Variational Gaussian Process Timeseries Inference

Carl Edward Rasmussen

May 7th 2017

Abstract

Variational inference in nonlinear dynamical models.

The model is specified by

$$\begin{aligned} f_e(\tilde{x}) &\sim \mathcal{GP}\big(0, k_e(\cdot,\cdot)\big), && \text{where } \tilde{x} = (x,u) \text{ and } e = 1,\dots,E, && (1)\\ x_t|f_t &\sim \mathcal{N}(x_t|f_t, Q), && Q \text{ diagonal}, && (2)\\ y_t|x_t &\sim \mathcal{N}(y_t|Cx_t, R), && && (3)\end{aligned}$$

and $u_t, y_t, t = 1, \dots, T$ are the control inputs and measurements (both observed), and $f_t, t = 2, \dots, T$ and $x_t, t = 1, \dots, T$ are unobserved, latent variables. The GPs implement the non-linear transition from one time point to the next conditioned on the state x_{t-1} and all the previous transition pairs $f_{2:t-1}, x_{1:t-2}$

$$f_t(x_{t-1}) = p(f_t|f_{2:t-1}, x_{1:t-1}), \text{ where } t = 2, \dots, T.$$

The joint probability of all the variables is given by the product of T observation probabilities and $T-1$ transition probabilities

$$p(y,x,f) = \prod_{t=1}^{T} p(y_t|x_t) \prod_{t=2}^{T} p(x_t|f_t)p(f_t|f_{1:t-1}, x_{1:t-1}).$$

Each GP is augmented with a set of M inducing inputs z and corresponding targets v such that $v_e = f_e(z_e)$. The augmented joint is

$$p(y,x,f,v) = p(y|x)p(x,f|v)p(v).$$

Exact inference in the model is intractable, instead we fit the model by optimizing a variational lower bound based on an approximating distribution q, which we chose to have the following form

$$q(x,f,v) = q(v)q(x)\prod_{t=2}^{T} p(f_t|f_{1:t-1}, x_{1:t-1}, v), \text{ where } q(x) = \mathcal{N}(\mu_x, \Sigma_x),$$

the assumptions being that 1) the joint on v and x factorizes, 2) that $q(x)$ is Gaussian and 3) that the conditional $q(f|x,v)$ is chosen to be equal to the conditional prior. Generally, we would expect the variational bound to be tight if the approximating distribution is close to the posterior, but for tractability we are forced to set the conditional $q(f|x,v)$ to be equal to the conditional prior. This may still be a good approximation, since we are conditioning on the inducing targets v. If the inducing targets are able to capture the properties of the posterior, then the bound may still be good.

The variational log marginal likelihood lower bound is a single time series (contributions for multiple time series are simply added together)

$$\begin{aligned}\mathcal{L}(y|q(v), q(x), \theta) = &-\mathrm{KL}(q(v)||p(v)) + H(q(x)) + \sum_{t=1}^{T}\langle \log p(y_t|x_t)\rangle_{q(x_t)}\\ &+ \sum_{t=2}^{T} -\tfrac{1}{2}\mathrm{tr}(Q^{-1}\langle B_{t-1}\rangle_{q(x_{t-1})}) + \langle \log \mathcal{N}(x_t|A_{t-1}v, Q)\rangle_{q(v), q(x_{t-1:t})},\end{aligned} \quad (4)$$

with the following definitions

$$A_{t-1} = k(x_{t-1}, z)K^{-1}, \text{ and } B_{t-1} = k(x_{t-1}, x_{t-1}) - k(x_{t-1}, z)K^{-1}k(z, x_{t-1}).$$

A free form optimization of this bound wrt $q(v)$ yields independent Gaussians for each GP

$$q^*(v_e) = \mathcal{N}\big(\mu_e = K_e(K_e+\Psi_{2e})^{-1}\Psi_{1e},\ \Sigma_e = K_e(K_e+\Psi_{2e})^{-1}K_e\big), \tag{5}$$

where $K_e = k_e(z_e, z_e)$ and we have defined the expectations

$$\Psi_1 = \sum_{t=2}^{T} \langle k(z,x_{t-1})Q^{-1}x_t\rangle_{q(x_{t-1:t})}, \quad \text{and} \quad \Psi_2 = \sum_{t=2}^{T} \langle k(z,x_{t-1})Q^{-1}k(x_{t-1},z)\rangle_{q(x_{t-1})}, \tag{6}$$

of size $M \times E$ and $M \times M \times E$ respectively.
Plugging the optimal $q^*(v)$ back into the bound eq. (4), we get

$$\begin{aligned}\mathcal{L}(y|q(x),\theta) = &- \mathrm{KL}(q^*(v)\|p(v)) + H(q(x)) + \sum_{t=1}^{T}\langle \log p(y_t|x_t)\rangle_{q(x_t)} \\ &+ \sum_{t=2}^{T} -\tfrac{1}{2}\langle \mathrm{tr}\big(Q^{-1}(B_{t-1} + A_{t-1}\Sigma A_{t-1})\big)\rangle_{q(x_{t-1})} + \langle \log \mathcal{N}(x_t|A_{t-1}\mu, Q)\rangle_{q(x_{t-1:t})}.\end{aligned} \tag{7}$$

Note that except for the entropy $H(q(x))$, the bound only depends on $q(x)$ through its pair-wise marginals. This means that the model will be identical for all $q(x)$ which have the same pair-wise marginals, except for an offset in the bound which depends on the entropy. We will chose $q(x)$ to be Markovian, ie the precision Σ_x^{-1} is block tri-diagonal.

Transition model

Writing out each term from the transition model from eq. (7) in detail

$$-\mathrm{KL}(q^*(v)\|p(v)) = -\tfrac{1}{2}\sum_{e=1}^{E} \mathrm{tr}(K_e+\Psi_{2e})^{-1}K_e + \mu_e^\top K_e^{-1}\mu_e - M - \log|(K_e+\Psi_{2e})^{-1}K_e|, \tag{8}$$

and

$$\begin{aligned}-\tfrac{1}{2}\sum_{t=2}^{T}\mathrm{tr}Q^{-1}\langle B_{t-1}\rangle_{q(x_{t-1})} &= -\tfrac{T-1}{2}\mathrm{tr}Q^{-1} + \tfrac{1}{2}\sum_{t=2}^{T}\mathrm{tr}K^{-1}\langle k(z,x_{t-1})Q^{-1}k(x_{t-1},z)\rangle_{q(x_{t-1})} \\ &= \tfrac{1}{2}\sum_{e=1}^{E}\mathrm{tr}K_e^{-1}\Psi_{2e} - \tfrac{T-1}{2}\mathrm{tr}Q^{-1},\end{aligned} \tag{9}$$

and

$$\begin{aligned}-\tfrac{1}{2}\sum_{t=2}^{T}\mathrm{tr}Q^{-1}\langle A_{t-1}\Sigma A_{t-1}\rangle_{q(x_{t-1})} &= -\tfrac{1}{2}\sum_{t=2}^{T}\mathrm{tr}\big(\Sigma K^{-1}\langle k(x_{t-1},z)Q^{-1}k(z,x_{t-1})\rangle_{q(x_{t-1})}K^{-1}\big) \\ &= -\tfrac{1}{2}\sum_{e=1}^{E}\mathrm{tr}(K_e+\Psi_{2e})^{-1}\Psi_{2e},\end{aligned} \tag{10}$$

and

$$\begin{aligned}&\sum_{t=2}^{T}\langle \log\mathcal{N}(x_t|A_{t-1}\mu, Q)\rangle_{q(x_{t-1:t})} \\ &\quad= -\tfrac{(T-1)E}{2}\log(2\pi) - \tfrac{T-1}{2}\log|Q| - \tfrac{1}{2}\sum_{t=2}^{T}\langle (x_t - A_{t-1}\mu)^\top Q^{-1}(x_t - A_{t-1}\mu)\rangle_{q(x_{t-1:t})} \\ &\quad= -\tfrac{(T-1)E}{2}\log(2\pi) - \tfrac{T-1}{2}\log|Q| - \tfrac{1}{2}\mathrm{tr}Q^{-1}\sum_{t=2}^{T}\langle x_t^\top x_t\rangle_{q(x_t)} + \mu^\top\langle x_t Q^{-1}A_{t-1}\rangle_{q(x_{t-1:t})} \\ &\qquad\qquad - \tfrac{1}{2}\mu^\top K^{-1}\langle k(x_{t-1},z)Q^{-1}k(z,x_{t-1})\rangle_{q(x_{t-1})}K^{-1}\mu \\ &\quad= -\tfrac{(T-1)E}{2}\log(2\pi) - \tfrac{T-1}{2}\log|Q| - \tfrac{1}{2}\mathrm{tr}Q^{-1}\sum_{t=2}^{T}\langle x_t^\top x_t\rangle_{q(x_t)} - \tfrac{1}{2}\mu^\top K^{-1}\Psi_2K^{-1}\mu + \mu^\top K^{-1}\Psi_1.\end{aligned} \tag{11}$$

Collecting terms form eq. (8-11) which depend on Ψ_1 and Ψ_2, two possibilies arise, either training or testing, in both cases we introduce μ and Σ from eq. (5) for the training cases, giving rise to

$$\begin{aligned}
\Psi &= \tfrac{1}{2}\sum_{e=1}^{E} \log|K_e| - \log|K_e+\Psi_{2e}| + \mathrm{tr}K_e^{-1}\Psi_{2e} + \Psi_{1e}^\top(K_e+\Psi_{2e})^{-1}\Psi_{1e},\\
\Psi^* &= \tfrac{1}{2}\sum_{e=1}^{E} \mathrm{tr}K_e^{-1}\Psi_{2e}(K_e+\Psi_{2e})^{-1}\Psi_{2e}^* - \mathrm{tr}(K_e+\Psi_{2e})^{-1}\Psi_{1e}\Psi_{1e}^\top(K_e+\Psi_{2e})^{-1}\Psi_{2e}^* + 2\Psi_{1e}^\top(K_e+\Psi_{2e})^{-1}\Psi_{1e}^*
\end{aligned} \tag{12}$$

for training and test respectively.

Pulling together all terms from eq. (8-11) we get the following contribution

$$\begin{aligned}
&\tfrac{1}{2}\sum_{e=1}^{E} \log|(K_e+\Psi_{2e})^{-1}K_e| + \mathrm{tr}K_e^{-1}\Psi_{2e} + \Psi_{1e}^\top(K_e+\Psi_{2e})^{-1}\Psi_{1e}\\
&-\tfrac{1}{2}\mathrm{tr}Q^{-1}\sum_{t=2}^{T}(I+\mu_t^\top\mu_t+\Sigma_{t,t}) - \tfrac{T-1}{2}\log|Q| - \tfrac{(T-1)E}{2}\log(2\pi).
\end{aligned} \tag{13}$$

Entropy

The entropy of Markovian Gaussian with specified E dimensional marginals and 2E dimensional consequtive pair-wise marginals and marginals is given by

$$\mathcal{H}(q(x)) = \tfrac{TE}{2}(1+\log(2\pi)) + \tfrac{1}{2}\sum_{t=2}^{T}\log|\Sigma_{t-1:t,t-1:t}| - \tfrac{1}{2}\sum_{t=2}^{T-1}\log|\Sigma_t| \tag{14}$$

3 ⟨entropy 3⟩≡ (5b)

```
function [L dLd dLo] = gaussMarkovEntropy(d, o);
[E, E, T] = size(d); dd = zeros(T,1); dp = zeros(T-1,1);
for t = 1:T, dd(t) = det(d(:,:,t)); end                        % det of diagonals
for t = 1:T-1, dp(t) = dd(t)*det(d(:,:,t+1)-o(:,:,t)'/d(:,:,t)*o(:,:,t)); end;
L = E*T*(1+log(2*pi))/2 + sum(log(dp))/2 - sum(log(dd(2:T-1)))/2;     % entropy
if nargout > 1                                                 % want derivatives?
  dLd = zeros(E,E,T); dLo = zeros(E,E,T-1);
  for t = 1:T-1
    dLd(:,:,t) = dLd(:,:,t) + inv(d(:,:,t)-o(:,:,t)/d(:,:,t+1)*o(:,:,t)')/2;
    dLd(:,:,t+1) = inv(d(:,:,t+1)-o(:,:,t)'/d(:,:,t)*o(:,:,t))/2;
    dLo(:,:,t) = -d(:,:,t)\o(:,:,t)/(d(:,:,t+1)-o(:,:,t)'/d(:,:,t)*o(:,:,t));
  end
  for t = 2:T-1, dLd(:,:,t) = dLd(:,:,t) - inv(d(:,:,t))/2; end
end
```

Likelihood

The linear Gaussian log likelihood is

$$\sum_{t=1}^{T}\langle\log p(y_t|x_t)\rangle_{q(x_t)} = -\tfrac{DT}{2}\log(2\pi) - \tfrac{T}{2}\log|R| - \tfrac{1}{2}\mathrm{tr}R^{-1}\sum_{t=1}^{T}\big((y-C\mu_t)(y-C\mu_t)^\top + C\Sigma_tC^\top\big). \tag{15}$$

Maximizing the log likelihood wrt observation noise covariance R and the parameters C yields:

$$R^* = \tfrac{1}{T}\Big[\sum_{t=1}^{T}y_ty_t^\top - C^*\sum_{t=1}^{T}\mu_ty^\top\Big], \quad\text{and}\quad C^* = \sum_{t=1}^{T}y_t\mu_t^\top\Big[\sum_{t=1}^{T}\mu_t\mu_t^\top+\Sigma_{t,t}\Big]^{-1}, \tag{16}$$

and the maximum attained is

$$\mathcal{L}^* = -\tfrac{DT}{2}(1+\log(2\pi)) - \tfrac{T}{2}\log|R^*|, \tag{17}$$

with derivatives

$$\frac{\mathcal{L}^*}{\partial\mu_t} = C^\top R^{-1}(y_t-C\mu_t), \quad\text{and}\quad \frac{\mathcal{L}^*}{\partial\Sigma_{t,t}} = -\tfrac{1}{2}C^\top R^{-1}C, \quad\text{evaluated at}\quad C=C^*, \quad\text{and}\quad R=R^*. \tag{18}$$

In the test case, when we are inferring the latent space representation of a given short sequence, we use the R* and C* parameters derived from the training sequences, so that the contribution to the log likelihood does not take on the simple form $\mathcal{L}^*$, but must be calculated in full from eq. (16).

4a ⟨likelihood 4a⟩≡ (5b)

```
function [L, R_out, C_out, dm, dS] = likelihood(y, qx, test)
persistent C R
D = size(y{1},2); E = size(qx(1).m,1); T = sum(arrayfun(@(x)size(x.m,2),qx));
N = size(y,2); yy = zeros(D); ym = zeros(D, E+1); mm = zeros(E+1); d = zeros(1,N);
for n = 1:N
  m = [qx(n).m' ones(size(qx(n).m,2),1)]; mm = mm + m'*m; ym = ym + y{n}'*m;
  yy = yy + y{n}'*y{n}; mm(1:E,1:E) = mm(1:E,1:E) + sum(qx(n).Sd,3);
end
if test %Use provided C, R
  for n=1:N
    w = y{n}' - C*[qx(n).m; ones(1,size(qx(n).m,2))];
    d(1,n) = sum(sum(inv(R).*(w*w' + C(:,1:E)*sum(qx(n).Sd,3)*C(:,1:E)')));
  end
  L = -D*T*log(2*pi)/2 - T*sum(log(diag(chol(R))))/2 - sum(d)/2;
else
  C = ym/mm; R = (yy - C*ym')/T;
  L = -D*T*(1+log(2*pi))/2 - T*sum(log(diag(chol(R))));          % log likelihood
end
if nargout > 3                                              % do we want derivatives?
  dm = cell(N,1);
  for n = 1:N,
    dm{n} = C(:,1:E)'/R*(y{n}'-C*[qx(n).m; ones(1,size(qx(n).m,2))]);
  end
  dS = -C(:,1:E)'/R*C(:,1:E)/2;            % all dS identical, return once only
  C_out = C; R_out = R;
end
```

The lower bound

Pulling together all terms

$$\begin{aligned}\mathcal{L}(y|q(x),\theta) \;=\; & \tfrac{1}{2}\sum_{e=1}^{E}\log|(K_e+\Psi_{2e})^{-1}K_e| + \mathrm{tr}K_e^{-1}\Psi_{2e} + \Psi_{1e}^\top(K_e+\Psi_{2e})^{-1}\Psi_{1e} + \tfrac{1}{2}\sum_{t=2}^{T}\log|\Sigma_{t-1:t,t-1:t}| \\ & -\tfrac{1}{2}\sum_{t=2}^{T-1}\log|\Sigma_t| - \tfrac{1}{2}\mathrm{tr}Q^{-1}\sum_{t=2}^{T}(I+\mu_t^\top\mu_t+\Sigma_{t,t}) - \tfrac{T-1}{2}\log|Q| - \tfrac{T}{2}\log|R^*| - \tfrac{(D-E)T}{2} - \tfrac{TD-E}{2}\log(2\pi).\end{aligned} \tag{19}$$

4b ⟨lower bound 4b⟩≡ (5b)

```
[L1, dnlml] = Psi(hyp, qx, z, u, test);
T = sum(arrayfun(@(x)size(x.m,2),qx)); L2 = 0; L3 = 0;
dLd = cell(1,N); dLo = cell(1,N);
for n = 1:N
  L2 = L2 + sum(qx(n).m(:,2:end).^2,2) + diag(sum(qx(n).Sd(:,:,2:end),3));
  [L dLd{n} dLo{n}] = gaussMarkovEntropy(qx(n).Sd, qx(n).So); L3 = L3 + L;
end
L5 = -exp(-2*[hyp(:).pn]) * (L2+T-N) / 2;
L4 = -(T-N)*sum([hyp(:).pn])-(T-N)*E*log(2*pi)/2;
[L6, R, C, dm, dS] = likelihood(y, qx, test);
nlml = -L1-L5-L3-L4-L6;
```

4c ⟨bound derivatives 4c⟩≡ (5b)

```
for e = 1:E
  dnlml.hyp(e).pn = ~test*dnlml.hyp(e).pn - exp(-2*hyp(e).pn)*(L2(e)+T-N)+T-N;
end
iQ = diag(exp(-2*[hyp(:).pn]));
for n = 1:N
  dnlml.qx(n).m(:,2:end) = dnlml.qx(n).m(:,2:end) + iQ*qx(n).m(:,2:end);
  dnlml.qx(n).m = dnlml.qx(n).m - dm{n};
  dnlml.qx(n).Sd(:,:,2:end) = bsxfun(@plus, dnlml.qx(n).Sd(:,:,2:end), iQ/2);
  dnlml.qx(n).Sd = dnlml.qx(n).Sd - bsxfun(@plus, dS, dLd{n});
  dnlml.qx(n).So = dnlml.qx(n).So - dLo{n};
end

out1 = nlml; out2 = dnlml; out3 = struct('C', C, 'R', R);    % rename outputs
```

5a ⟨predictions 5a⟩≡

```
[Psi1, Psi2] = Psi(hyp, qx, z, u);
```

5b ⟨vgpt.m 5b⟩≡

```
function [out1, out2, out3] = vgpt(p, data, x);
⟨usage 5c⟩
if nargin == 2
  N = length(p.qx); z = p.z; [M, F, E] = size(z); D = size(data(1).y,2); hyp = p.hyp;
  u = arrayfun(@(x)(x.u),data,'UniformOutput',false); [qx(1:N).m] = deal(p.qx(:).m);
  y = arrayfun(@(x)(x.y),data,'UniformOutput',false);
  for n = 1:N, [qx(n).Sd qx(n).So] = convert(p.qx(n).s); end % convert covariance
  test = 0;
  predict = 0;
elseif isfield(data, 'y')%We are in the test regime, inferring the relevant latent states
  z = x.z; [M, F, E] = size(z); hyp = x.hyp;
  u = arrayfun(@(x)(data.u),data,'UniformOutput',false);
  N = length(p); [qx(1:N).m] = deal(p(:).m);
  y = arrayfun(@(x)(x.y),data,'UniformOutput',false);
  for n = 1:N, [qx(n).Sd qx(n).So] = convert(p(n).s); end % convert covariance
  test = 1;
  predict = 0;
else %We are in the case where we are predicting the next latent space point.
  N = length(p); z = x.z; [M, F, E] = size(z); hyp = x.hyp; qx = p;
  u = arrayfun(@(x)(x.u),data,'UniformOutput',false);
  test = 1;
  predict = 1;
end
if test & predict
   out1 = predict_forwards(hyp, qx, z, u); out2 = []; out3 = [];
else
  ⟨lower bound 4b⟩
  ⟨bound derivatives 4c⟩
  ⟨assign outs 10c⟩
end


⟨Psi 6⟩
⟨entropy 3⟩
⟨likelihood 4a⟩
⟨convert 10a⟩
⟨revert 10b⟩
⟨maha 10d⟩
⟨prediction 11⟩
```

5c ⟨usage 5c⟩≡ (5b)

```
% Variational GP Timeseries inference. Compute the nlml lower bound and its
% derivative wrt hyp hyperparameters, qx distribution and z inducing inputs.
%Given two arguments, we expect the following
% p                   parameter struct
%   hyp     1 x E     GP hyperparameter struct
%     l     F x 1     log length scale
%     pn    1 x 1     log process noise std dev
%   qx      1 x N     struct array for Gaussian q(x) distribution
%     m     E x T_n   mean
%     s  2ExE x T_n   representation of covariance
%   z      M x F x E  inducing inputs
% data      1 x N     data struct
%   y      T_n x D    cell array of observations
% u     T_n x U    cell array of control inputs
%

%Given three arguments and data points for every qx,
%we are predicting the latent states of a set of observation
%given an example seed trajectory, we expect

% qx         1 x N    struct array for Gaussian q(x) distribution
%   m       E x T_n  mean
%   s    2ExE x T_n  representation of covariance
% data      1 x N    data struct
%   y     T_n x D    cell array of seed observations
%   u     T_n x U    cell array of seed control inputs
% x
%   hyp     1 x E    GP hyperparameter struct
%     l     F x 1    log length scale
%     pn    1 x 1    log process noise std dev
%   z     M x F x E  inducing inputs

%Given three arguments, and no observations,
%we are predicting the next state given
%the previous state. We can do this analytically.

% qx         1 x N    struct array for Gaussian q(x) distribution
%   m       E x 1    mean of the last state
%   So      E x E    pairwise marginal covariance between last two states
%   Sd      E x E    marginal covariance of last state
% data      1 x N    data struct
%   u       1 x U    cell array of control input for prediction
% x
%   hyp     1 x E    GP hyperparameter struct
%     l     F x 1    log length scale
%     pn    1 x 1    log process noise std dev
%   z     M x F x E  inducing inputs

% Copyright (C) 2016 by Carl Edward Rasmussen, 20160530.
```

The Ψ function

In the implementation, a function `Psi` handles the part of the (negative) log marginal likelihood which depends on the quantities Ψ_1 and Ψ_2:

$$\psi \;=\; \tfrac{1}{2}\sum_{e=1}^{E} \log|\mathsf{K}_e| - \log|\mathsf{K}_e + \Psi_{2e}| - \mathrm{tr}\mathsf{K}_e^{-1}\Psi_{2e} - \Psi_{1e}^\top(\mathsf{K}_e + \Psi_{2e})^{-1}\Psi_{1e}. \qquad (20)$$

6 ⟨Psi 6⟩≡ (5b)

```
function [lml, dnlml] = Psi(hyp, qx, z, u, test);
% hyp       1 x E    GP hyperparameter struct
%   l       F x 1    log length scale
%   pn      1 x 1    log process noise std dev
% qx        1 x N    Gaussian q(x) distribution
%   m     E x T_n    mean
%   Sd  ExE x T_n    diagonal elements of covariance matrix
%   So  ExE x T_n-1  immediately off-diagonal elements of covariance matrix
% z       M x F x E  inducing inputs
% u       T_n x U    cell array of control inputs
% lml       1 x 1    contribution to the log marginal likelihood
% dnlml              derivatives
% test               flag indicating test mode

persistent K Psi1 Psi2;                          % keep these around if necessary
[M, F, E] = size(z);                                                 % get sizes
```

```
if nargin < 5 %Use to return stored Psi values
  lml = Psi1; dnlml = Psi2;
else
  ⟨expectations 7⟩
  if nargout > 0
    ⟨expectation derivatives 8a⟩
  end
end
```

The Ψ_1 and Ψ_2 expectations

The expectations from eq. (6) and derivatives wrt hyperparameters, the parameters of the $q(x)$ distribution and the pseudo-inputs z are calculated by the `Psi` function. To compute these expectations, the pairwise joint

$$q(x_{t-1:t}) \;=\; \mathcal{N}\Big(\begin{bmatrix}\mu_{t-1}\\ \mu_t\end{bmatrix}, \begin{bmatrix}\Sigma_{t-1,t-1} & \Sigma_{t-1,t}\\ \Sigma_{t,t-1} & \Sigma_{t,t}\end{bmatrix}\Big),$$

is multiplied with the covariance function, which can be written as an un-normalized joint Gaussian

$$k_e(x_{t-1}, z_{ie}) \;=\; \exp\Big(-\tfrac{1}{2}\begin{bmatrix}x_{t-1}-z_{ie}\\ x_t\end{bmatrix}^\top \begin{bmatrix}\Lambda_e^{-1} & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}x_{t-1}-z_{ie}\\ x_t\end{bmatrix}\Big),$$

yielding

$$\begin{aligned}\int x_t k_e(x_{t-1}, z_{ie}) q(x_{t-1:t}) dx_{t-1} dx_t \;&=\; \big(\mu_t + \Sigma_{t,t-1}[\Lambda_e + \Sigma_{t-1,t-1}]^{-1}(z_{ie} - \mu_{t-1})\big)\\ &\times |I + \Lambda_e^{-1}\Sigma_{t-1,t-1}|^{-1/2} \exp\big(-\tfrac{1}{2}(\mu_{t-1} - z_{ie})[\Lambda_e + \Sigma_{t-1,t-1}]^{-1}(\mu_{t-1} - z_{ie})\big).\end{aligned} \tag{21}$$

For Ψ_2 we have from eq. (6)

$$\begin{aligned}\int k_e(z_{ie}, x_{t-1}) k_e(x_{t-1}, z_{je}) q(x_{t-1}) dx_{t-1} \;&=\; \exp(-(z_{ie} - z_{je})\Lambda_e^{-1}(z_{ie} - z_{je})/4)\\ &\times |I + 2\Lambda_e^{-1}\Sigma_{t-1,t-1}|^{-1/2} \exp(-(\tfrac{z_{ie}+z_{je}}{2} - \mu_{t-1})[\Lambda_e/2 + \Sigma_{t-1,t-1}]^{-1}(\tfrac{z_{ie}+z_{je}}{2} - \mu_{t-1})/2).\end{aligned} \tag{22}$$

Both Ψ_1 and Ψ_2 are computed for each GP $e = 1, \dots, E$, each inducing input $z_{ie}, i = 1, \dots, M$, and added over (N time series and) $T_n - 1$ time points:

7 ⟨expectations 7⟩≡ (6)

```
if ~test %If we test, we use the stored values of Psi1, Psi2 from the training set.
  K = zeros(M,M,E); Psi1 = zeros(M,E); Psi2 = zeros(M,M,E);
end
lml = 0;
Sd = zeros(F,F);
for e = 1:E                                                       % for each GP
  K(:,:,e) = exp(-maha(z(:,:,e),[],diag(exp(-2*hyp(e).l)))/2) + 1e-6*eye(M);
  iL = diag(exp(-hyp(e).l)); L2 = diag(exp(2*hyp(e).l));
  b1 = zeros(M,1); b2 = zeros(M,M);
  for n = 1:length(qx)                                   % for each time series
    for t = 2:size(qx(n).m, 2)                             % for each time step
      Sd(1:E,1:E) = qx(n).Sd(:,:,t-1);           % covariance in top left corner
      r1 = prod(diag(chol(eye(F)+iL*Sd*iL)));                         % sqrt det
      r2 = prod(diag(chol(eye(F)+2*iL*Sd*iL)));                       % sqrt det
      s = bsxfun(@minus, z(:,:,e), [qx(n).m(:,t-1)' u{n}(t-1,:)]);
      a = s/(L2+Sd);
      b1 = b1 + (qx(n).m(e,t) + a(:,1:E)*qx(n).So(:,e,t-1)) ...
                                           .*exp(-sum(a.*s,2)/2)/r1;
      b2 = b2 + exp(-maha(s,-s,inv(L2+2*Sd))/4) / r2;
    end
  end
  if test
    w = (K(:,:,e)+Psi2(:,:,e))\Psi1(:,e);
    W = -K(:,:,e)\Psi2(:,:,e)/(Psi2(:,:,e)+K(:,:,e)) + w*w';
    lml = lml + exp(-2*hyp(e).pn)*(b1'*w ...
                        - sum(sum(b2.*exp(-maha(z(:,:,e),[],inv(L2))/4).*W))/2);
  else
    Psi1(:,e) = b1 * exp(-2*hyp(e).pn);
    Psi2(:,:,e) = b2 * exp(-2*hyp(e).pn) .* exp(-maha(z(:,:,e),[],inv(L2))/4);
    lml = lml - sum(log(diag(chol(K(:,:,e)+Psi2(:,:,e))))) + ...
           sum(log(diag(chol(K(:,:,e))))) + trace(K(:,:,e)\Psi2(:,:,e))/2 + ...
                                   Psi1(:,e)'/(K(:,:,e)+Psi2(:,:,e))*Psi1(:,e)/2;
  end
end
```

Note that in the implementation the state distribution $q(x)$ is concatenated with the (deterministic) control inputs u.

Psi derivatives

We need to compute the derivatives of Ψ wrt the parameters of the $q(x)$ distribution, wrt the u inducing inputs and wrt the hyperparameters. First, from eq. (20) we note

$$\begin{aligned}
\frac{\partial\psi}{\partial\Psi_{1e}} &= -(K_e+\Psi_{2e})^{-1}\Psi_{1e} \;=\; -w_e,\\
\frac{\partial\psi}{\partial\Psi_{2e}} &= -\tfrac{1}{2}K_e^{-1}\Psi_{2e}(K_e+\Psi_{2e})^{-1}+\tfrac{1}{2}w_ew_e^\top \;=\; -\tfrac{1}{2}R_e(K_e+\Psi_{2e})^{-1}+\tfrac{1}{2}w_ew_e^\top \;=\; W_e,\\
\frac{\partial\psi}{\partial K_e} &= -\tfrac{1}{2}R_e(K_e+\Psi_{2e})^{-1}R_e^\top+\tfrac{1}{2}w_ew_e^\top \;=\; V_e,
\end{aligned}$$

where we have defined $w_e = (K_e+\Psi_{2e})^{-1}\Psi_{1e}$ and $R_e = K_e^{-1}\Psi_{2e}$. These can be used together with the derivatives of Ψ_1e, Ψ_{2e} and K_e and the chain rule to get the desired derivatives.

8a ⟨expectation derivatives 8a⟩≡ (6)

```
dnlml.z = zeros(M,F,E);
for n = 1:size(qx,2), dnlml.qx(n).m = 0*qx(n).m; dnlml.qx(n).So = 0*qx(n).So;
dnlml.qx(n).Sd = -0*qx(n).Sd; end;
for e = 1:E
  w = (K(:,:,e)+Psi2(:,:,e))\Psi1(:,e);
  R = K(:,:,e)\Psi2(:,:,e);
  W = -R/(Psi2(:,:,e)+K(:,:,e)) + w*w';
  ⟨hyp derivatives 8b⟩
  ⟨Psi derivatives 8c⟩
end
```

8b ⟨hyp derivatives 8b⟩≡ (8a)

```
dnlml.hyp(e).pn = 2*sum(w.*Psi1(:,e)) - sum(sum(W.*Psi2(:,:,e)));
```

8c ⟨Psi derivatives 8c⟩≡ (8a)

```
iL = diag(exp(-hyp(e).l)); L2 = diag(exp(2*hyp(e).l));
W1 = W .* exp(-maha(z(:,:,e),[],inv(L2))/4);
D = zeros(M,F); H = zeros(F,1);
for n = 1:length(qx)
  T = size(qx(n).m,2);
  A = zeros(E,T); B = zeros(E,E,T); C = zeros(E,E,T-1);
  for t = 2:T
    Sd(1:E,1:E) = qx(n).Sd(:,:,t-1);          % covariance in top left corner
    r2 = prod(diag(chol(eye(F)+2*iL*Sd*iL)));                       % sqrt det
    s = bsxfun(@minus, z(:,:,e), [qx(n).m(:,t-1)' u{n}(t-1,:)]);
    a = s/(L2+Sd);
    a2 = s/(2*L2+4*Sd);
    SiS = (L2(1:E,1:E)+Sd(1:E,1:E))\qx(n).So(:,e,t-1);
    r = exp(-sum(a.*s,2)/2) / prod(diag(chol(eye(F)+iL*Sd*iL)));
    g = (qx(n).m(e,t) + a(:,1:E)*qx(n).So(:,e,t-1)).*w.*r;
    W2 = W1.*exp(-maha(s,-s,inv(L2+2*Sd))/4);
    X = bsxfun(@plus,permute(a2,[1 3 2]),permute(a2,[3 1 2]));
    A(:,t-1) = A(:,t-1) + SiS*(w'*r) - a(:,1:E)'*g + ...
                          squeeze(sum(sum(bsxfun(@times,W2,X(:,:,1:E)),2),1))/r2;
    A(e,t) = -w'*r;
    B(:,:,t-1) = squeeze(sum(sum(bsxfun(@times, ...
       bsxfun(@times,W2,X(:,:,1:E)),permute(X(:,:,1:E),[1 2 4 3])),2),1))/r2;
    B(:,:,t-1) = B(:,:,t-1) + SiS*((w.*r)'*a(:,1:E)) ...
                                  + inv(L2(1:E,1:E)+Sd(1:E,1:E))*sum(g)/2 ...
                                  - a(:,1:E)'*bsxfun(@times,g,a(:,1:E))/2 ...
                          - inv(L2(1:E,1:E)+2*Sd(1:E,1:E))*sum(sum(W2))/r2/2;
    C(:,e,t-1) = -bsxfun(@times,a(:,1:E),r)'*w;
    if ~test
      D(:,1:E) = D(:,1:E) - bsxfun(@times,w,r)*SiS';
      D = D + bsxfun(@times,g,a) - W2*a2/r2 - bsxfun(@times,sum(W2,2),a2)/r2;
      H = H + diag(Sd/(L2+2*Sd))*sum(sum(W2))/r2 ...
         + exp(2*hyp(e).l).*squeeze(sum(sum(bsxfun(@times,W2,X.^2),2),1))/r2;
      H(1:E) = H(1:E) ...
            + 2*exp(2*hyp(e).l(1:E)).*diag(SiS*bsxfun(@times,w,r)'*a(:,1:E));
      H = H - diag(Sd/(L2+Sd))*sum(g);
      H = H - exp(2*hyp(e).l).*diag(a'*bsxfun(@times,g,a));
    end
  end
  dnlml.qx(n).m = dnlml.qx(n).m + A * exp(-2*hyp(e).pn);
  dnlml.qx(n).Sd = dnlml.qx(n).Sd + B * exp(-2*hyp(e).pn);
  dnlml.qx(n).So = dnlml.qx(n).So + C * exp(-2*hyp(e).pn);
end
if ~test
G = W.*Psi2(:,:,e);
a = z(:,:,e)*diag(exp(-2*hyp(e).l)/2);
dnlml.z(:,:,e) = D*exp(-2*hyp(e).pn) + G*a - bsxfun(@times,sum(G,2),a);
B = bsxfun(@minus,permute(a,[1 3 2]),permute(a,[3 1 2]));
dnlml.hyp(e).l = H * exp(-2*hyp(e).pn)  ...
         + exp(2*hyp(e).l).*squeeze(sum(sum(bsxfun(@times,G,B.^2),1),2));
G = (R/(K(:,:,e)+Psi2(:,:,e))*R' + w*w').*K(:,:,e);
a = z(:,:,e)*diag(exp(-2*hyp(e).l));
B = bsxfun(@minus,permute(z(:,:,e),[1 3 2]),permute(z(:,:,e),[3 1 2]));
dnlml.hyp(e).l = dnlml.hyp(e).l ...
      + exp(-2*hyp(e).l).*squeeze(sum(sum(bsxfun(@times,B.^2,G),1),2))/2;
dnlml.z(:,:,e) = dnlml.z(:,:,e) + G*a - bsxfun(@times,sum(G,2),a);
end
```

Test set calculation

The distinguishing factor between training and test set calculations is whether the inducing target distribution is updated (training set) or kept fixed (test set). For the test set the contribution from the transition model to the log probability is

$$\begin{aligned}&\sum_{e=1}^{E}\tfrac{1}{2}\mathrm{tr}K_e^{-1}\Psi_{2e}(K_e+\Psi_{2e})^{-1}\Psi_{2e}^{*}-\tfrac{1}{2}\mathrm{tr}(K_e+\Psi_{2e})^{-1}\Psi_{1e}\Psi_{1e}^{\top}(K_e+\Psi_{2e})^{-1}\Psi_{2e}^{*}+\Psi_{1e}^{\top}(K_e+\Psi_{2e})^{-1}\Psi_{1e}^{*}\\&+\tfrac{1}{2}\sum_{t=2}^{T}\log|\Sigma_{t-1:t,t-1:t}|-\tfrac{1}{2}\sum_{t=2}^{T-1}\log|\Sigma_t|-\tfrac{1}{2}\mathrm{tr}Q^{-1}\sum_{t=2}^{T}(I+\mu_t^{\top}\mu_t+\Sigma_{t,t})-\tfrac{T-1}{2}\log|Q|-\tfrac{(T-1)E}{2}\log(2\pi).\end{aligned}\qquad(23)$$

The testing consists of two steps. From a supplied initial set of observations, we find the most likely latent states, maximising a q distribution while keeping the hyperparameters and inducing inputs constant, using eq. (23), along with the likelihood term.

After the most likely latent state has been found, we predict forward by adding a further point to the timeseries, and maximising the likelihood. This can be solved analytically..

Representation of the $q(x)$ distribution

The $q(x)$ distribution is parameterised through its mean `qx.m` and the marginal and pairwise covariances. Conceptually, we wish to parameterize the E by E covariance matrices (for the marginal distibutions) which we call `qx.Sd` (for diagonal) and the E by E covariances between conseqtive time points (for the pairwise marginals) which we call `qx.So` (for off-diagonal). However it is inconvenient to parametrise these matrices directly, as it would be difficult to ensure positive definiteness of the marginal and pairwise marginal covariance matrices. Instead, we use 2E by E representation `qx.s` such that

$$S_{d,t} = s_t^\top s_t, \quad \text{and} \quad S_{o,t-1} = s_{t-1}^\top s_t. \tag{24}$$

Using this representation, we can use call the optimizer with the unconstrained representation, which is the converted to the more convenient diagonal and off-diagonal representation at the beginning and the derivatives are reverted back at the end.

10a ⟨convert 10a⟩≡ (5b)
```
function [Sd, So] = convert(s)
[t, E, T]= size(s); Sd = zeros(E,E,T); So = zeros(E,E,T-1);
for t = 1:T, Sd(:,:,t) = s(:,:,t)'*s(:,:,t); end                 % diagonal terms
for t = 2:T, So(:,:,t-1) = s(:,:,t-1)'*s(:,:,t); end               % off-diagonal
```
The derivatives are

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial s_t} &= \frac{\partial\mathcal{L}}{\partial S_{d,t}}\frac{\partial S_{d,t}}{\partial s_t} + \frac{\partial\mathcal{L}}{\partial S_{o,t-1}}\frac{\partial S_{o,t-1}}{\partial s_t} + \frac{\partial\mathcal{L}}{\partial S_{o,t}}\frac{\partial S_{o,t}}{\partial s_t} = \frac{\partial}{\partial s_t}\mathrm{tr}\Big(\frac{\partial\mathcal{L}}{\partial S_{d,t}}s_t^\top s_t\Big) \\ &+ s_{t-1}\frac{\partial\mathcal{L}}{\partial S_{o,t-1}} + s_t\Big[\frac{\partial\mathcal{L}}{\partial S_{o,t}}\Big]^\top = s_t\Big(\frac{\partial\mathcal{L}}{\partial S_{d,t}} + \Big[\frac{\partial\mathcal{L}}{\partial S_{d,t}}\Big]^\top\Big) + s_{t-1}\frac{\partial\mathcal{L}}{\partial S_{o,t-1}} + s_t\Big[\frac{\partial\mathcal{L}}{\partial S_{o,t}}\Big]^\top.\end{aligned} \tag{25}$$

10b ⟨revert 10b⟩≡ (5b)
```
function r = revert(s, dSd, dSo)
for t = 1:size(s,3), r(:,:,t) = s(:,:,t)*(dSd(:,:,t)+dSd(:,:,t)'); end
for t = 2:size(s,3)
  r(:,:,t-1) = r(:,:,t-1) + s(:,:,t)*dSo(:,:,t-1)';
  r(:,:,t) = r(:,:,t) + s(:,:,t-1)*dSo(:,:,t-1);
end
```

10c ⟨assign outs 10c⟩≡ (5b)
```
out2.qx = rmfield(out2.qx,{'Sd','So'});        % change to qx.s representation
[out2.qx.s] = deal([]);                                  % create the "s" field
if test
  for n = 1:N
    out2.s = revert(p.s, dnlml.qx(n).Sd, dnlml.qx(n).So);
  end
  out2 = rmfield(out2, 'hyp');
  out2 = rmfield(out2, 'z');
  out2.m = out2.qx.m;
  out2 = rmfield(out2, 'qx');
  else
    for n = 1:N
      out2.qx(n).s = revert(p.qx(n).s, dnlml.qx(n).Sd, dnlml.qx(n).So);
    end
end
```

10d ⟨maha 10d⟩≡ (5b)
```
% Squared Mahalanobis distance (a-b)*Q*(a-b)'; vectors are row-vectors
% a, b  d x n  matrices containing n length d row vectors
% Q     d x d  weight matrix
% K     n x n  squared distances
function K = maha(a, b, Q)
if isempty(b), b = a; end
aQ = a*Q; K = bsxfun(@plus,sum(aQ.*a,2),sum(b*Q.*b,2)')-2*aQ*b';
```

Analytic Prediction

Predicting the next step is found by taking an existing latent representation of a timeseries, and maximising the likelihood of the timeseries found by increasing its length by one. Given a timeseries, if increase the length of the timeseries by one, we can analytically compute the parameters μ_t, $\Sigma_{t-1,t}$, $\Sigma_{t,t}$ which maximise the likelihood. From equation ??, $\frac{\partial\mathcal{L}}{\partial\mu_t} = \sum_{e=1}^{E}\Psi_{1e}^\top\left(K_e+\Psi_{2e}^*\right)^{-1}\frac{\partial\Psi_{1e}^*}{\partial\mu_t} - \mathrm{tr}(Q^{-1})\mu_t$, and using eq. 21,

$$\mu^*_{t,e} = \Psi_{1e}^\top (K_e + \Psi_{2e})^{-1/2} |I + \Lambda_e^{-1}\Sigma_{t-1,t-1}|^{-1/2} \exp\left(-\frac{1}{2}(\mu_{t-1} - z_{ie})[\Lambda_e + \Sigma_{t-1,t-1}]^{-1}(\mu_{t-1} - z_{ie})\right), \quad (26)$$

which corresponds to the mean prediction of the GP, conditioned on the inducing inputs. For the marginal $\Sigma_{t,t}$, we find that $\frac{\partial \mathcal{L}}{\partial \Sigma_{t,t}} = -\frac{1}{2}\mathrm{tr}Q^{-1} + \frac{1}{2}\frac{\partial}{\partial \Sigma_{t,t}}|\log \Sigma_{t-1:t,t-1:t}|$. Optimising this gives us

$$\Sigma^*_{t,t} = Q + \Sigma^*_{t-1,t}\Sigma^{-1}_{t-1,t-1}\Sigma^*_{t,t-1} \quad (27)$$

For the pair-wise marginal, $\frac{\partial \mathcal{L}}{\partial \Sigma_{t-1,t}} = \sum_{e=1}^E \Psi_{1e}^\top (K_e + \Psi_{2e})^{-1} \frac{\partial \Psi^*_{1e}}{\partial \Sigma_{t-1,t}} + \frac{1}{2}\frac{\partial}{\partial \Sigma_{t-1,t}}|\log \Sigma_{t-1:t,t-1:t}|$. This gives us

$$0 = \sum_{e=1}^E \Psi_{1e}^\top (K_e + \Psi_{2e})^{-1} \frac{\partial \Psi^*_{1e}}{\partial \Sigma_{t-1,t}} - \Sigma^{-1}_{t-1,t-1}\Sigma^*_{t-1,t}\left(\Sigma^*_{t,t} - \Sigma^*_{t-1,t}\Sigma_{t-1,t-1}\Sigma^*_{t-1,t}\right)^{-1} \quad (28)$$

.

Substituting, we find that

$$\begin{aligned}\Sigma^*_{t-1,t} =& \Sigma_{t-1,t-1}\sum_{e=1}^E \Psi_{1e}^\top (K_e + \Psi_{2e})^{-1} (\Lambda_e + \Sigma_{t-1,t-1})^{-1}(z - \mu_{t-1}) \\ &\times |I + \Lambda_e^{-1}\Sigma_{t-1,t-1}|^{-1/2} \exp\left(-\frac{1}{2}(\mu_{t-1} - z_{ie})[\Lambda_e + \Sigma_{t-1,t-1}]^{-1}(\mu_{t-1} - z_{ie})\right),\end{aligned} \quad (29)$$

which we can then substitute in to equation 27 to find $\Sigma^*_{t,t}$.

We can motivate our choice of $\Sigma^*_{t,t}$ by considering the prediction of the value of x_t conditional on the value of x_{t-1}. We have a conditional distribution $q(x_t|x_{t-1}) = \mathcal{N}(\mu', \Sigma')$, with $\Sigma' = \Sigma^*_{t,t} - \Sigma^{\top *}_{t-1,t}\Sigma^{-1}_{t-1,t-1}\Sigma^*_{t-1,t-1} = Q$. We can clearly see how the derived $\Sigma^*_{t,t}$ is exactly the variance for which the extra uncertainty added to the prediction of the next timestep is Q.

The intuition for equation 29 is that we are directly calculating the covariance $\mathbb{E}((z_i - \mu_{t-1})(z_j - \mu_t)) = \mathbb{E}(z_i(z_j - \mu_{t-1}))$, from the inducing points z. We rescale the individual contributions by $(\Lambda_e + \Sigma_{t-1,t-1})^{-1}$, and then scale up by a factor of $\Sigma_{t-1,t-1}$ after collecting the contributions from all z.

11 ⟨prediction 11⟩≡ (5b)

```

function qx_opt = predict_forwards(hyp, qx, z, u)
%Given a timeseries corresponding to latent states of an observed
%timeseries, analytically calculate the optimal next step params
%Fetch the Psi1,2 values by calling Psi with 4 arguments
[Psi1, Psi2] = Psi(hyp, qx, z, u);
E = size(qx, 2); [M, F, E] = size(z);
for e = 1:E
  K(:,:,e) = exp(-maha(z(:,:,e),[],diag(exp(-2*hyp(e).l)))/2) + ...
      1e-6*eye(M);
  w = (K(:,:,e)+Psi2(:,:,e))\Psi1(:,e);
  iL = diag(exp(-hyp(e).l)); L2 = diag(exp(2*hyp(e).l));
  b3 = zeros(E,1);
  b4 = zeros(1);
  for n = 1:size(qx,2)
    t = size(qx.m, 2); Sd = zeros(F,F);
    Sd(1:E,1:E) = qx(n).Sd(:,:,t);          % covariance in top left corner
    r1 = prod(diag(chol(eye(F)+iL*Sd*iL)));                      % sqrt det
    %  s = bsxfun(@minus, z(:,:,e), [qx(n).m(:,t)'
    %                          u{n}(t,:)]);
    s = bsxfun(@minus, z(:,:,e), [qx(n).m(:,t)' u{n}(t,:)]);
    a = s/(L2+Sd);
    b3(e,1) = w'*(exp(-sum(a.*s,2)/2)/r1); b4 = zeros(M,E);
    for f=1:E, b4(:,f) = a(:,f).*exp(-sum(a.*s,2)/2)/r1;end
    b5(e, :) = w'*b4;
  end
```

```
end
qx_opt(n).m = b3;
qx_opt(n).So = qx.Sd*b5;
qx_opt(n).Sd = exp(2*hyp(e).pn) + qx_opt(n).So * inv(qx(n).Sd(:, :, end)) ...
    * qx_opt(n).So';
```