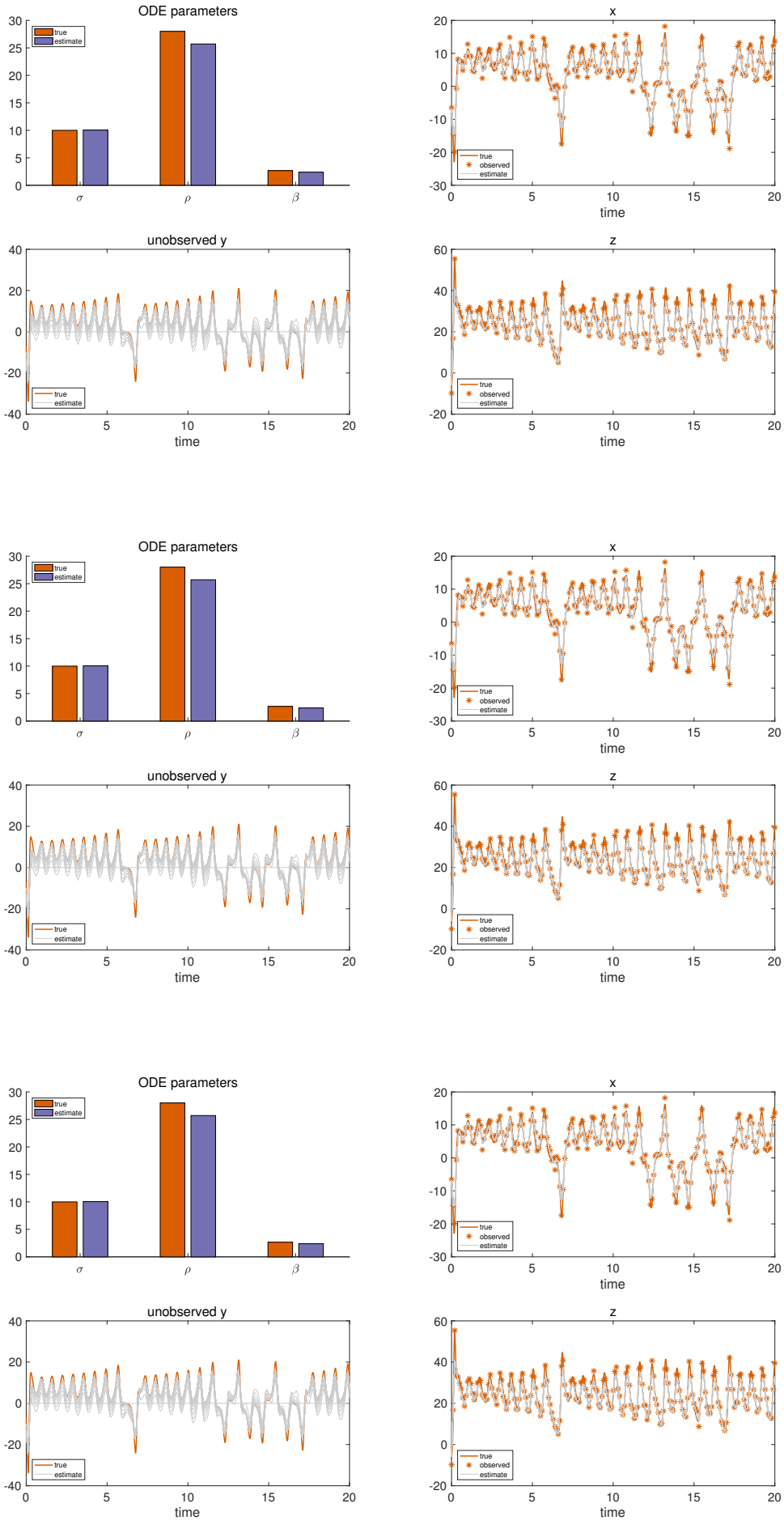

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z
ODE parameters
x
unobserved y
z
ODE parameters
x
unobserved y
z

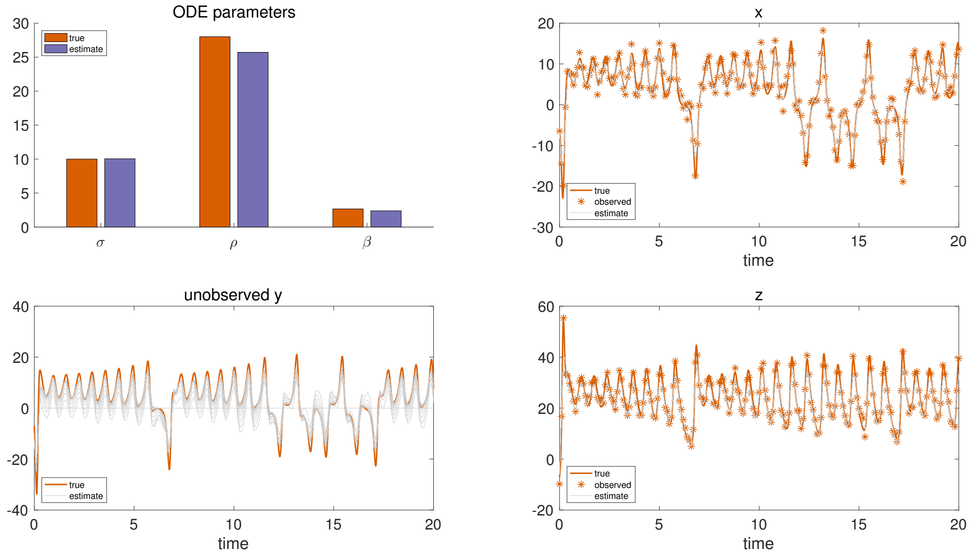

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z

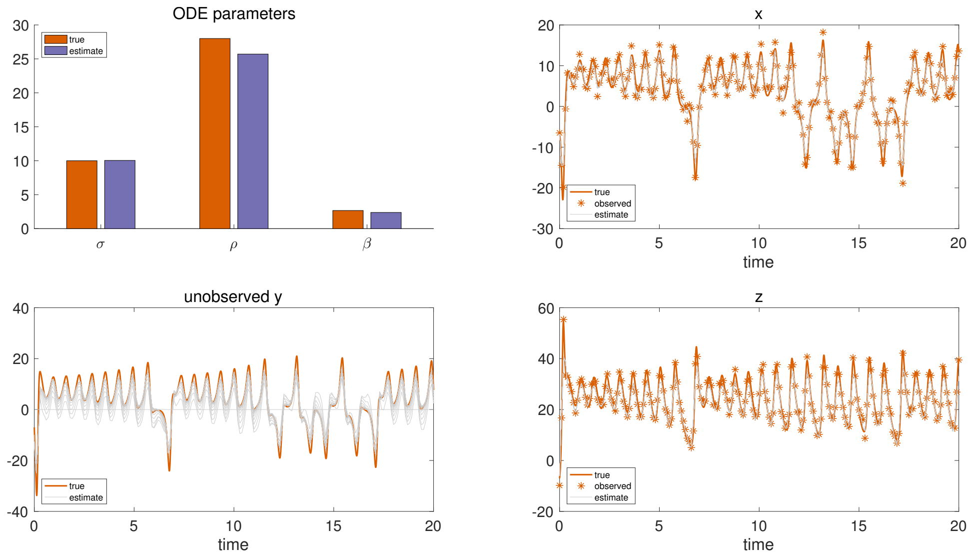

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z

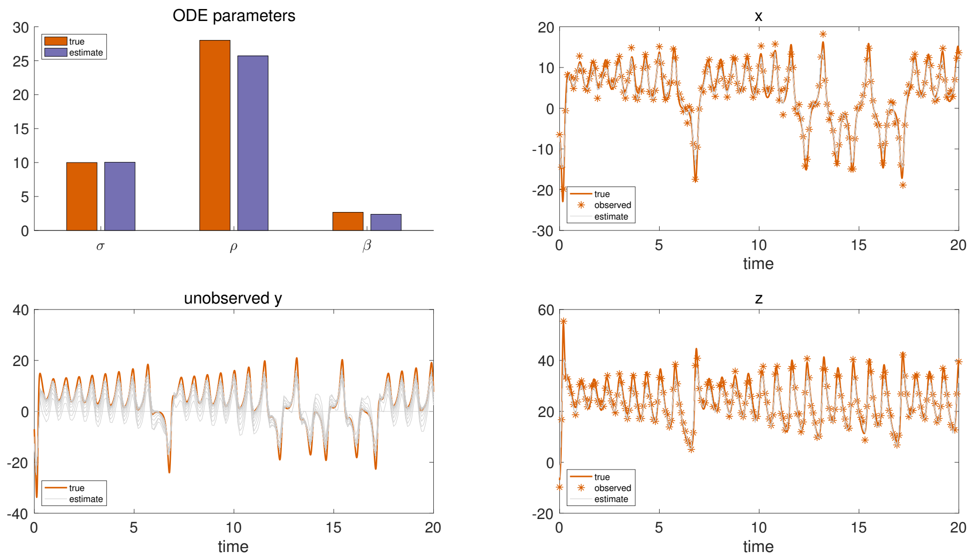

ODE parameters
x
unobserved y
z
time

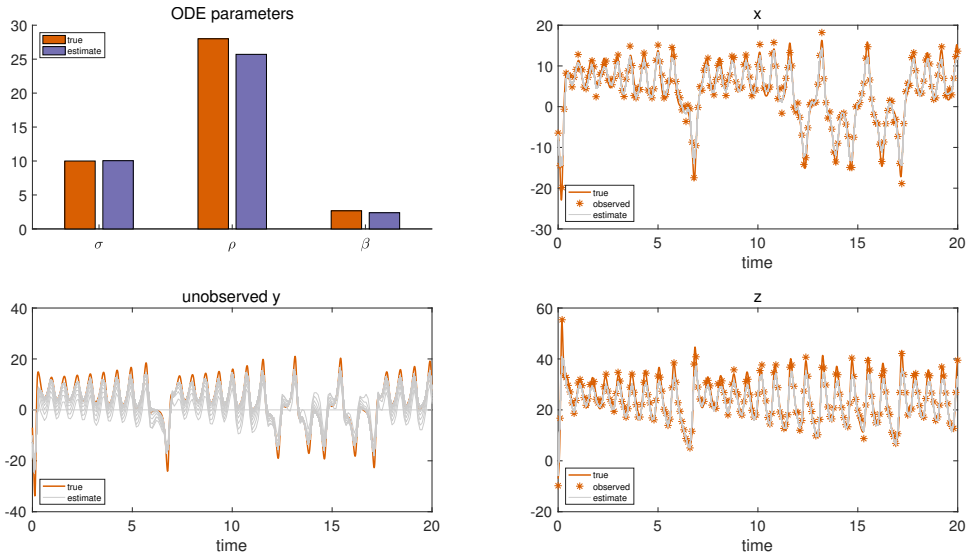

ODE parameters
x
unobserved y
z
time

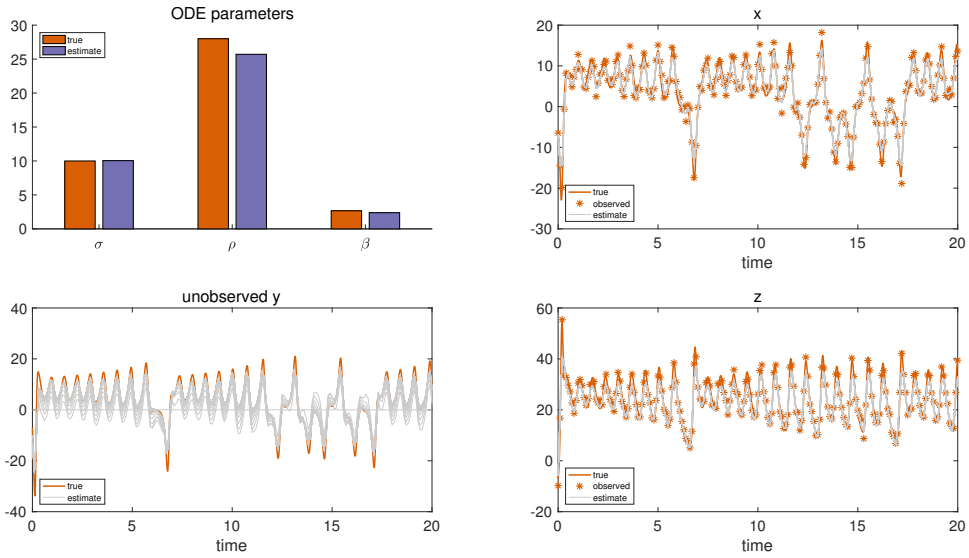

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z

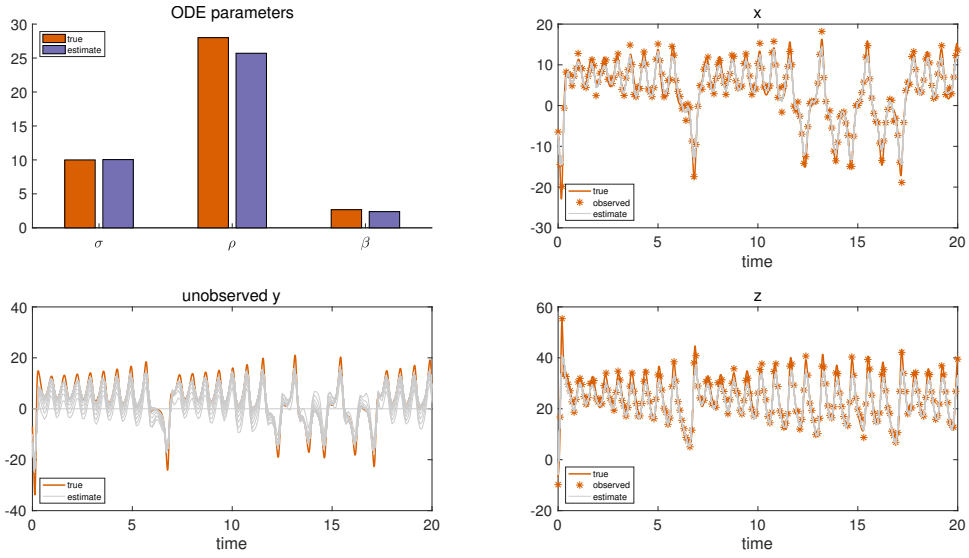

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
true
estimate
time
z
true
observed
estimate
time

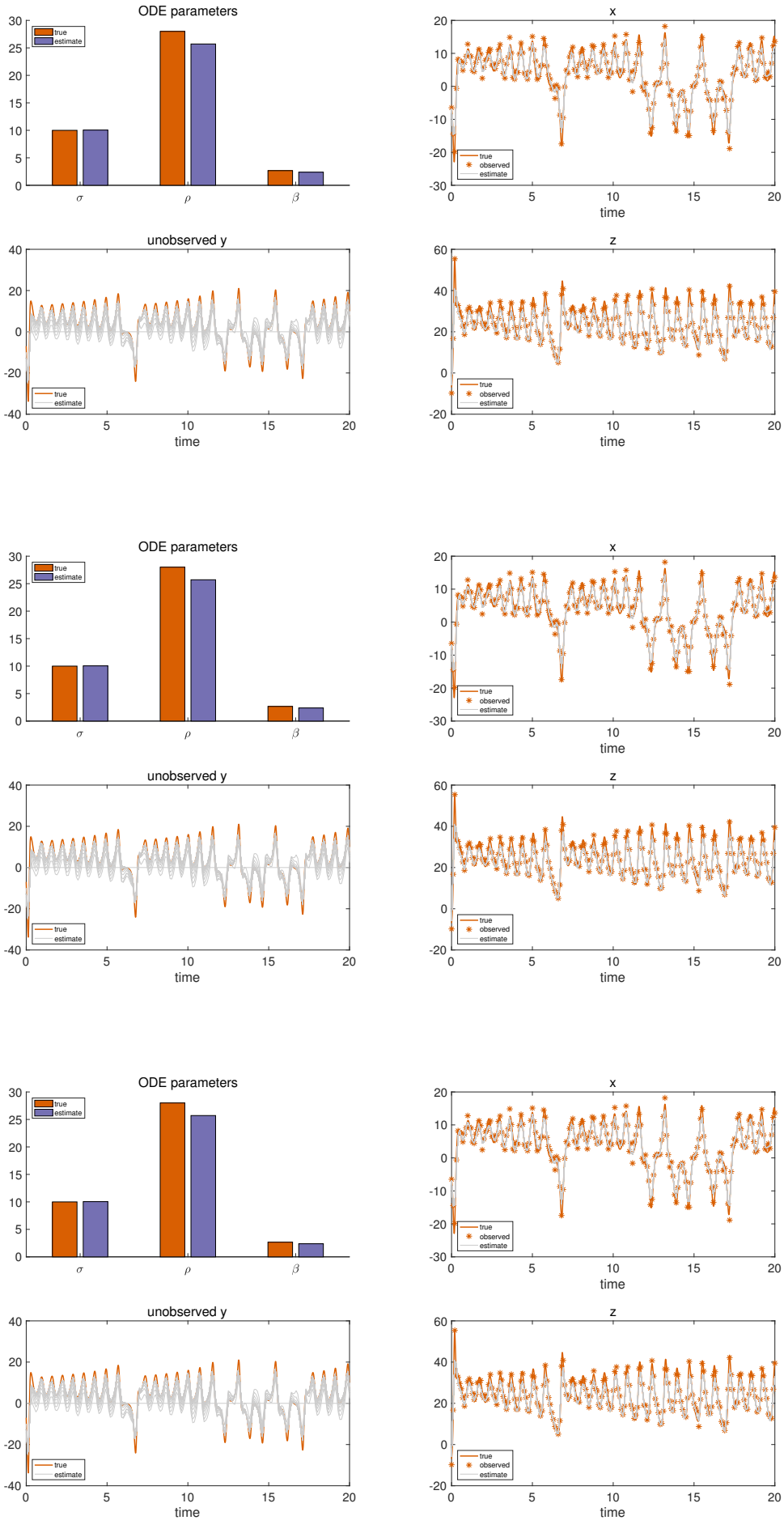

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
true
estimate
time
z
true
observed
estimate
time

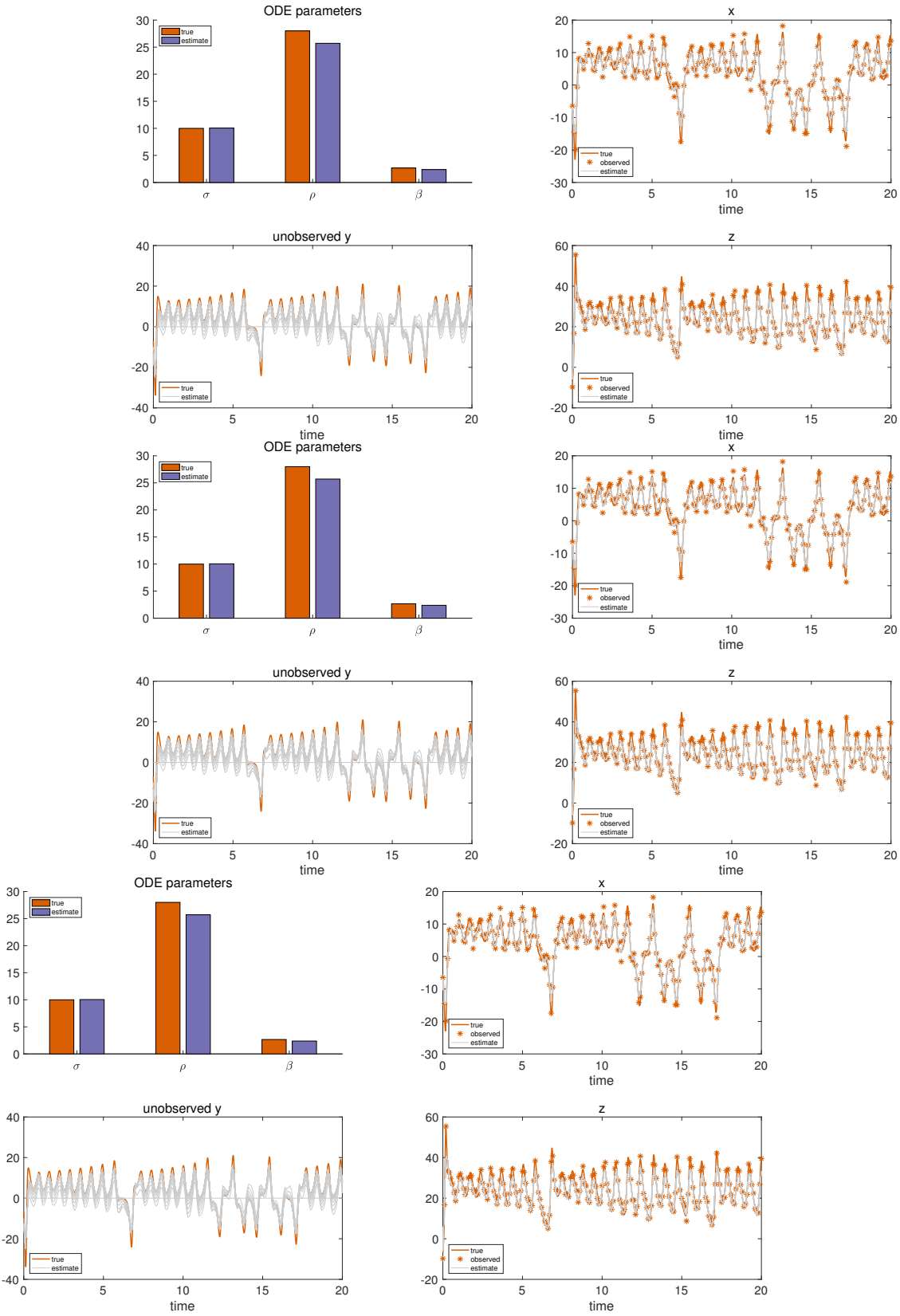

ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
true
estimate
time
z
true
observed
estimate
time


ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z


ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z


ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z


ODE parameters
true
estimate
σ
ρ
β
x
true
observed
estimate
time
unobserved y
z



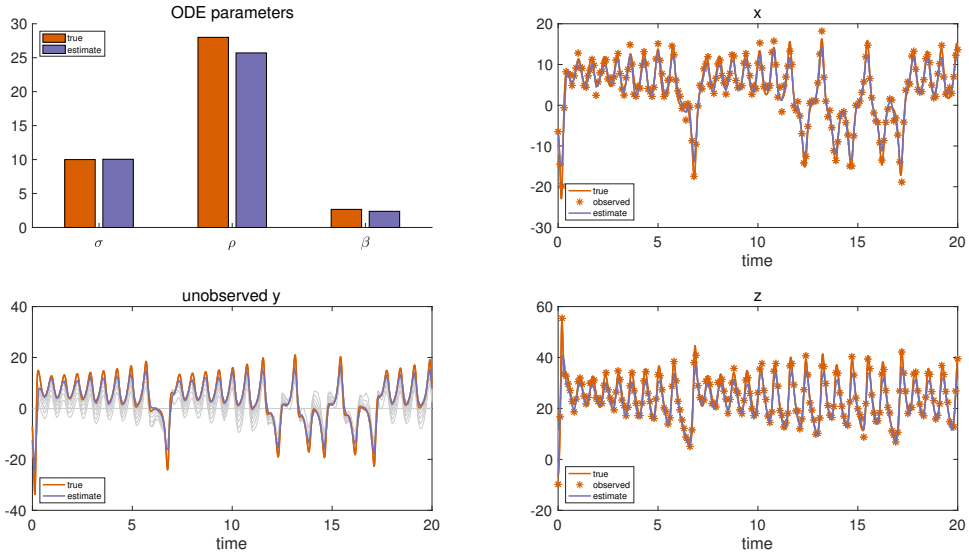
5.9.2 Proxy for individual states

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

```
end
```

5.9.3 Final result

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



5.10 Time Taken

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

```
time taken: 71.0552 seconds
```

Chapter 6

Subroutines for Lotka-Volterra, Lorenz 96 and Lorenz Attractor

6.1 Kernel function

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.t1 = sym('time1'); assume(kernel.t1,'real'); kernel.t2 = sym('time2');
assume(kernel.t2,'real');
% RBF kernel
kernel.func = kernel.param_sym(1).*exp(-(kernel.t1-kernel.t2).^2./...
    (kernel.param_sym(2).^2));
kernel.name = 'rbf';
```

kernel derivatives

```
for i = 1:length(kernel)
    kernel.func_d = diff(kernel.func,kernel.t1);
    kernel.func_dd = diff(kernel.func_d,kernel.t2);
   GP.fun = matlabFunction(kernel.func,'Vars',{kernel.t1,kernel.t2,kernel.param_sym});
   GP.fun_d = matlabFunction(kernel.func_d,'Vars',{kernel.t1,kernel.t2,kernel.param_sym});
    GP.fun_dd = matlabFunction(kernel.func_dd,'Vars',{kernel.t1,kernel.t2,...
        kernel.param_sym});
end
```

populate GP covariance matrix

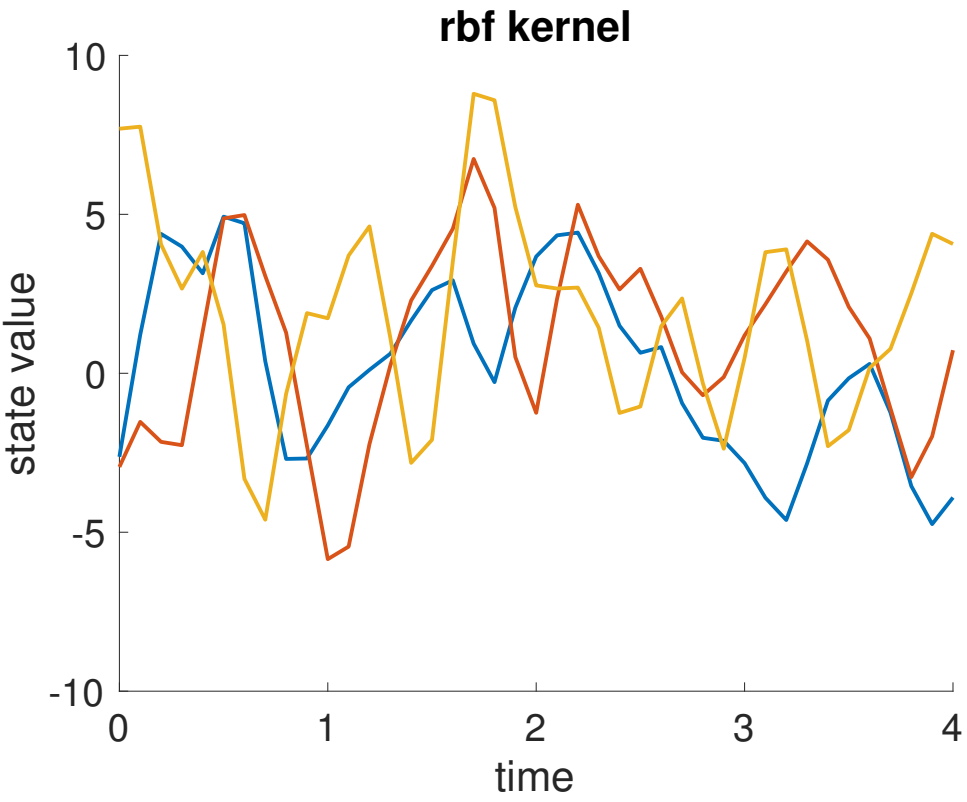
```
for t=1:length(time_est)
    C(t,:)=GP.fun(time_est(t),time_est,kernel.param);
    dC(t,:)=GP.fun_d(time_est(t),time_est,kernel.param);
    Cd(t,:)=GP.fun_d(time_est,time_est(t),kernel.param);
    ddC(t,:)=GP.fun_dd(time_est(t),time_est,kernel.param);
end
```

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(size(C,1),1);
end
[~,D] = eig(C);
if any(diag(D)<0)
    error('C has negative eigenvalues!');
elseif any(diag(D)<1e-6)
    warning('C is badly scaled');
end
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
```

6.2 Fitting state observations

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_relation,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: $\boldsymbol{\Sigma}_{\boldsymbol{k}}^{-\mathbf{1}} := \boldsymbol{\sigma}_{\boldsymbol{k}}^{-\mathbf{2}}\mathbf{I} + \mathbf{C}_{\boldsymbol{\phi}_{\boldsymbol{k}}}^{-\mathbf{1}}$

```
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: $\boldsymbol{\mu}_{\boldsymbol{k}}(\mathbf{y}_{\boldsymbol{k}}) := \boldsymbol{\sigma}_{\boldsymbol{k}}^{-\mathbf{2}} \left(\boldsymbol{\sigma}_{\boldsymbol{k}}^{-\mathbf{2}}\mathbf{I} + \mathbf{C}_{\boldsymbol{\phi}_{\boldsymbol{k}}}^{-\mathbf{1}}\right)^{-\mathbf{1}} \mathbf{y}_{\boldsymbol{k}}$

```
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
```

6.3 Find state ODE couplings

```
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
```

6.4 Rewrite ODEs as linear combination in parameters

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

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
```

6.5 Rewrite ODEs as linear combination in individual states

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

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
```

6.6 Proxy for ODE parameters

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

```
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(eye(size(param_proxy_inv_cov,1)))) ...
        + 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
```

6.7 Proxy for individual states

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

$$+ E_{Q_{-u}} \ln \mathcal{N}\left(\mathbf{x}_u; \mu_u(\mathbf{Y}), \Sigma_u\right)\big),$$

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

Indices of observed states

```
tmp = cellfun(@(x) {strcmp(x,observed_states)},symbols.state);
state_obs_idx = cellfun(@(x) any(x),tmp);
```

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
```

Iterate through states

```
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) * mu(:,u)));


end


end



function generate_Lorenz96_ODEs(numb_odes)

for i = 1:numb_odes
    state{i} = ['[x_' num2str(i) ']'];
end
param = '[\theta]';

ode{1} = ['(' state{2} ' - ' state{end-1} ') .* ' state{end} ' - ' state{1} ' + ' param];
ode{2} = ['(' state{3} ' - ' state{end} ') .* ' state{1} ' - ' state{2} ' + ' param];
for i = 3:numb_odes-1
   ode{i} = [ '(' state{i+1} ' - ' state{i-2} ') .* ' state{i-1} ' - ' state{i} ' + ' param];
end
ode{numb_odes} = ['(' state{1} ' - ' state{end-2} ') .* ' state{end-1} ' - ' state{end} ' + ' param];
```

```
dlmwrite('Lorenz96_ODEs.txt',[])

for i = 1:numb_odes
    dlmwrite('Lorenz96_ODEs.txt',char(ode{i}),'delimiter','','-append')
end
end
```

6.8 Import ODEs

```
function ode = import_odes(symbols,odes_path)
```

Import ODEs expressions

```
ode.raw = importdata(odes_path);
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}],['state(:,' num2str(u) ')']);
end
for j = 1:length(symbols.param)
   ode.refined{k} = strrep(ode.refined{k},symbols.param{j},['param(' num2str(j) ')']);
end
end
for k = 1:length(ode.refined)
    ode.system{k} = str2func(['@(state,param)(' ode.refined{k} ')']);
end


end
```

6.9 Generate ground truth

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

Integration times

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

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

```
if ~strcmp(odes_path,'Lorenz96_ODEs.txt')
    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.init_val);
else
   OutX_solver = create_Lorenz96(min(time.true),max(time.true),time.true(2)-time.true(1),...
      state.derivative_variance',[simulation.ode_param, length(state_sym)])';ToutX = time.true';
end
state.true_all=OutX_solver;
state.true=state.true_all(itrue,:);


end
```

6.10 Generate observations of states

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


tmp = cellfun(@(x) {strcmp(x,simulation.observed_states)},symbols.state);
state_obs_idx = cellfun(@(x) any(x),tmp);
```

State observations

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

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_samp),time.idx);
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.idx);
end
time.obs_time_to_state_time_relation = zeros(length(simulation.time_samp),...
    length(time.est));
```

```
time.obs_time_to_state_time_relation(time.ind) = 1;
state_mat = eye(size(state.true,2));
state_mat(~state_obs_idx,:) = [];
obs_to_state_relation = sparse(kron(state_mat,time.obs_time_to_state_time_relation));
time.samp = simulation.time_samp;


end
```

6.11 Setup plots

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

Indices of observed states

```
tmp = cellfun(@(x) {strcmp(x,simulation.observed_states)},symbols.state);
state_obs_idx = cellfun(@(x) any(x),tmp);
obs_ind = find(state_obs_idx);
```

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,plot_settings.size(1),plot_settings.size(2)]);
```

ODE parameters

```
h_param = subplot(plot_settings.layout(1),plot_settings.layout(2),1);
h_param.FontSize = 20; h_param.Title.String = 'ODE parameters';
h_param.Title.FontWeight = 'Normal';
set(gca,'XTick',[1:length(symbols.param)]); set(gca,'XTickLabel',symbols.param);
hold on; drawnow

nStates_plot=length(symbols.state);if nStates_plot > 8; nStates_plot = 8; end
```

States

```
u2=0;
for u = 1:nStates_plot
    h_states{u} = subplot(plot_settings.layout(1),plot_settings.layout(2),u+1); cla;
   p.true = plot(time.true,state.true_all(:,u),'LineWidth',2,'Color',[217,95,2]./255);
    h_states{u}.Title.String = symbols.state{u}(2:end-1);
    hold on;
    if any(obs_ind==u)
        u2=u2+1;
        p.obs = plot(simulation.time_samp,state.obs(:,u2),'*','Color',...
            [217,95,2]./255,'MarkerSize',10);
    end
```

```
    h_states{u}.FontSize = 20;
    if any(obs_ind==u)
        h_states{u}.Title.String = symbols.state{u}(2:end-1);
    else
        h_states{u}.Title.String = ['unobserved ' symbols.state{u}(2:end-1)];
    end
    h_states{u}.Title.FontWeight = 'Normal';
    h_states{u}.XLim = [min(time.est),max(time.est)];
    h_states{u}.XLabel.String = 'time'; hold on;
end


end
```

6.12 Plot results

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

% Indices of observed states
tmp = cellfun(@(x) {strcmp(x,simulation.observed_states)},symbols.state);
state_obs_idx = cellfun(@(x) any(x),tmp);
obs_ind = find(state_obs_idx);

nStates_plot=length(symbols.state);if nStates_plot > 8; nStates_plot = 8; end

u2=0;
for u = 1:nStates_plot
    if strcmp(plot_type,'final')

        % State proxy variance
        if ~any(obs_ind==u)
            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(:,u)-1*sqrt(state_proxy_variance),1)];
            f = fill(h_states{u},[time.est'; flip(time.est',1)], shaded_region,...
                [222,235,247]/255);
            set(f,'EdgeColor','None');
        end

        % Replot true states
        p.true = plot(h_states{u},time.true,state.true_all(:,u),'LineWidth',...
            2,'Color',[217,95,2]./255);

        % Replot state observations
        if any(obs_ind==u)
            u2=u2+1;
            p.obs = plot(h_states{u},simulation.time_samp,state.obs(:,u2),...
                '*','Color',[217,95,2]./255,'MarkerSize',10);
        end
```

```
        % State proxy mean (final)
        hold on;
        p.est = plot(h_states{u},time.est,state.proxy.mean(:,u),'Color',...
            [117,112,179]./255,'LineWidth',2);
    else
        % state proxy mean (not final)
        hold on; p.est = plot(h_states{u},time.est,state.proxy.mean(:,u),...
            'LineWidth',0.1,'Color',[0.8,0.8,0.8]);
    end
    % Specify legend entries
    if state_obs_idx(u)
        legend(h_states{u},[p.true,p.obs,p.est],{'true','observed','estimate'},...
            'Location','southwest','FontSize',12);
    else
        legend(h_states{u},[p.true,p.est],{'true','estimate'},'Location',...
            'southwest','FontSize',12);
    end
end

% ODE parameters
cla(h_param);
try
    b = bar(h_param,1:length(param_proxy_mean),[simulation.ode_param',...
        param_proxy_mean]);
catch
    b = bar(h_param,1:length(param_proxy_mean)+1,[[simulation.ode_param';0],...
        [param_proxy_mean;0]]);
end
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','FontSize',12);
drawnow
end
```

Chapter 7

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.pd

- **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.