Variational Gradient Matching for Dynamical Systems: Dynamic Causal Modeling

Table of Contents

,

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. 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).

Example dynamical system used in this code:* Dynamic Causal Modeling (**visual attention system) with *three hidden neuronal- and 12 hidden hemodynamic states**. The system is affected by **given external inputs** and the states are only **indirectly observed** through the BOLD signal change equation.

User Input: Simulation Settings

- **Simulation ODEs**

```
                Input the ODEs "type" used to generate the data as a string
    Options: 'nonlinear_forward_modulation_by_attention', 'forward_modulation_
    'forward_modulation_by_attention', 'backward_modulation_by_attention', 'ba
    'absent_modulation', 'absent_attention_input', 'absent_photic_input', 'dri
    'photic_input'.

    simulation.odes = 'nonlinear_forward_modulation_by_attention';
```

- **Observed States**

```
                Input a cell vector containing the symbols (characters)
    in the '_ODEs.txt' file. Eg: to observe deoxyhemoglobin content, blood vol
    and blood flow set simulation.observed_states =   {'q_1','q_3','q_2','v_1'

    simulation.observed_states = {};
```

- **Final time for simulation**

```
                Input a positve real number:

    simulation.final_time = 359*3.22;
```

- **Observation noise**

```
                Input a function handle:

    simulation.state_obs_variance = @(x)
(repmat(bsxfun(@rdivide,var(x),5),size(x,1),1));
```

- **Time interval between observations**

```
                Input a positive real number:

    simulation.interval_between_observations = 0.1;
```

User Input: Estimation

- **Candidate ODEs**

- * Input the ODEs "type" used for estimation as a string. Options: 'nonlinear_forward_modulation_by_attention', 'forward_modulation_and_driven_by_attention'', 'forward_modulation_by_attention', 'backward_modulation_by_attention', 'backward_modulation_and_driven_by_attention', 'absent_modulation', 'absent_attention_input', 'absent_photic_input', 'driven_by_attention', 'photic_input'.

```
candidate_odes = 'nonlinear_forward_modulation_by_attention';
```

- *Kernel parameters *$\mathbf\phi$

```
                Input a row vector of positive real numbers of size 1 x
2:

kernel.param = [10,0.2];
```

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

```
                Input a row vector of positive real numbers of size 1 x
number of ODEs:

state.derivative_variance = 6.*ones(11-3,1);
```

- **Estimation times**

```
                Input a row vector of positive real numbers in ascending
order:

time.est = 0:3.22:359*3.22;
```

Preprocessing for candidate ODEs

```
[symbols,ode,plot_settings,state,simulation,odes_path,coupling_idx,opt_settings]
 = ...
    preprocessing_dynamic_causal_modeling
 (simulation,candidate_odes,state);
```

```
ODEs:

 (d*q_1)/dt == - (5*exp((17*v_1)/8))/8 - (25*exp(-
q_1)*exp(f_1)*((3/5)^exp(-f_1) - 1))/16
 (d*q_3)/dt == - (5*exp((17*v_3)/8))/8 - (25*exp(-
q_3)*exp(f_3)*((3/5)^exp(-f_3) - 1))/16
 (d*q_2)/dt == - (5*exp((17*v_2)/8))/8 - (25*exp(-
q_2)*exp(f_2)*((3/5)^exp(-f_2) - 1))/16
                          (d*v_1)/dt == (5*exp(-v_1)*exp(f_1))/8 -
 (5*exp((17*v_1)/8))/8
                          (d*v_3)/dt == (5*exp(-v_3)*exp(f_3))/8 -
 (5*exp((17*v_3)/8))/8
                          (d*v_2)/dt == (5*exp(-v_2)*exp(f_2))/8 -
 (5*exp((17*v_2)/8))/8
                                                              (d*f_1)/
dt == s_1*exp(-f_1)
                                                              (d*f_3)/
dt == s_3*exp(-f_3)
```

```
                                                                  (d*f_2)/
dt == s_2*exp(-f_2)
                                        (d*s_1)/dt == n_1 - (3*s_1)/5 -
 (8*exp(f_1))/25 + 8/25
                                        (d*s_3)/dt == n_3 - (3*s_3)/5 -
 (8*exp(f_3))/25 + 8/25
                                        (d*s_2)/dt == n_2 - (3*s_2)/5 -
 (8*exp(f_2))/25 + 8/25
                                                  (d*n_1)/dt == a_11*n_1 +
 a_12*n_2 + c_11*u_1
                                                  (d*n_3)/dt == a_32*n_2 +
 a_33*n_3 + c_33*u_3
                    (d*n_2)/dt == a_22*n_2 + a_23*n_3 + n_1*(a_21 +
 d_213*n_3 + b_212*u_2)
```

Simulate Trajectories

- **Preprocessing for true ODEs**

```
   [symbols_true,ode_true] = preprocessing_dynamic_causal_modeling
 (simulation,simulation.odes,state);
```

```
ODEs:

 (d*q_1)/dt == - (5*exp((17*v_1)/8))/8 - (25*exp(-
q_1)*exp(f_1)*((3/5)^exp(-f_1) - 1))/16
 (d*q_3)/dt == - (5*exp((17*v_3)/8))/8 - (25*exp(-
q_3)*exp(f_3)*((3/5)^exp(-f_3) - 1))/16
 (d*q_2)/dt == - (5*exp((17*v_2)/8))/8 - (25*exp(-
q_2)*exp(f_2)*((3/5)^exp(-f_2) - 1))/16
                           (d*v_1)/dt == (5*exp(-v_1)*exp(f_1))/8 -
 (5*exp((17*v_1)/8))/8
                           (d*v_3)/dt == (5*exp(-v_3)*exp(f_3))/8 -
 (5*exp((17*v_3)/8))/8
                           (d*v_2)/dt == (5*exp(-v_2)*exp(f_2))/8 -
 (5*exp((17*v_2)/8))/8
                                                                  (d*f_1)/
dt == s_1*exp(-f_1)
                                                                  (d*f_3)/
dt == s_3*exp(-f_3)
                                                                  (d*f_2)/
dt == s_2*exp(-f_2)
                                        (d*s_1)/dt == n_1 - (3*s_1)/5 -
 (8*exp(f_1))/25 + 8/25
                                        (d*s_3)/dt == n_3 - (3*s_3)/5 -
 (8*exp(f_3))/25 + 8/25
                                        (d*s_2)/dt == n_2 - (3*s_2)/5 -
 (8*exp(f_2))/25 + 8/25
                                                  (d*n_1)/dt == a_11*n_1 +
 a_12*n_2 + c_11*u_1
                                                  (d*n_3)/dt == a_32*n_2 +
 a_33*n_3 + c_33*u_3
```

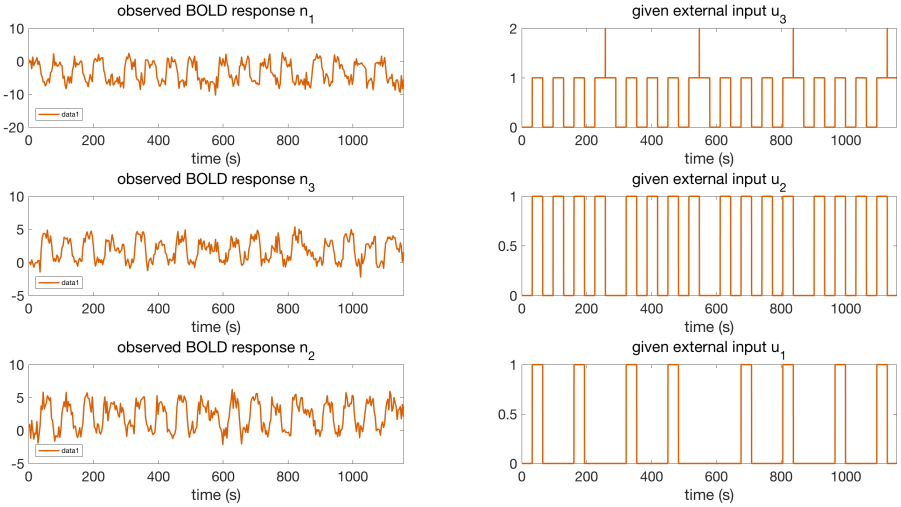
```
                    (d*n_2)/dt == a_22*n_2 + a_23*n_3 + n_1*(a_21 +
 d_213*n_3 + b_212*u_2)


non_diverging_trajectories = false; i = 0;
while ~non_diverging_trajectories
```

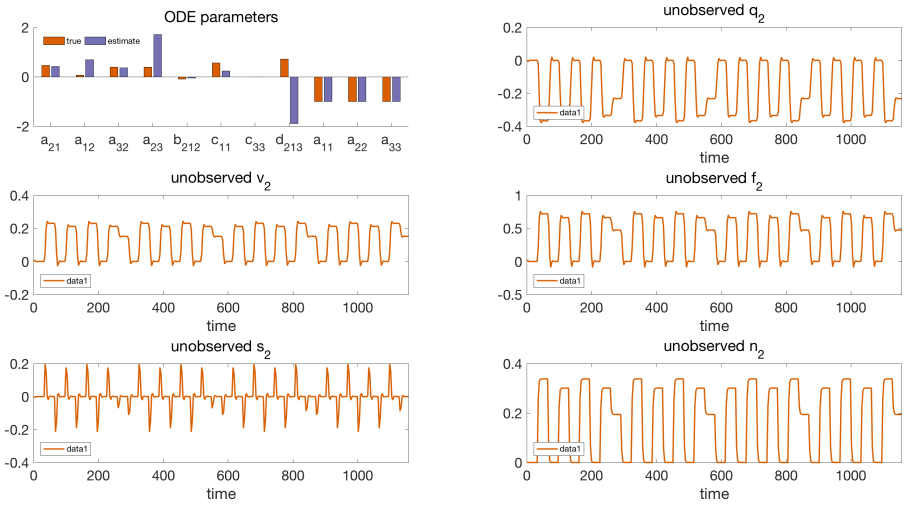
- **Sample ODE parameters**

```
                    non-selfinhibitory neuronal couplings (sampled uniformily in
        the interval $[-0.8,0.8]$);

    simulation.ode_param = -0.8 + (0.8-(-0.8)) *
 rand(1,length(symbols_true.param));

                    self-inhibitory neuronal couplings set to -1.

    simulation.ode_param(end-2:end) = -1;
```

- **Numerical integration**

```
    try
        simulation_old = simulation;
        [simulation,obs_to_state_relation,fig_handle,plot_handle] =
 simulate_state_dynamics_dcm(...

 simulation,symbols_true,ode_true,time,plot_settings,state.ext_input,'plot');
        non_diverging_trajectories = 1;
    end
```





end

Mass Action Dynamical Systems

A deterministic dynamical system is represented by a set of K ordinary differential equations (ODEs) with model parameters $\theta \in \mathcal{R}^d$ 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 $\mathbf{x}$. 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 $\mathcal{M}_{ki} \subseteq \{ 1, \dots, K\}$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).

start timer

```
tic;
```

Prior on States and State Derivatives

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

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

with

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

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

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

$\mathrm{cov}(\dot{x}_k(t), \dot{x}_k(t)) = \frac{\partial^2 C_{\mathbf{\phi}_k}(t,t') }{\partial t \partial t'} =: C_{\mathbf{\phi}_k}''(t,t')$.

```
Error updating Text.

 Character vector must have valid interpreter syntax:
$\mathrm{cov}(\dot{x}_k(t), \dot{x}_k(t)) = \frac{\partialC_{\mathbf\phi_k
```

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:

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

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

where $\mathbf{F} := \mathbf{f}(\mathbf{X}, \theta)$ and $\mathbf{A} := \mathbf{C}''_\phi - {}'\mathbf{C}_\phi \mathbf{C}_\phi^{-1} \mathbf{C}'_\phi$ 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 $\dot{\mathbf{X}}$:

```
Error updating Text.

 Character vector must have valid interpreter syntax:
$\dot{\mathbf{X}}$
```

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

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

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

Rewrite ODEs as Linear Combination in Parameters

Since, according to the mass action dynamics (equation 2), the ODEs are **linear in the parameters ** $\mathbf{\theta}$ we can 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);
```

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:

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

```
Error updating Text.
```

```
 Character vector must have valid interpreter syntax:
$p(\mathbf{\theta} \mid \mathbf{X}, \mathbf{\phi}, \gamma) = \mathcal{N}\l
```

Rewrite Hemodynamic ODEs as Linear Combination in (monotonic functions of) Individual Hemodynamic States

- *Deoxyhemoglobin content *

```
                Rewrite the BOLD signal change equation as a linear combina
in a monotonic function of the deoxyhemoglobin content $$\exp(\mathbf{q})$

                $$\mathbf{R}_{q~\mathbf\lambda} ~ \exp(\mathbf{q}) ~+ ~
\mathbf{r}_{v~\mathbf\lambda} \stackrel{!}{=} ~\mathbf\lambda(\mathbf{q},\

 [state.deoxyhemo.R,state.deoxyhemo.r] =
rewrite_bold_signal_eqn_as_linear_combination_in_deoxyhemo(symbols);
```

- **Blood volume**

```
                Rewrite the deoxyhemoglobin content ODE as a linear combina
in a monotonic function of the blood volume $\exp\left( 17 / 8 ~\mathbf{v}

                 $\mathbf{R}_{v\dot{q}} ~\exp\left( 17 / 8 ~\mathbf{v}\righ
~+~ \mathbf{r}_{v\dot{q}} \stackrel{!}{=} \mathbf{f}_{\dot{q}}(\mathbf{X},

 [state.vol.R,state.vol.r] =
rewrite_deoxyhemo_ODE_as_linear_combination_in_vol(ode,symbols);
```

- **Blood flow**

```
                Rewrite the blood volume ODE as a linear combination in
a monotonic function of the blood flow $\exp(\mathbf{f})$.

                  $$\mathbf{R}_{f~\dot{v}} ~ \exp(\mathbf{f}) + \mathbf{r}
\stackrel{!}{=} \mathbf{f}_{\dot{v}}(\mathbf{X},\mathbf\theta)$$.

 [state.flow.R,state.flow.r] =
rewrite_vol_ODE_as_linear_combination_in_flow(ode,symbols);
```

- **Vasosignalling**

```
                Rewrite the blood flow and vasoginalling ODEs as a linear
combination in vasosignalling $\mathbf{s}$.

                  $$\mathbf{R}_{s\dot{f}} ~\mathbf{s} + \mathbf{r}_{s\dot
\stackrel{!}{=} \mathbf{f}_{\dot{f}}(\mathbf{X},\mathbf\theta)$$.

                  $$\mathbf{R}_{s\dot{s}}~ \mathbf{s} + \mathbf{r}_{s\dot
\stackrel{!}{=} \mathbf{f}_{\dot{s}}(\mathbf{X},\mathbf\theta)$$.
```

```
        [state.vaso.R,state.vaso.r] =
 rewrite_vaso_and_flow_odes_as_linear_combination_in_vaso(ode,symbols);
```

Rewrite Neuronal ODEs as Linear Combination in Individual Neuronal States

We rewrite the ODE(s) $\mathbf{f}_k(\mathbf{X}, \theta)$ as a linear combination in the individual state $\mathbf{n}_u$:

```
$$\mathbf{R}_{uk}  \mathbf{n}_u + \mathbf{r}_{uk} \stackrel{!}{=}
\mathbf{f}_{k}(\mathbf{X},\mathbf{\theta})$$,
```

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 $\mathbf{n}_u$.

```
[state.neuronal.R,state.neuronal.r] =
 rewrite_odes_as_linear_combination_in_ind_neuronal_states(ode,symbols,coupling_id
```

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 $\mathbf{x}_u$:

$$\mathbf{B}_u \mathbf{x}_u + \mathbf{b}_u \stackrel{!}{=} \mathbf{f}(\mathbf{X}, \theta) - {}'\mathbf{C}_\phi \mathbf{C}_\phi^{-1} \mathbf{X} \qquad (7)$$

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

$$\mathbf{x}_u = \mathbf{B}_u^+ \left(\epsilon_\mathbf{0} - \mathbf{b}_u \right),$$

where $\mathbf{B}_u^+$ denotes the pseudo-inverse of $\mathbf{B}_u$. Since $\mathbf{C}_\phi$ is block diagonal we can rewrite the expression above as:

$$\mathbf{x}_u = \left(\mathbf{B}_u \mathbf{B}_u^T \right)^{-1} \mathbf{B}_u^T \sum_k \left({\epsilon_\mathbf{0}}^{(k)} - \mathbf{b}_{uk} \right) \quad = \left(\mathbf{B}_u \mathbf{B}_u^T \right)^{-1} \sum_k \mathbf{B}_{uk}^T \left({\epsilon_\mathbf{0}}^{(k)} - \mathbf{b}_{uk} \right),$$

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 an individual state $\mathbf{x}_u$:

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

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

Mean-field Variational Inference

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

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

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:

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

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:

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

Denoising BOLD Observations

We denoise the BOLD observation by standard GP regression.

```
bold_response.denoised_obs =
 denoising_BOLD_observations(simulation.bold_response{:,
{'n_1','n_3','n_2'}},inv_C,symbols,simulation);
```

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 $p(\mathbf{X} \mid \phi) = \prod_k \mathcal{N}(\mathbf{x}_k; \mathbf{0}, \mathbf{C}_{\phi_k})$:

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

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

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

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)$$ and the proxies for the distribution over individual states $\hat{q}(\mathbf{x}_u)$$.

- **Initialize the state estimation by the GP regression posterior**

```
state.proxy.mean = array2table([time.est',mu],'VariableNames',
['time',symbols.state_string]);
bold_response.obs_old = bold_response.denoised_obs;
ode_param.proxy.mean = zeros(length(symbols.param),1);
```

- **Coordinate ascent**

```
for i = 1:opt_settings.coord_ascent_numb_iter
```

*** ***

Intercept due to Confounding Effects

The intercept is determined by a minimum least squares estimator:

$$\mathbf{X}\hat{\beta} := \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{y} - \lambda(\mathbf{q}, \mathbf{v}))$$.

```
bold_response_signal_change = bold_signal_change_eqn(state.
    state.proxy.mean{:,'q_1','q_2','q_3'});
bold_response.confounding_effects.intercept = determine_int
    bold_response.confounding_effects.X0,bold_response.conf
```

```
bold_response.confounding_effects.intercept =
 zeros(height(simulation.bold_response),width(simulation.bold_response)-1);
bold_response.denoised_obs = bold_response.obs_old -
 bold_response.confounding_effects.intercept;
```

Proxy for Hemodynamic States

Determine the proxies for the states, starting with deoxyhemoglobin followed by blood volume, blood flow and finally vasosignalling. The information flow in the hemodynamic system is shown in its factor graph below:

.

The model inversion in the hemodynmic factor graph above occurs locally w.r.t. individual states. Given the expression for the BOLD signal change equation, we invert the BOLD signal change equation analytically to determine the deoxyhemoglobin content $\mathbf{q}$$ (1). The newly inferred deoxyhemoglobin content $\mathbf{q}$$ influences the expression for the factor associated with the change in deoxyhemoglobin content $\mathbf{h}_{\dot{q}}$$,

which we subsequently invert analytically to infer the blood volume $\mathbf{v}$ (2). Thereafter, we infer the blood flow $\mathbf{f}$ (3) by inverting the factors associated with the change in blood volume $\mathbf{h}_{\dot{v}}$ as well as vasosignalling $\mathbf{h}_{\dot{s}}$, followed by inferring vasosignalling $\mathbf{s}$ (4) by inverting the factors associated with blood flow induction $\mathbf{h}_{\dot{f}}$ and vasosignalling $\mathbf{h}_{\dot{s}}$. Finally, the neuronal dynamics (5) are learned, in part, by inverting the factor associated with vasosignalling $\mathbf{h}_{\dot{s}}$. The typical trajectories of each of the states are shown (red) together with their iterative approximation (grey lines) obtained by graphical DCM.

- **Proxy for deoxyhemolgobin content**

```
                    Damping is required since we invert only the factor for
       the BOLD signal change equation w.r.t. a monotonic function of deoxyhemogl
       content $\exp( \mathbf{q})$.

                                * Undamped proxy:*

              state_proxy_undamped =
 proxy_for_deoxyhemoglobin_content(state.deoxyhemo,state.proxy.mean{:,symbols.stat

 bold_response.denoised_obs,symbols,A_plus_gamma_inv,opt_settings);

                                * Damped proxy:*

              state.proxy.mean{:,{'q_1','q_3','q_2'}} = (1-
opt_settings.damping) * state.proxy.mean{:,{'q_1','q_3','q_2'}} + ...
                    opt_settings.damping * state_proxy_undamped;
```

- **Proxy for blood volume**

- * Damping is required since we invert only the a subset of ODEs w.r.t. a monotonic function of blood volume $\exp\left(17/8\ \mathbf{v}\right)$.

- Undamped proxy:*

```
              state_proxy_undamped =
 proxy_for_blood_volume(state.vol,dC_times_invC,state.proxy.mean{:,symbols.state_s

 ode_param.proxy.mean,symbols,A_plus_gamma_inv,opt_settings);

                                *Damped proxy:*

              state.proxy.mean{:,{'v_1','v_3','v_2'}} = (1-
opt_settings.damping) * state.proxy.mean{:,{'v_1','v_3','v_2'}} + ...
                    opt_settings.damping * state_proxy_undamped;
```

- **Proxy for blood flow**

```
                    Damping is required since we invert only the a subset of
       ODEs w.r.t. a mononic function of blood flow $\exp(\mathbf{f})$.

                                *Undamped proxy:*

              state_proxy_undamped =
 proxy_for_blood_flow(state.flow,dC_times_invC,state.proxy.mean{:,symbols.state_st

 ode_param.proxy.mean,symbols,A_plus_gamma_inv,opt_settings);
```

```
                                        *Damped proxy:*

                    state.proxy.mean{:,{'f_1','f_3','f_2'}} = (1-
opt_settings.damping) * state.proxy.mean{:,{'f_1','f_3','f_2'}} + ...
                        opt_settings.damping * state_proxy_undamped;
```

- **Proxy for vasosignalling**

```
                          No damping is required because we invert all ODEs w.r.t.
        vasosingalling $\mathbf{s}$.

                    state.proxy.mean{:,{'s_1','s_3','s_2'}} =
 proxy_for_vasosignalling(state.vaso,dC_times_invC,...

 state.proxy.mean{:,symbols.state_string},ode_param.proxy.mean,symbols,A_plus_gamm
```

Proxy for Neuronal States

- *Determine the proxies for the neuronal states. An example of the information flow in the neuronal part of the nonlinear forward modulating (nonlin_fwd_mod) is shown in its factor graph below:

.

```
                     In the neuronal factor graph (for the nonlinear forwad modulati
        above each individual state appears linear in every factor in the neuronal
        We can therefore analytically invert every factor to determine the neurona
        state. The typical trajectories of each of the states are shown (red) toge
        with their iterative approximation (grey lines) obtained by variational gr
        matching.
```

No damping is required because we invert all ODEs w.r.t. neuronal populations $\mathbf{n}$.

```
                    state.proxy.mean{:,{'n_1','n_3','n_2'}} =
 proxy_for_neuronal_populations(state.neuronal,...

 state.proxy.mean{:,symbols.state_string},ode_param.proxy.mean',dC_times_invC,...

 coupling_idx.states,symbols,A_plus_gamma_inv,opt_settings);
```

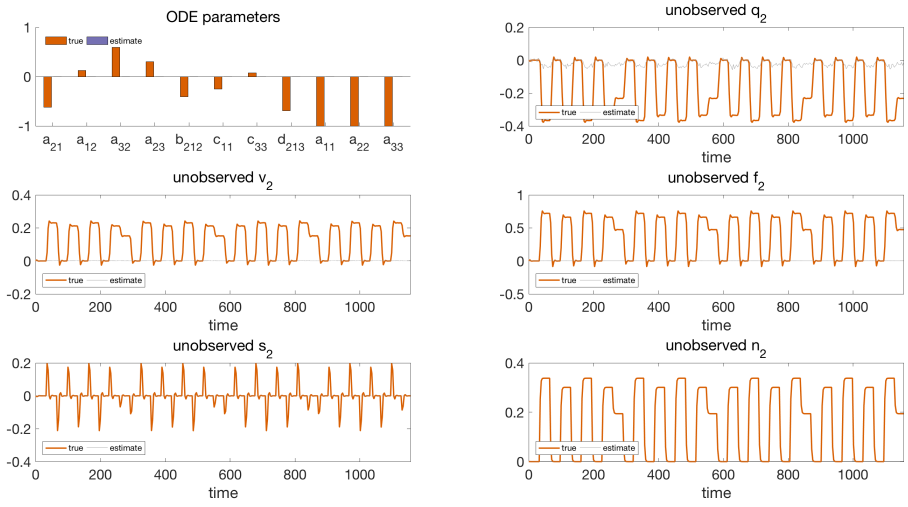
Keep initial value at zero

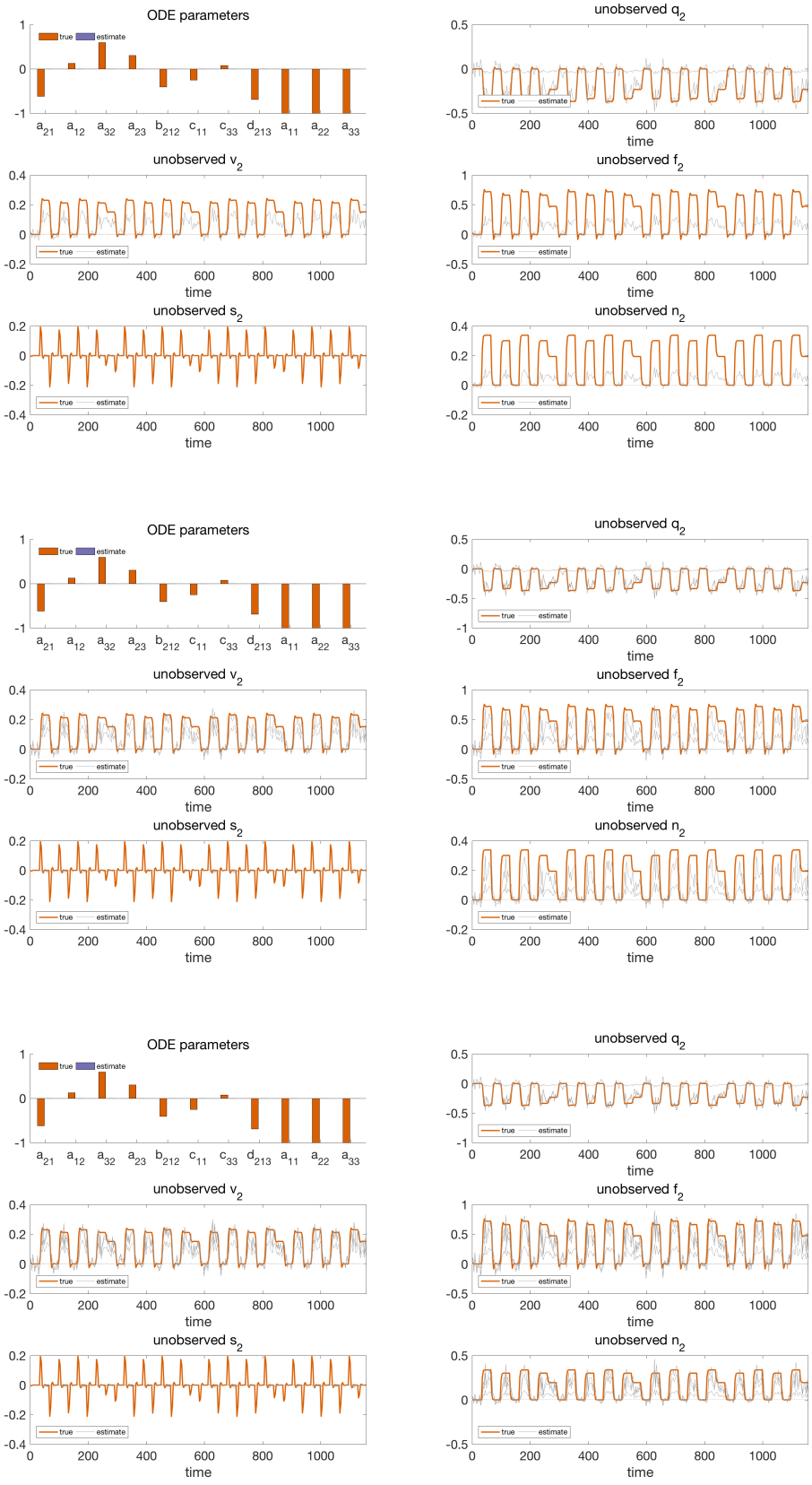
```
                    state_idx = cellfun(@(x)
 ~strcmp(x(1),'u'),symbols.state_string);
                    state.proxy.mean{1,state_idx} =
 bsxfun(@minus,state.proxy.mean{1,symbols.state_string(state_idx)},...

 state.proxy.mean{1,symbols.state_string(state_idx)});
```

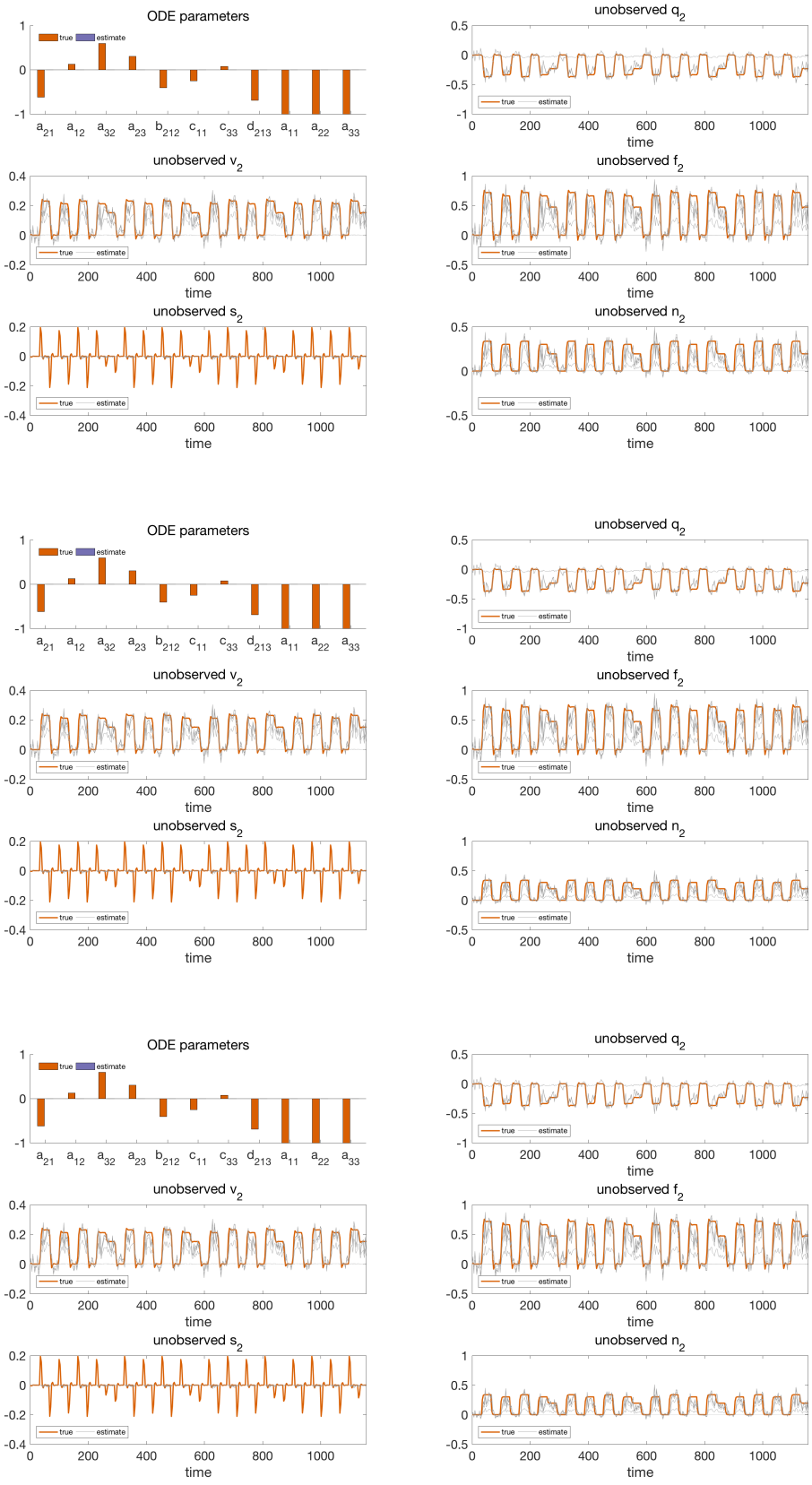
*** ***

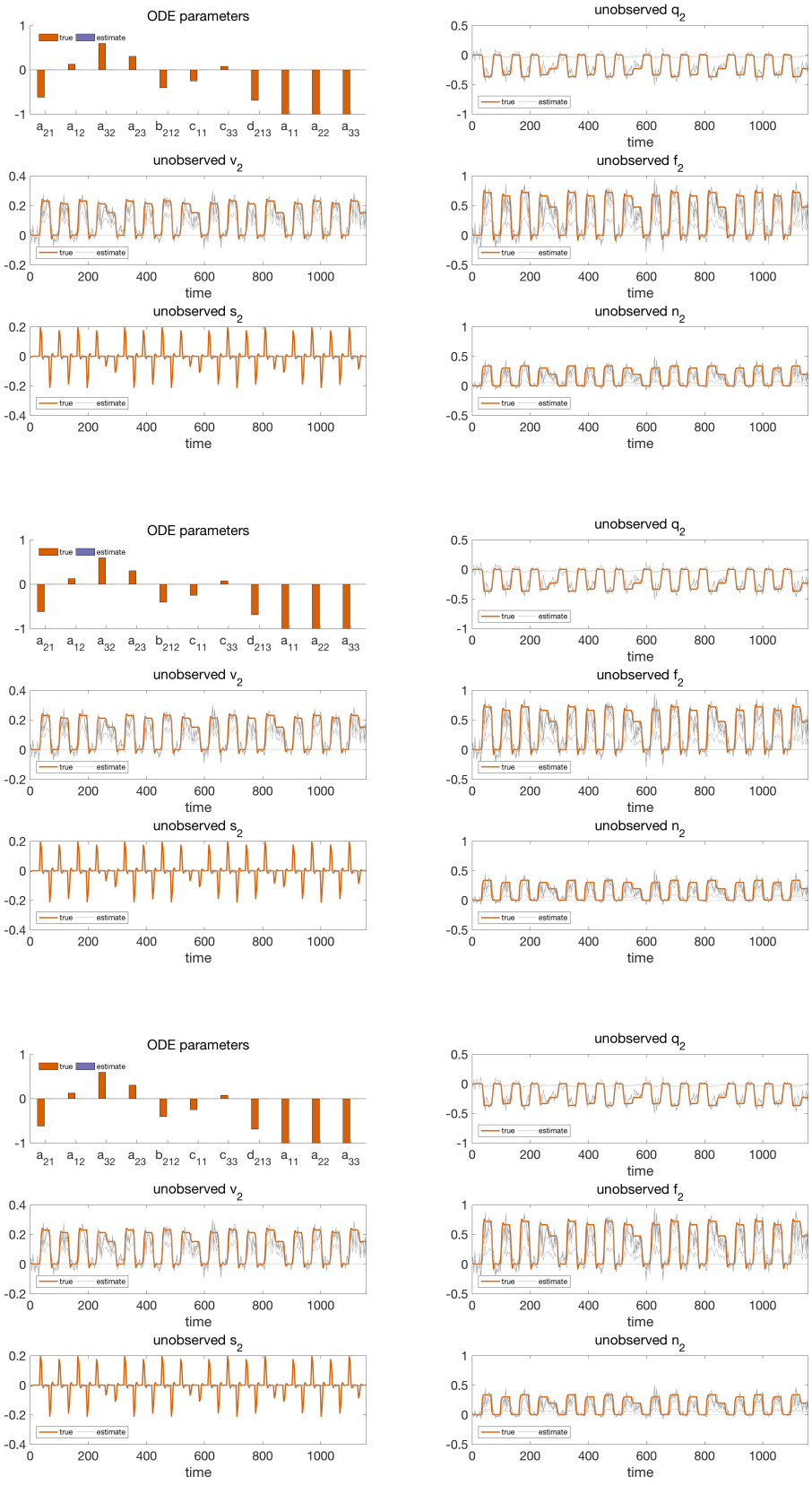
Proxy for ODE parameters

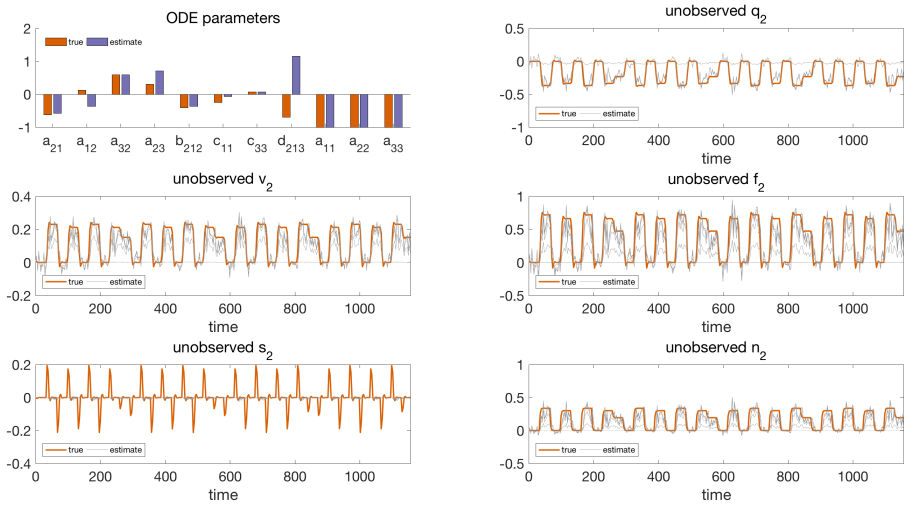
```
                        Expanding the proxy distribution in equation (11) for
        $$\mathbf{\theta}$$ yields:

                                    $\hat{q}(\mathbf{\mathbf\theta}) \propto \exp
        \left( ~E_{Q_{-\mathbf\theta}}     \ln p(\theta \mid \mathbf{X},\mathbf{Y}
        ~     \right) \\ \qquad= \exp \left( ~E_{Q_{-\mathbf{\theta}}} \ln \mathca
        ; \left(    \mathbf{B}_{\mathbf{\theta}}^T \mathbf{B}_{\mathbf{\theta}} \r
        \left( \sum_k    \mathbf{B}_{\mathbf{\theta} k}^T ~ \left( {'\mathbf{C}_{\
        k}}    \mathbf{C}_{\mathbf{\phi} k}^{-1} \mathbf{X}_k - \mathbf{b}_{\mathb
        k} \right)    \right), ~ \mathbf{B}_{\mathbf{\theta}}^+ ~ (\mathbf{A} + \m
        ~    \mathbf{B}_{\mathbf{\theta}}^{+T} \right) ~\right)$,

                        where we substitute $$p(\mathbf{\theta} \mid \mathbf{X}
        with its density given in equation (6).

                        No damping is required because we invert all ODEs w.r.t
        neuronal couplings $\mathbf{\theta}$.

            if i>200 || i==opt_settings.coord_ascent_numb_iter
                [ode_param.proxy.mean,ode_param.proxy.inv_cov] =
proxy_for_ode_parameters(...

state.proxy.mean{:,symbols.state_string},dC_times_invC,ode_param.lin_comb,...
                    symbols,A_plus_gamma_inv,opt_settings);
            end

            if i==1 || ~mod(i,20)

plot_results(fig_handle,state,simulation,ode_param.proxy.mean,plot_handle,symbols
            end
```




ODE parameters
true
estimate
a21 a12 a32 a23 b212 c11 c33 d213 a11 a22 a33
unobserved q2
unobserved v2
unobserved f2
unobserved s2
unobserved n2
time


ODE parameters
true
estimate
a21 a12 a32 a23 b212 c11 c33 d213 a11 a22 a33
unobserved q2
unobserved v2
unobserved f2
unobserved s2
unobserved n2
time


ODE parameters
true
estimate
a21 a12 a32 a23 b212 c11 c33 d213 a11 a22 a33
unobserved q2
unobserved v2
unobserved f2
unobserved s2
unobserved n2
time



end

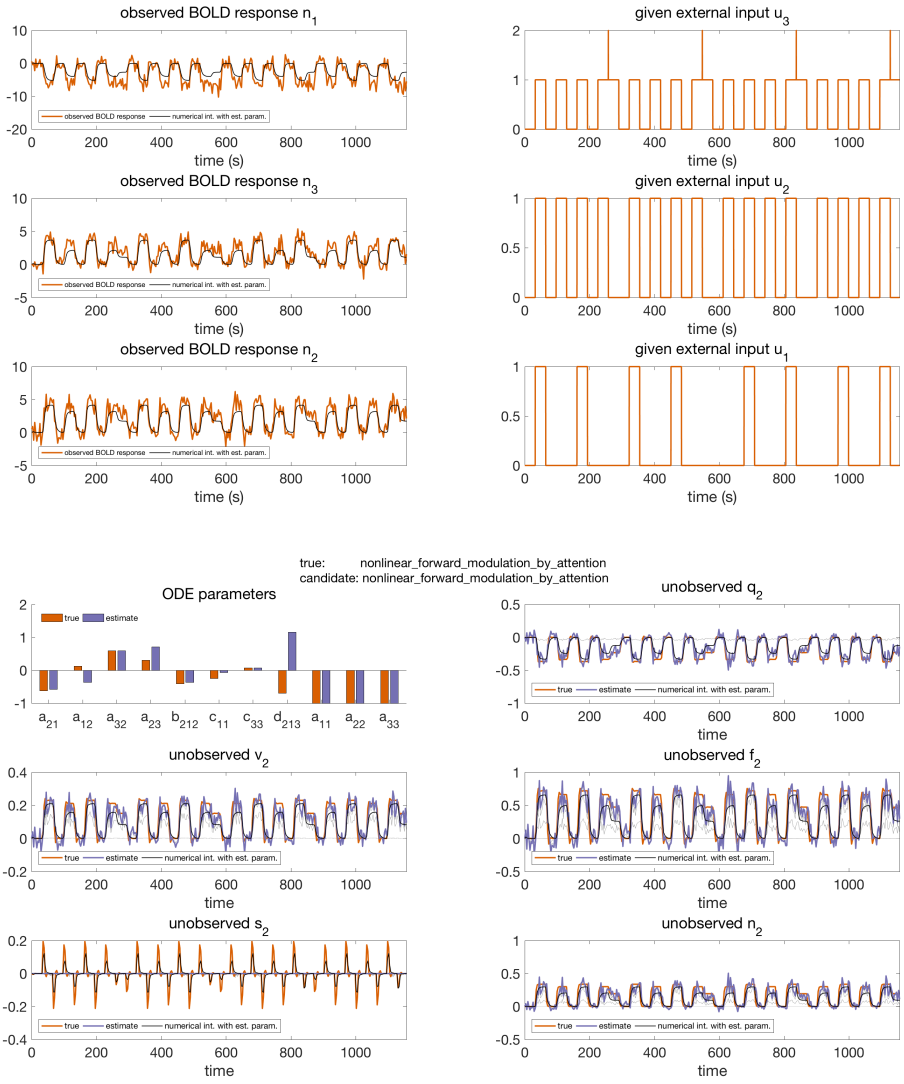
Numerical Integration with Estimated ODE Parameters

See whether we actually fit the BOLD responses well. Curves are shown in black.

```
simulation2 = simulation_old; simulation2.ode_param =
 ode_param.proxy.mean';
[simulation2,obs_to_state_relation] =
 simulate_state_dynamics_dcm(simulation2,symbols,ode,time,...
    plot_settings,state.ext_input,'no plot');
state.proxy.num_int = simulation2.state;
```

- **Final result**

```
plot_results(fig_handle,state,simulation,ode_param.proxy.mean,plot_handle,symbols

plot_settings,'final',simulation2.bold_response_true,simulation.odes,candidate_od
```



Time Taken

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

```
time taken: 69.5508 seconds
```

References

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

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

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). Link to arxiv paper here.

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.

Published with MATLAB® R2017a