# Message passing algorithms for time series

Due on February 09, 2015

*Literature survey*

**Adrian Millea**

# Contents

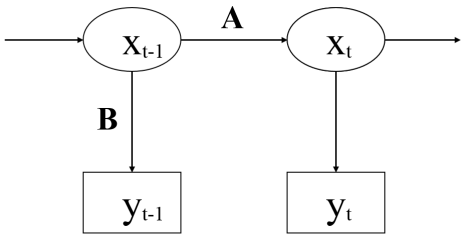


Figure 1: Graphical model for the SSM

# 1 Recursive Bayesian estimation

## 1.1 Introduction

In this literature review we intend to describe a set of algorithms for dealing with time series data, i.e. continuous, sequential data coming from an unknown dynamical process. We will investigate and describe mainly message passing algorithms in probabilistic graphical models with the occasional description of the tools needed. The problem setting is as follows: we are given a set of time-dependent observations ($\mathbf{y}_t$) and we want to uncover the true process that is generating the time-series. We will make use of what is called a *latent space* of variables ($\mathbf{x}_t$), that we assume it exists and it generated the observations. We can see a depiction of the graphical model in Figure 1. Observed variables are denoted with squares and latent variables are denoted by ovals. Note that the latent space might be higher dimensional or lower dimensional than the observation space, depending on the problem. In this context we are interested mainly in three different processes: filtering, prediction and smoothing.

- **Filtering** is an estimation operation performed upon a time-series in which previous information ($< t$) is used in some way with the purpose of producing some quantity of interest at time $t$.
- **Prediction** uses the same principle, i.e. using existent data until time $t$, but this time with the purpose of predicting a quantity of interest at time $t + \tau$. We will consider the simple case of $\tau = 1$. However, the general principles usually apply also to bigger $\tau$.
- **Smoothing** refers to the same estimation of a quantity of interest at time $t$, but this time we can use data from subsequent time-steps ($> t$).

This paper is written in the context of the Bayesian paradigm [1], however we have to note that this is not the only induction principle for performing statistical inference. Other choices could be: the frequentist approach [2], *minimax* [3] (worst case analysis), SRM (structural risk minimization) [4], Akaike Information Criterion-based inference (AIC)[5] which includes MDL (minimum description length principle) [6], Fiducial inference [7] and Structural inference [8]. The Bayesian principle is not optimal, and makes sense if the quantitative prior is correct [9].

## 1.2 State Space Models

The main tool for modeling time series in a filtering context is the assumption that the data is generated by a dynamical process which leads to the state space formulation [10]. A dynamical process or dynamical system as it is often called, is a mathematical description of a trajectory through an (often) high-dimensional manifold. The manifold is often called a state space model

(SSM) and can be described by the following equations (the following description is based on [**?**]):

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{v}_t$$
$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t + \mathbf{w}_t$$

where $\mathbf{x}_t$ is the unknown high-dimensional vector we want to uncover (this is often considered to be the true state of the system), at time step $t$, the $\mathbf{v}$ and $\mathbf{w}$ vectors are noise vectors (process noise and observation noise respectively) and $\mathbf{y}_t$ is the observation at time step $t$. $\mathbf{A}$ is the transition matrix (keep in mind that in some cases instead of A we can have a highly non-linear function) and describes how the system transitions from one state to the next, while the $\mathbf{C}$ matrix is called the observation (or emission) matrix (this can also be a non-linear function). We will first talk about the linear-Gaussian SSM, as this is the most simple model one can find and most importantly, it supports exact inference, as we will see in a moment. By linear we mean that $\mathbf{A}$ and $\mathbf{C}$ are linear functions, while for Gaussian we mean that the noise of the system is assumed to be Gaussian ($\mathbf{v} \sim \mathcal{N}(0, \mathbf{Q}_t)$ and $\mathbf{w} \sim \mathcal{N}(0, \mathbf{R}_t)$). We have to note here that in both equations we can sometimes find an additional term ($\mathbf{u}_t$ multiplied in each one with a linear operator) which is often called the control signal. If all the variables of the system do not change with time ($\mathbf{A}, \mathbf{C}, \mathbf{Q}, \mathbf{R}$) then the process is called stationary. One of the main goals when using SSMs is to uncover the posterior distribution of the hidden states ($p(\mathbf{x}_t|\mathbf{y}_{0:t})$). Then we can usually easily get the posterior predictive distribution $p(\mathbf{y}_{t+1}|\mathbf{y}_{0:t})$. We mentioned that the linear-Gaussian SSM supports exact inference, this is because as the initial state is Gaussian according to $p(\mathbf{x}_1) = \mathcal{N}(\mu_{1|0}, \mathbf{\Sigma}_{1|0})$ then all subsequent states will also be Gaussian according to $p(\mathbf{x}_t|\mathbf{y}_{1:t}) = \mathcal{N}(\mu_{t|t}, \mathbf{\Sigma}_{t|t})$ where we have denoted $\mu_{t|t} = \mathcal{E}[\mathbf{x}_t|\mathbf{y}_{1:t}]$ and analogous for $\Sigma$. The first $\mu$ and $\mathbf{\Sigma}$ are the parameters of the prior of $\mathbf{x}$, before seeing any data. The specified conditionals can be efficiently calculated using the well known Kalman filter (KF), which we will describe shortly. The KF can be derived using the Bayesian recursion equations [11, 12, 13], even though Kalman himself does not agree with this approach.

## 1.3 Bayesian recursion

For the derivation of the Bayesian filtering technique we have to have the assumption that the states follow a first-order Markov process (i.e. $p(\mathbf{x}_t|\mathbf{x}_{0:t-1}) = p(\mathbf{x}_t|\mathbf{x}_{t-1})$). We denote by $\mathbf{y}_T$ the set of all observations $\mathbf{y}_{0:t} = \mathbf{y}_0, ...\mathbf{y}_t$ for convenience. The posterior is then given by (for the exact derivation see [Chen]):

$$p(\mathbf{x}_t|\mathbf{y}_T) = \frac{p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{T-1})}{p(\mathbf{y}_t|\mathbf{y}_{T-1})}$$

This is mostly refered to as the posterior density and as we can see it is described by the terms:

- **Prior**: $p(\mathbf{x}_t|\mathbf{y}_{T-1})$ which is critical, as we said above and defines the knowledge of the model used:

  $$p(\mathbf{x}_t|\mathbf{y}_{T-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{y}_{T-1})d\mathbf{x}_{t-1}$$

  where $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ defines the transition of the states.

- **Likelihood**: $p(\mathbf{y}_t|\mathbf{x}_t)$ defines the measurement noise model of the system.

- **Evidence**: $p(\mathbf{y}_t|\mathbf{y}_{T-1})$, which involves the computation of an integral:

$$p(\mathbf{y}_t|\mathbf{y}_{T-1}) = \int p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{T-1})d\mathbf{x}_t$$

These three terms are at the core of the Bayesian paradigm. We outline now the two main steps in the recursive Bayesian estimation algorithm:

- This is called the *time update* or *prediction step* and it uses the system model to predict forward, trying to produce $p(\mathbf{x}_t|\mathbf{y}_{T-1})$ from $p(\mathbf{x}_{t-1}|\mathbf{y}_{T-1})$, i.e.:

$$p(\mathbf{x}_t|\mathbf{y}_{T-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{y}_{T-1})d\mathbf{x}_{t-1}$$

- The second step is called the *measurement update* step which involves using Bayes' theorem for incorporating the new data into the prediction and is trying to produce $p(\mathbf{x}_t|\mathbf{y}_T)$ from $p(\mathbf{x}_t|\mathbf{y}_{T-1})$:

$$p(\mathbf{x}_t|\mathbf{y}_T) = \frac{p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{T-1})}{p(\mathbf{y}_t|\mathbf{y}_{T-1})}$$

## 1.4 Optimal filtering

When we talk about optimal sequential filtering we need to define a measure of optimality [14]. Measures often used are:

- *Minimum mean-squared error (MMSE).* This is defined as:

$$\mathbb{E}[\|\mathbf{x}_n - \widehat{\mathbf{x}}_n\|^2 |\mathbf{y}_{0:n}] = \int \|\mathbf{x}_n - \widehat{\mathbf{x}}_n\|^2 p(\mathbf{x}_n|\mathbf{y}_{0:n})d\mathbf{x}_n$$

  The point is to minimize this measure by finding the appropriate conditional mean: $\widehat{\mathbf{x}}_n = \mathbb{E}[\mathbf{x}_n|\mathbf{y}_{0:n}] = \int \mathbf{x}_n p(\mathbf{x}_n|\mathbf{y}_{0:n})d\mathbf{x}_n$.

- *Maximum a posterior* (MAP) tries to find the mode of the posterior $p(\mathbf{x}_n|\mathbf{y}_{0:n})$.

- *Maximum likelihood* tries to maximize the likelihood function with respect to the parameters of the model.

- *Minimax* tries to find the median of the posterior $p(\mathbf{x}_n|\mathbf{y}_{0:n})$

- *Minimum conditional inaccuracy* (a generalization of Kerridge's inaccuracy [15]):

$$\mathbb{E}_{p(\mathbf{x},\mathbf{y})}[-\log \widehat{p}(\mathbf{x}|\mathbf{y})] = \int p(\mathbf{x},\mathbf{y}) \log \frac{1}{\widehat{p}(\mathbf{x}|\mathbf{y})} d\mathbf{x}d\mathbf{y}$$

- *Minimum conditional KL divergence*[16]:

$$KL = \int p(\mathbf{x},\mathbf{y}) \log \frac{p(\mathbf{x},\mathbf{y})}{\widehat{p}(\mathbf{x}|\mathbf{y})p(\mathbf{x})} d\mathbf{x}d\mathbf{y}$$

- *Minimum free energy.* This is a lower bound of maximum log-likelihood, and minimizes:

$$\mathcal{F}(q;p)\mathbb{E}_q(\mathbf{x})[-\log p(\mathbf{x}|\mathbf{y})] \qquad = \mathbb{E}_q(\mathbf{x})[\log \frac{q(\mathbf{x})}{p(\mathbf{x}|\mathbf{y})}] - \mathbb{E}_q(\mathbf{x})[\log q(\mathbf{x})]$$

  where $q(\mathbf{x})$ is usually an approximating distribution of the posterior. This is called a variational approximation and is usually employed in offline estimation.

For Bayesian filtering the Bayesian risk of MMSE is used. For the difference between Bayes risk and the frequentist risk see [17]. In general one cannot find the truly optimal solution as it would require infinite computing time and memory, however for some special cases, like for example linear-Gaussian case, this computation can be performed exactly in finite time. This is what the Kalman filter does.

### 1.4.1 Kalman Filter

Kalman filter is a well known and highly regarded filtering technique for on-line estimation. The Kalman filter has been derived for first time as a special case of the recursive Bayesian filter by [11]. In the linear-Gaussian case it is an unbiased minimum variance estimator, but if the Gaussian noise condition is violated then the KF is still optimal in the mean square sense, but then it is biased and also not the minimum variance estimate. We proceed to showing the derivation of the Kalman filter (based on [18]).

### 1.4.2 Kalman filter derivation

The Kalman filter model has a simple graphical representation, as in Figure 1. The posterior at the time $t$ is given by:

$$p(\mathbf{x}_t|\mathbf{y}_t) = \mathcal{N}(\mathbf{x}_t|\mu_t, \mathbf{\Sigma}_t)$$

Because we the model is linear and Gaussian we can derive the two steps (prediction and update) of the Kalman filter in closed form. In short, the prediction step predicts the posterior distribution of the latent given the previous observations:

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{y}_{t-1}) &= \int \mathcal{N}(\mathbf{x}_t|\mathbf{A}_t\mathbf{x}_{t-1}, \mathbf{Q}_t)\mathcal{N}(\mathbf{x}_{t-1}|\mu_{t-1}, \mathbf{\Sigma}_{t-1})d\mathbf{x}_{t-1} \\ &= \mathcal{N}(\mathbf{x}_t|\mu_{t|t-1}, \Sigma_{t|t-1}) \\ \text{with} \quad & \mu_{t|t-1} \triangleq \mathbf{A}_t\mu_{t-1} \\ \text{and} \quad & \mathbf{\Sigma}_{t|t-1} \triangleq \mathbf{A}_t\mathbf{\Sigma}_{t-1}\mathbf{A}_t^T + \mathbf{Q}_t \end{aligned}$$

The measurement update then includes the current measurement and the prediction in the calculation, using Bayes' Theorem.

$$p(\mathbf{x}_t|\mathbf{y}_t, \mathbf{y}_{1:t-1}) \propto p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$$

which is then given by:

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) = \mathcal{N}(\mathbf{x}_t|\mu_t, \mathbf{\Sigma}_t)$$

where $\mu_t$ and $\mathbf{\Sigma}_t$ are given by the following equations:
First, using Bayes' Theorem for Gaussians distributions (Appendix), we get the following for the posteriors:

$$\mathbf{\Sigma}_t^{-1} = \mathbf{\Sigma}_{t|t-1}^{-1} + \mathbf{C}_t^T\mathbf{R}_t^{-1}\mathbf{C}_t$$

Then, using the matrix inversion lemma (Appendix), we can rewrite it as:

$$\begin{aligned} \mathbf{\Sigma}_t &= \mathbf{\Sigma}_{t|t-1} - \mathbf{\Sigma}_{t|t-1}\mathbf{C}_t^T(\mathbf{R}_t + \mathbf{C}_t\mathbf{\Sigma}_{t|t-1}\mathbf{C}_t^T)^{-1}\mathbf{C}_t\mathbf{\Sigma}_{t|t-1} \\ &= (\mathbf{I} - \mathbf{K}_t\mathbf{C}_t)\mathbf{\Sigma}_{t|t-1} \end{aligned}$$

Again using Bayes' Theorem for Gaussians we get:

$$\mu_t = \mathbf{\Sigma}_t\mathbf{C}_t\mathbf{R}_t^{-1}\mathbf{y}_t + \mathbf{\Sigma}_t\mathbf{\Sigma}_{t|t-1}^{-1}\mu_{t|t-1}$$

Applying another matrix inversion lemma (Appendix) to the first term of the previous equation, we then get:

$$\begin{aligned}\boldsymbol{\Sigma}_t\mathbf{C}_t\mathbf{R}_t^{-1}\mathbf{y}_t &= (\boldsymbol{\Sigma}_{t|t-1}^{-1}\mathbf{C}_t^T\mathbf{R}_t^{-1}\mathbf{C}_t)^{-1}\mathbf{C}_t\mathbf{R}_t^{-1}\mathbf{y}_t \\ &= \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T(\mathbf{R}_t + \mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T)^{-1}\mathbf{y}_t = \mathbf{K}_t\mathbf{y}_t\end{aligned}$$

Applying again the matrix inversion lemma to the second term of the equation we get:

$$\begin{aligned}\boldsymbol{\Sigma}_t\boldsymbol{\Sigma}_{t|t-1}^{-1}\mu_{t|t-1} &= (\boldsymbol{\Sigma}_{t|t-1}^{-1} + \mathbf{C}_t^T\mathbf{R}_t^{-1}\mathbf{C}_t)^{-1}\boldsymbol{\Sigma}_{t|t-1}^{-1}\mu_{t|t-1} \\ &= [\boldsymbol{\Sigma}_{t|t-1} - \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}^T(\mathbf{R}_t + \mathbf{C}_t^T\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T)\mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}]\boldsymbol{\Sigma}_{t|t-1}^{-1}\mu_{t|t-1} \\ &= (\boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t\mathbf{C}_t^T\boldsymbol{\Sigma}_{t|t-1})\mathbf{C}_t^T\boldsymbol{\Sigma}_{t|t-1}^{-1}\mu_{t|t-1} \\ &= \mu_{t|t-1} - \mathbf{K}_t\mathbf{C}_t^T\mu_{t|t-1}\end{aligned}$$

Putting everything together gives:

$$\mu_t = \mu_{t|t-1} + \mathbf{K}_t(\mathbf{y}_t - \mathbf{C}_t\mu_{t|t-1})$$

## 1.5 Nonlinear filtering - an ill posed problem

Assuming instead of $A$ and $C$ we have nonlinear functions $f$ and $g$, but both function are known, the goal is to estimate the unknown latent variables $\mathbf{x}_t$ from the observations $\mathbf{y}_{0:t}$. This can be seen as an inverse mapping problem, find the input $\mathbf{x}_t$, at every time step, given the output $\mathbf{y}_t$ mapped through a composite function ($f$ and $g$). This is inverse in the sense that the usual problems, is given the inputs, one is interested in the output, here the goal is the other way around. This mapping can be non-unique, in the sense that there might be multiple inputs that generate the same outputs. The function from inputs to outputs might not be injective and so might not have an inverse. A well-posed problem must satisfy three conditions: *existence*, *uniqueness* and *stability*. If a problem does not satisfy one of these conditions, it is said to be an ill-posed problem. We already mentioned the non-uniqueness, but also stability is not satisfied, given the fact that we are dealing with density estimation in high-dimensional spaces, which is known to be ill-posed [9]. One often used method for solving inverse problems is through the Bayesian principle, by taking into account prior knowledge and observation evidence. The interpretation of probability as a conditional measure of uncertainty is critical for this approach.

## 1.6 Extensions of the Kalman Filter

### 1.6.1 Extended Kalman Filter

As we said earlier the Kalman filter is optimum in the MMSE sense, if the system dynamics is linear and if the noise can be assumed to be Gaussian. However, these conditions do not always hold for all problems. We consider next the non-linear case, with Gaussian noise:

$$\begin{aligned}\mathbf{x}_t &= f_t(\mathbf{x}_{t-1}) + \mathbf{v}_t \\ \mathbf{y}_t &= g_t(\mathbf{x}_t) + \mathbf{w}_t\end{aligned}$$

where $f$ and $g$ are two nonlinear functions and the rest of notations are the same as in the previous section. As we said earlier, computing the posterior $p(\mathbf{x}_t|\mathbf{y}_T$ implies knowing the entire conditional pdf, which is exact and known just for the linear-Gaussian case and is in fact also Gaussian, as we stated before. In this nonlinear case, the posterior is not known and evaluation of the functions $f$ and $g$ on the covariance cannot be done directly. The solution which Extended Kalman Filter (EKF) [19] employs is to linearize these functions about the previous estimate (in

the case of $f$) and about the predicted state (for $g$) by computing a first order Taylor expansion (higher order terms can be included also, see [20]), involving the computation of the Jacobian of the two functions. For this, the functions need to be differentiable. The rest of the process is the same, with the computation of the Jacobians at each step (where the notation is consistent with the previous section on the derivation of the regular KF):

$$A_t = \nabla f_t | \mu_t$$
$$C_t = \nabla g_t | \mu_{t|t-1}$$

The rest of the algorithm is identical with the regular KF, with the observation that now the matrices $A_t$ and $C_t$ depend on the estimates and thus, on measurements, so the Kalman gain and the covariances cannot be computed offline as is the case in the regular KF. EKF is not optimal, is based on approximations with Taylor series and so the covariances are not the true covariances of the state estimates but approximations. The EKF may also diverge, if the successive linearizations introduce a big enough error into the approximation, or sometimes may produce inaccurate results. There is a solution to some extent to the insufficiencies of the EKF we will present it in the next section, it is called the Unscented Kalman Filter (UKF) [21].

### 1.6.2 Unscented Kalman Filter

In the UKF the state distribution is still represented by a Gaussian random variable, but is specified using a minimal set of chosen sample points. The sample points capture the true mean and covariance of the Gaussian random variable and when propagated through the nonlinear system can capture the posterior as accurate as a 3rd order Taylor expansion, and this with the same computational burden as the EKF. The UKF is based on what is known as an unscented transformation (UT) [22]. The UT is a method for computing the statistics of a random variable which goes through a nonlinear transformation $\mathbf{y} = g(\mathbf{x})$. Let $\mathbf{x}$ be a random variable with a mean $\mu$ and covariance $\Sigma_{\mathbf{x}}$. To get the statistics of $\mathbf{y}$, a matrix $\chi$ is formed of $2L + 1$ *sigma* vectors $\chi_i$ with associated weights $W_i$, given by:

$$\begin{aligned}
\chi_0 &= \mu \\
\chi_i &= \mu + (\sqrt{(L+\lambda)\Sigma_{\mathbf{x}}})_i, i = 1, .., L \\
\chi_i &= \mu + (\sqrt{(L+\lambda)\Sigma_{\mathbf{x}}})_{i-L}, i = L+1, .., 2L \\
W_0^{(m)} &= \lambda/(L+\lambda) \\
W_0^{(c)} &= \lambda/(L+\lambda) + (1 - \alpha^2 + \beta) \\
W_i^{(m)} &= W_i^{(m)} = \frac{1}{/} 2(L+\lambda), i = 1, .., 2L
\end{aligned}$$

where $\lambda = \alpha^2(L + \kappa) - L$ is a scaling parameter, $\alpha$ is determining the spread of sigma points around $\mu$ (in general set to a small value $\approx 1e-3$), $\kappa$ is another scaling parameter (in general set to 0) and $\beta$ is used to incorporate prior knowledge about the distribution of interest ($\beta = 2$ for Gaussian is optimal). These so defined sigma points are then propagaed through the nonlinear function:

$$\dagger_i = g(\chi_i), i = 0, .., 2L$$

Then the mean and covariance of the posterior sigma points are given by:

$$\mu_{\mathbf{y}} \approx \sum_{i=0}^{2L} W_i^{(m)} \dagger_i \Sigma_{\mathbf{y}} \approx \sum_{i=0}^{2L} W_i^{(c)} (\dagger_i - \mu_{\mathbf{y}})(\dagger_i - \mu_{\mathbf{y}})^T$$

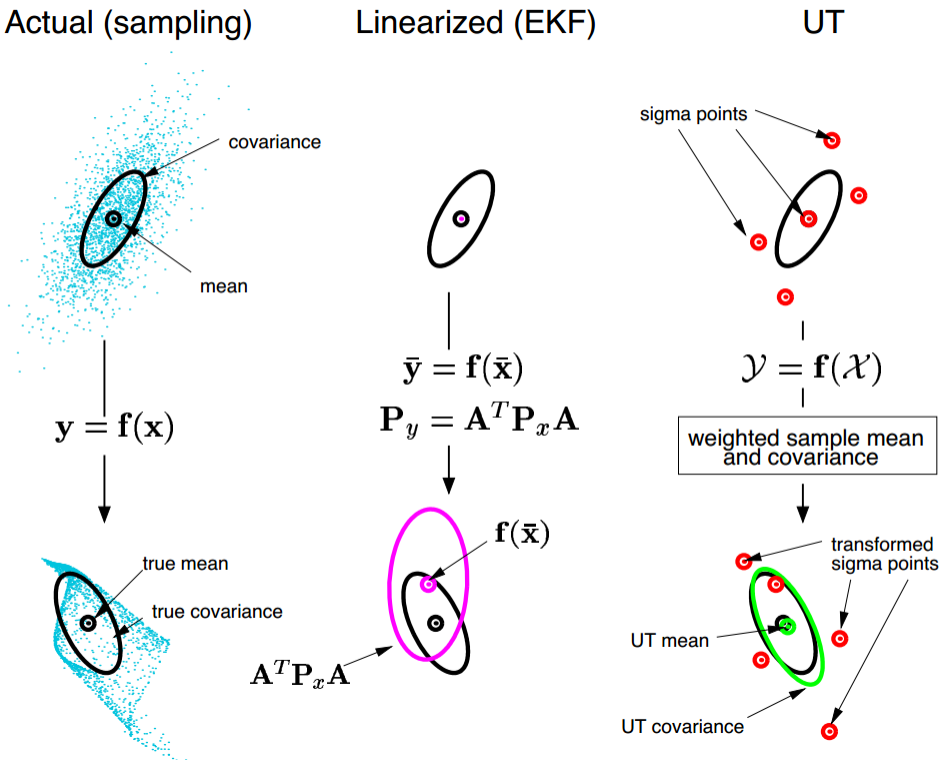


Figure 2: Example of the Unscented transformation: a) actual b) EKF c) UT

We show a depiction of the UT in Figure 2. The UKF is then a simple extension of the above equations to the recursive estimation process, with the addition that first the new state vector is a concatenation of the original state vector and the noise vectors and then the sigma point calculation (the equations in the UT) are used to compute the sigma matrix. For more details of the algorithm see [21].

#### 1.6.3 Variational Kalman Filter

# 2 Factor graphs

## 2.1 Introduction

Graphical models are probabilistic tools for describing statistical dependencies among (usually) large sets of random variables by employing graph teoretic rules to model the interaction between the random variables. They are useful because complex computations can be employed based on the graphical representations to perform inference and additional learning. A graph can be defined by a set of vertices $V$ and a set of edges $E$. For directed graphical models the edges have a start node and an end node, denoted by the direction of the arrow of the edge, whilst for non-directed graphs the edge does not have a direction associated with it. There is a process, called moralization, through which a directed graph can be transformed to an undirected graph, by the addition of certain edges, not to violate the certain dependencies present in the directed graph. The directed graphs are more useful for modeling causality, while non-directed graphs are used to define soft constrains between the random variables. There is an alternative representation of both directed and undirected graphs, called factor graphs

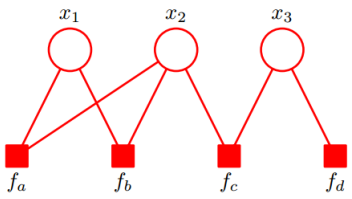


Figure 3: Example of a factor graph.

[23, 24], which emphasizes the factorization properties of such a joint distribution represented by a graphical model. The representation is possible because both the directed and undirected graphs allow the overall function of the variables of the graph to be expressed as a product of functions (called factors) of subsets of the variables. This factorization is explicit in factor graphs with the introduction of additional nodes, besides the variable nodes, with different properties. Thus, the joint distribution of a set of variables can be written as:

$$p(\mathbf{X}) = \prod_s f_s(\mathbf{x}_s)$$

where we have denoted $\mathbf{x}_s$ as a subset of variables $\mathbf{X}$. Directed graphs can be specified also by the previous factorization, where the individual factors $f_s$ represent local conditional distributions, so they are special cases of factor graphs. While the undirected graphs can be described by the same equation, with the difference that the factors are the potential functions over the maximal cliques of the graph, while the partition function (in undirected graphs the partition function is $1/Z$ and represents a normalization coefficient) can be considered a special factor defined over the empty set. The circles in a factor graph represent variables, while the squares represent the factors of the joint distribution. Each variable node is connected to a factor node, no two variable nodes can be connected, as well as no two factor nodes can be connected. A graph (directed or undirected) can have multiple different representing factor graphs, however, keep in mind that not always the minimal representation is the best one, as more factors are able to convey more information about the underlying factorization. The choice of factors is usually a matter of the problem modeled. We show an example of a factor graph in Figure 3, which has the following factorization of the joint distribution:

$$p(\mathbf{X}) = f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

Note that in an undirected graph the product of two factors which depend upon the same variables would be grouped into the same clique potential. We will see later that the tree structure of a graph (i.e. the graph does not contain any cycles, there is only one path connecting any two nodes) is very important for performing inference. Converting a directed or undirected tree to a factor graph yields also a tree. However, there is an additional desirable difference of factor graphs: converting a directed polytree into a undirected graph results in the graph having cycles (because of moralization), while converting into a factor graph guarantees that the conversion will result still in a tree, moreover by choosing special sets of variables on which the factors depend on, cycles can even be eliminated. See Figure 4. Exact inference can be applied to graphs that have a tree structure (note that we didn't say tree-like, for locally tree-like graphs we have some bounds for inference but approximations are still needed [25, 26]). The problem is formulated as follows: we will evaluate marginals locally over nodes or sets of nodes to give in the end the *sum-product* algorithm [24], a powerful and efficient algorithm for performing inference in graphs. If we want to find the maximal posterior distribution over all variables we can replace the sum with the *max* operator to yield the *max-product*, or if we apply

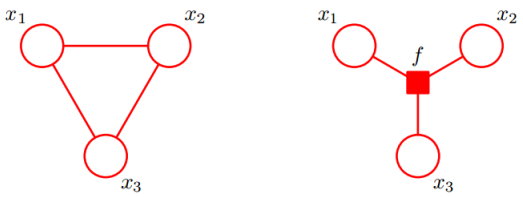


Figure 4: Example of conversion into a factor graph and eliminating the cycle present in the undirected graph.

the logarithm for the products we get the *max-sum* algorithm. In the end, the principle is the same, even though there are minor differences between the different flavors of the algorithms. We assume that the variables are discrete, so that marginalizations reduce to performing sums (we can also use continuous variables as long as we are in the linear-Gaussian case). The well-known *belief propagation* algorithm, used for exact inference on directed graphs without loops is a special case of the sum-product algorithm.

## 2.2 The sum-product algorithm

As we said earlier, we are dealing first with a factor graph that has a tree structure. The goal of the problem is to find marginals (using exact inference in this case), and also to find marginals efficiently, i.e. reuse as many computations as possible for the different marginals. We assume that all the variables in the graph are hidden, for now (we will see later how to use also observed variables). By definition of the marginalization operation, we have:

$$p(x) = \sum_{\mathbf{x} \setminus x} p(\mathbf{x})$$

where $\mathbf{x} \setminus x$ denotes the set of all the variables in $\mathbf{x}$ without the variable $x$. Thus we can rewrite the orignal factorization as:

$$p(\mathbf{x}) = \prod_{s \in ne(x)} F_s(x, X_s)$$

where $ne(x)$ is the set of all neighbors of $x$, and $X_s$ is the set of all variables in the subtree of $f_s$ (i.e. all variables connected to $x$ through $f_s$), while $F_s$ is the product of all factors associated with $f_s$. Rewriting again the original factorization, while reordering the sum and product gives:

$$\begin{aligned} p(x) &= \prod_{s \in ne(x)} \left[ \sum_{X_s} F_s(x, X_s) \right] \\ &= \prod_{s \in ne(x)} \mu_{f_s \to x}(x) \end{aligned}$$

where we have defined $\mu_{f_s \to x}(x) \triangleq \sum_{X_s} F_s(x, X_s)$. This is considered to be a message to the variable node $x$ from factor node $f_s$. Thus, the marginal of interest $p(x)$ is the product of all the messages arriving at node $x$. For each factor $F_s$ we can further use the factorization described by the subgraph:

$$F_s(x, X_s) = f_s(x, x_1, ...x_M) G_1(x_1, X_{s1}) ... G_M(x_M, X_{sM})$$

where the variables associated with factor $f_s$ (we can denote this $\mathbf{x}_s$) are, besides $x$: $x_1, x_2, ...x_M$. Replacing the factorization of $F_s$ in the definition for the message $f_s \to x$ we have:

$$\begin{aligned}\mu_{f_s \to x}(x) &= \sum_{x_1} ... \sum_{x_M} f_s(x, x_1, ..., x_M) \prod_{m \in ne(f_s) \setminus x} \left[ \sum_{X_{sm}} G_m(x_m, X_{sm}) \right] \\ &= \sum_{x_1} ... \sum_{x_M} f_s(x, x_1, ..., x_M) \prod_{m \in ne(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)\end{aligned}$$

where we have defined again the message from variable node $x_m$ to factor node $f_s$ as:

$$\mu_{x_m \to f_s}(x_m) \triangleq \sum_{X_{sm}} G_m(x_m, X_{sm})$$

and $ne(f_s)$ is as before the set of all neighbors of $f_s$; in this case the neighbors are variable nodes. We see that we have two types of messages: from variables nodes to factor nodes ($\mu_{x \to f}$) and from factor nodes to variable nodes ($\mu_{f \to x}$). To note here is that, when considering an edge and the message passed through it, we see that in both cases the message is a function of the variable node which is at one end of the edge. In short, when considering the message from a factor node to a variable node, we see that first the factor node has to receive messages from all the other variable nodes which it is connected to, then take the product of all these messages, multiply with the actual factor of the node and then marginalize over all the variables connected to the node. Analogous, for messages from variable nodes to factor nodes, we have the factorization:

$$G_m(x_m, X_{sm}) = \prod_{l \in ne(x_m) \setminus f_s} F_l(x_m, X_{ml})$$

where again we consider all the neighbors of variable $x_m$ excluding factor $f_s$. We see that each factor $F_l(x_m, X_{ml})$ is a subtree of the original graph. Again substituting this last factorization in the same way as before, in the definition of the message from $x_m \to f_s$, and reordering the sum and product, we get:

$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in ne(x_m) \setminus f_s} \left[ \sum_{X_{ml}} F_l(x_m, X_{ml}) \right]$$

Again, we note that a variable node has to receive all messages from all the other neighboring factor nodes before sending a message to a specific factor node. This message consists of taking this product of all incoming messages. If a variable node has two neighboring factor nodes, then there is no product anymore, the variable node just passes messages unchanged. Now we come back at the original problem, computing the marginal for variable $x$. As we saw, the messages are computed recursively as a function of other messages, however we have to bootstrap somehow, thus we consider the $x$ node as being the root of the tree and we can start computing messages from the leaves. Of course we have two types of leaves, variable nodes and factor nodes. If a leaf is a variable node, the first message to its neighboring factor node ($x$ to $f$) is:

$$\mu_{x \to f}(x) = 1$$

and if the leaf is a factor node then the message to its neighboring variable node is:

$$\mu_{f \to x}(x) = f(x)$$

Choosing a specific root makes reasoning easier, but any node can be chosen as the root because the graph is a tree. We now give a simple argument for the fact that each node will receive enough messages to send itself messages further to the root. Consider that we start with the simplest version of a tree: the root node is directly connected to the leaf nodes. In this case, the computation is trivial, however when adding nodes, we have to respect the constraint of keeping the graph with a tree structure, so a single node added means we connect it through a single link, which then gives a new subtree which is a trivial tree as the one we began with. The tree structure is critical for the sum-product algorithm. Sometimes we need to find the marginals for all the variable nodes. This could be done by running the above algorithm for each node separately, however this would increase the computation significantly, making the overall algorithm quadratic in the size of the graph. A more efficient solution is to just propagate messages in both directions, first from the leaves to the root, and then the root would have received all messages from the neighbors, so it can itself send messages to neighbors; the same is valid for subsequent nodes, neighbors of the root and so on. All nodes would have received and sent messages to the neighbors, and so the marginals can be easily computed. This gives an algorithm which requires just two times the computations for a single marginal. As before, the root was selected arbitrarily and any node can be selected as root. If we want to find the marginals $p(\mathbf{x}_s)$ where each $x$ represents the set of variables connected to each of the factors. Then the marginal is given by:

$$p(\mathbf{x}_s) = f_s(\mathbf{x}_s) \prod_{i \in ne(f_s)} \mu_{x_i \to f_s}(x_i)$$

Suppose the factors depend on some parameters we wish to learn. We can do this using the EM algorithm, while using in the E step exactly these calculations of the marginals. Another important issue is normalization. If the factor graph was built from a directed graph, then the joint distribution is already normalized, however if the factor graph was built from an undirected graph we need to compute the normalization coefficient, also known as the partition function $Z$. Luckily this can be done efficiently. The algorithm can work with the unnormalized versions of the marginals at first, and then the partition function can be computed just by normalizing any of the local marginals. We said earlier that the algorithm can be extended to include observations. So the goal would be to compute posterior distributions conditioned on the observed variables. We first partition $\mathbf{x}$ into observed ($\mathbf{y}$) and hidden variables ($\mathbf{z}$) and we denote the observed values of $\mathbf{y}$ as $\widehat{\mathbf{y}}$. We then consider the function $I(y_i, \widehat{y_i})$ which is equal to 1 if $y = \widehat{y}$ and 0 otherwise. Then the joint distribution $p(\mathbf{z}, \mathbf{y} = \widehat{\mathbf{y}}) = p(\mathbf{x}) \prod_i I(y_i, \widehat{y_i})$ is the unnormalized version of $p(\mathbf{z}|\mathbf{y} = \widehat{\mathbf{y}})$. The normalization coefficient can be found efficiently using a local computation as we mentioned before. Then summations over $\mathbf{y}$ collapse into a single term.

## 2.3 The max-sum algorithm

One other problem in a factor graph is to find a joint setting of the individual variables for which the probability is the largest. This can be done through a modified version of the sum-product algorithm called max-sum, which employs dynamic programmming in the context of graphical models [27]. An easy approach would be to use the sum-product algorithm to find marginals $p(x_i)$ and then find the value $x_i$ that maximizes the marginal. However, this would give the individual maximal values for the marginals, and most probably this would not coincide with the overall maximal value for the joint distribution. Thus, the problem is to find $\mathbf{x}_{max}$ that maximizes the joint distribution, such that:

$$p(\mathbf{x}_{max}) = \max_{\mathbf{x}} p(\mathbf{x})$$

The *max* operator can be written as:

$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_1} ... \max_{x_M} p(\mathbf{x})$$

where M is the total number of variables in our model. We know that the *max* operator can be distributed with respect to multiplication (if $a \geq 0$, which is always true for graphical models):

$$\max(ab, bc) = a \max(b, c)$$

Thus, we can exchage the max and product operators to give the $max - product$ algorithm. The process of sending messages is the same as in the sum-product algorithm. We can add a logarithm to the product terms such that we avoid numerical errors due to very small probabilities. We can do this, as the logarithm is a monotonic function of it's argument, so we can reorder the *max* and *ln* operators like this:

$$\ln \left( \max_{\mathbf{x}} p(\mathbf{x}) \right) = \max_{\mathbf{x}} \ln p(\mathbf{x})$$

The distributive property of the two operators (*max* and *product* now transformed into *max* and *sum*) still holds. This gives rise to the max-sum algorithm. We show next the form of the messages from factor nodes to variable nodes and the vice-versa:

$$\mu_{f \to x}(x) = \max_{x_1, ..., x_M} \left[ \ln f(x, x_1, ..., x_M) + \sum_{m \in ne(f_s) \setminus x} \mu_{x_m \to f}(x_m) \right]$$

$$\mu_{x \to f}(x) = \sum_{l \in ne(x) \setminus f} \mu_{f_l \to x}(x)$$

The messages from the leafs follow the same rule as in the sum-product algorithm, meaning:

$$\mu_{x \to f}(x) = 0$$
$$\mu_{f \to x}(x) = \ln f(x)$$

and at the root, similar again to the sum-product algorithm, the maximum probability of interest is given by:

$$p_{max} = \max_{x} \left[ \sum_{s \in ne(x)} \mu_{f_s \to x}(x) \right]$$

So we have propagated all the messages from the leaves to the root. We now continue with the problem of finding the configuration of the individual variables for which the joint distribution is maximum. However, because there can be multiple configurations which generate the maximum joint distribution (because we are using the max instead of the sum), if we would just propagate the messages back to the leaves, we might stumble upon different maximizing configurations for the individual variables at each local computation. This means that in the end the overall joint might not be maximum. Thus, we need to remember for every previous variable, what was the value of the variable that generated the maximum, so we need to store:

$$\phi(x_n) =_{x_{n-1}} \left[ \ln f_{n-1,n}(x_{n-1}, x_n) + \mu_{x_{n-1} \to f_{n-1,n}}(x_n) \right]$$

So for each state of a given variable, there is a unique state of the previous one which maximizes its probability. When we reach the final node $x_N$ we can then simply use back-tracking to go

back to the maximizing state of the previous node $x_{N-1}$. This is equivalent to sending the following message:

$$x_{n-1}^{max} = \phi(x_n^{max})$$

So when a message is sent from a factor node to a variable node, a maximization over all neighbors of the factor node is performed. We now store the maximizing configurations of the neighbors of the factor node. After finding the maximum value for the original variable node we can then back-track such that we always use the correct maximizing values of the neighboring variables. This back-tracking algorithm gives an *exact* maximizing configuration of the variables, if the graph has a tree structure.

# 3 Variational Inference

## 3.1 Introduction

Variational methods refers to a set of optimization techniqes using what is know as calculus of variations. While standard calculus is concerned with computing derivatives of functions, calculus of variations is concerned with computing derivatives of functionals, that is, instead of having a variable as the input to the function and expressing infinitesimal changes of that, we have a function as the input and we are interested in how the value of the functional changes when having infinitesimal changes in the input function. In short, one defines an optimization problem, in which the quantity of interest is a functional. Even though, in itself, variational inference (VI) is not an approximation technique, by restricting the set of used functions, the problem is transformed into an approximation problem. Variational inference is considered to solve some of the problems of the existing approaches, like for example expectation propagation, which is limited to certain classes of models for which the desired expectations can be evaluated, also, it is not guaranteed to converge, and most importantly, it does not deal well with multi-modal distributions. Belief propagation is guaranteed to converge only for tree-structured graphs, limiting a lot its actual applicability. The main idea of the above mentioned algorithms (including VI) is to transform the computation into local computations at each node in the graphs and then passing messages between a small number of neighboring nodes. In the variational message passing framework [28] the messages are from the exponential family of distributions, with child nodes sending their natural parameter vector to parents, while parents send a vector of moments to child nodes. The overall idea is to get the optimal distribution of a node by summing over all the messages from its children together with a function of the messages from the parents, and this function depends on the conditional of that node. The framework presented here can deal with models which can be represented by directed acyclic graphs of discrete or continuous nodes. The marginals can come from any distribution, they need not be exponential. Thus, many machine learning algorithms can be defined as a version of the VMP presented here. These include HMM, PPCA, FA, KF, etc. The framework is fully Bayesian, meaning latent variables and parameters of the model are considered unobserved and marginalized out to make predictions.
We proceed to give a general description of the variational approach to Bayesian networks. After the problem has been set up, we will investigate further the message passing framework. We split the random variables in the model into visible ($\mathbf{V}$) and hidden, or latent variables, as they are usually known ($\mathbf{H}$), where the whole set of variables is defines as $\mathbf{X} = (\mathbf{V}, \mathbf{H})$. Being in a Bayesian framework the overall joint distribution has the form:

$$p(\mathbf{X}) = \prod_i p(X_i|pa_i)$$

where $pa_i$ is the set of all parents of node $i$ and $X_i$ is the variable or subset of variables associated with node $i$. The goal is as usual in a Bayesian network to find the true posterior $p(\mathbf{H}|\mathbf{V})$. We will try to approximate the true posterior with an auxiliary distribution (we will call this the variation distribution) $q(\mathbf{H})$. First step is to decompose the log marginal probability of the visible set of variables as follows:

$$\ln p(\mathbf{V}) = \mathcal{L}(q) + KL(q||p)$$

where we have defined:

$$\mathcal{L}(q) = -\int q(\mathbf{H}) \ln \frac{p(\mathbf{H}, \mathbf{V})}{q(\mathbf{H})}$$
$$KL(q||p) = \int q(\mathbf{H}) \ln \frac{p(\mathbf{H}|\mathbf{V})}{q(\mathbf{H})}$$

where $KL(q||p)$ is known as the Kullback-Leibler divergence. For discrete variables, sums can be used instead of integrals. We will continue assuming that our variables are discrete, so we will use the sum notation from here onwards. The $KL$ is always positive, which means that $\mathcal{L}$ can be considered a lower bound on the overall log marginal. So the goal is now to find a variational distribution $q(\mathbf{H})$ which maximizes this lower bound, in the same time minimizing the $KL$, which is 0 when $q(\mathbf{H})$ equals the true posterior $p(\mathbf{H}|\mathbf{V})$. The goal being to optimize $q$, we then need to choose it such that it is efficient to optimize, while still being flexible to approximate well enought the true posterior. This version of the $KL$ is called the 'exclusive' divergence, and one of its insufficiencies is that it can ignore moded of $p$. However, if we choose to minimize the other version of the $KL$ (note that the KL is not symmetric, i.e. $KL(q||p) \neq KL(p||q)$), the inclusive divergence, $KL(p||q)$ we can end up with posterior mass in regions where $p$ has vanishing density. The inclusive divergence is minimized in what is known as the expectation propagation algorithm [29] and we will present it in the next section. The optimization process should be tractable, thus we should choose a structure for $q$, such that is it much simpler than the true posterior. One usual approach [30, 31, 32] to this is to consider that $q$ factorizes as:

$$q(\mathbf{H}) = \prod_i q_i(\mathbf{H}_i)$$

where $\mathbf{H}_i$ are disjoint subsets of variables in $\mathbf{H}$. Using this factorization, the equation for the lower bound (also known as the evidence lower bound, or ELBO) can be rewritten as:

$$\mathcal{L}(q) = \sum_{\mathbf{H}} \prod_i q_i(\mathbf{H}_i) \ln p(\mathbf{H}, \mathbf{V}) - \sum_i \sum_{\mathbf{H}_i} q_i(\mathbf{H}_i) \ln q_i(\mathbf{H}_i)$$

The variational approach is also known as the mean-field theory in physics. It involves expectations of the factors in the following way (we rewrite the previous equation, with one factor j separated):

$$\begin{aligned}\mathcal{L}(q) &= \sum_{\mathbf{H}_j} q_j(\mathbf{H}_j) \langle \ln p(\mathbf{H}, \mathbf{V}) \rangle_{\sim q(\mathbf{H}_j)} + \mathbb{H}(q_j) + \sum_{i \neq j} \mathbb{H}(q_i) \\ &= -KL(q_j||q_j^*) + \text{terms not in} q_j\end{aligned}$$

where $\mathbb{H}$ is the well-known entropy and we have defined the new distribution $q_j^*$ as:

$$\ln(q_j^*(\mathbf{H}_j)) = \langle \ln p(\mathbf{H}, \mathbf{V}) \rangle_{\sim q(\mathbf{H}_j)} + const.$$

where $\langle\rangle_{\sim q(\mathbf{H}_j)}$ is the expectation with respect to all factors except the factor $q_j(\mathbf{H}_j)$. The $KL$ divergence is 0 when $q_j = q_j^*$, and so the lower bound is maximized if we set $q_j$ equal to $q_j^*$.

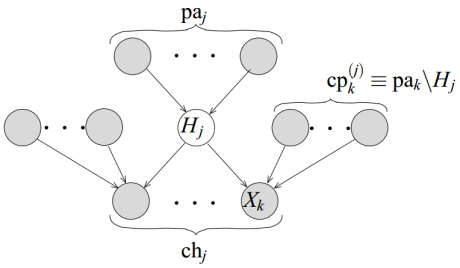


Figure 5: The update for one node $\mathbf{H}_j$depends on the set of parents, children and co-parents of the node.

Taking exponentials of both sides we then get:

$$q_j^*(\mathbf{H}_j) = \frac{1}{Z} \exp\langle \ln p(\mathbf{H}, \mathbf{V}) \rangle_{\sim q(\mathbf{H}_j)}$$

where $\frac{1}{Z}$ is the normalization coefficient such that $q_j^*$ is a valid probability distribution. We can see that the equations for all the factors are coupled, as the solution for each factor $q_j(\mathbf{H}_j)$ depends on all the other factors $i \neq j$ (actually, the solution depends on their expectations). The variational procedure initializes all factors and then cycles through all updating the distributions with the newest estimate.

## 3.2 Variational Message Passing

We show next the variational message passing algorithm which employs a graphical model and passes messages between nodes to optimize the factorized variational distribution. We show in Figure 5 the local dependency of node $\mathbf{H}_j$, where $ch_j$ is the set of children of node $j$, and $cp_k$ is the set of all parents of node $k$ but without the parent $\mathbf{H}_j$, as depicted in the figure. We rewrite the posterior $p(\mathbf{H}, \mathbf{V})$ as it is usually written in a Bayesian network:

$$\ln q_j^*(\mathbf{H}_j) = \langle \sum_i \ln p(X_i|pa_i) \rangle_{\sim q(\mathbf{H}_j)} + const.$$

We can split this into term that depend on $\mathbf{H}_j$ and terms that do not, and thus, under the expectation they will be constant. Terms depending on $\mathbf{H}_j$ are all the children of $\mathbf{H}_j$ and the conditional $p(\mathbf{H}_j|pa_j)$. Thus, we can rewrite:

$$\ln q_j^*(\mathbf{H}_j) = \langle \ln p(\mathbf{H}_j|pa_j) \rangle_{\sim q(\mathbf{H}_j)} + \sum_{k \in ch_j} \langle \ln p(X_k|pa_k) \rangle_{\sim q(\mathbf{H}_j)} + const. \quad (1)$$

Thus, the optimization of $q_j$ is expressed as a local computations at node $\mathbf{H}_j$, with terms from child nodes and one term from all the parent nodes (a sum). These terms can be considered 'messages' from the respective nodes in the graphical model, and thus the optimization can be decomposed into local computations involving neighbors (children and parents) of the node $\mathbf{H}_j$. The exact form of the messages will depend on the choice of the conditional distributions in the model. Simplifications of the variational updates have been shown to exist [30, 31] when the distribution of a node conditioned on the parents are exponential family distributions and are conjugate with respect to distributions over parent variables. This is called a conjugate-exponential model, and we will describe next the message passing procedure for these models.

### 3.3 Conjugate-exponential models

We show first the conditional distribution which is in the exponential family:

$$p(\mathbf{X}|\mathbf{Y}) = \exp\left[\phi(\mathbf{Y})^T \mathbf{u}(\mathbf{X}) + f(\mathbf{X}) + g()\right]$$

where $\phi(\mathbf{Y})$ is known as the *natural parameter* and $\mathbf{u}(\mathbf{X})$ is known as the *natural statistic*. For the distribution to integrate to unity of any setting of the parameters $\mathbf{Y}$, the term $g(\mathbf{Y})$ is added, which acts like a normalization function. Some advantages of exponential family distributions are that the expectations of their logarithm are tractable to compute as well as the fact that they are completely described by their natural parameters, so that in a conjugate model the posterior just changes the values of the parameter, and not the functional form of the distribution. Knowing the natural parameter vector $\phi(\mathbf{Y})$ enables us to know the expectation of the natural statistic vector [28]. This is given by:

$$\langle \mathbf{u}(\mathbf{X}) \rangle_{p(\mathbf{X}|\phi)} = -\frac{d\widehat{g}(\phi)}{d\phi} \tag{2}$$

where $\widehat{g}$ is a reparametrization of $g$ in terms of $\phi$. The factors of the variational distribution $q$ are in the exponential family and they have the same natural statistic as the corresponding factor of $p$, so the expectation of $\mathbf{u}$ under the $q$ distribution can also be found using the above equation.

### 3.4 Optimization of the variational distribution $q$

The distribution corresponding to a node $\mathbf{Y}$ is given by:

$$\ln p(Y|pa_Y) = \phi_Y(pa_Y)^T \mathbf{u}_Y(Y) + f_Y(Y) + g_Y(pa_Y)$$

The subscript $Y$ for the functions in the above equations is there to denote the fact that the functions are different for different members of the exponential family, so they might be different for each node. Let $X \in ch_Y$, so then the conditional given its parents is given by:

$$\ln p(X|Y, cp_Y) = \phi_X(Y, cp_Y)^T \mathbf{u}_X(X) + f_X(X) + g_X(Y, cp_Y) \tag{3}$$

where $cp_Y$ is like before, the set of co-parents of $Y$ with respect to $X$. $p(Y|pa_Y)$ can be considered a prior over $Y$ and $p(X|Y, cp_Y)$ can be considered a contribution to the likelihood of $Y$. As we said before, the fact that we are dealing with a conjugate model enables us to have the same functional form with respect to $Y$ for the two conditionals above. Thus, we write:

$$\ln p(X|Y, cp_Y) = \phi_{XY}(X, cp_Y)^T \mathbf{u}_Y(Y) + \lambda(X, cp_Y) \tag{4}$$

where we have defined two additional functions $\phi_X Y$ and $\lambda$. The function $\phi_{XY}$ can be calculated locally at $X$ because the conjugacy constraint imposes the form $\mathbf{u}_Y(Y)$ for any parent $Y$ of $X$ and so this function can be computed from the conditional $p(X|pa_X)$. We can see from 3 and 4 that $p(X|Y, \beta)$ must be linear in $\mathbf{u}_X(X)$ and $\mathbf{u}_Y(Y)$, and the same for all the co-parents of $Y$ ($cp_Y$). So because of the conjugacy constraints $p(X|pa_X)$ is a multi-linear function of $\mathbf{u}$ of $X$ and all parents of $X$ ($pa_X$). This is a general results, saying that if we constrain our model to be conjugate-exponential, then for any variable $X$ it follows that $\ln p(X|pa_X)$ is a multi-linear function of all $\mathbf{u}_A(A)$ and $\mathbf{u}_i(i)$, where $i \in pa_X$. Going back to equation 1 we can write the expectations in terms of $\mathbf{u}$ for each node in the Markov blanket of $Y$. Rewriting gives:

$$\ln q_Y^*(Y) = \left[ \langle \phi_Y(pa_Y) \rangle \sim q(Y) + \sum_{k \in ch_Y} \langle \phi_{XY}(X_k, cp_k) \rangle \sim q(Y) \right]^T \mathbf{u}_Y(Y) + f_Y(Y) + const. \tag{5}$$

So then $q_j^*$ is in the exponential family of distributions, having the same functional form as $p(Y|pa_Y)$ with a natural parameter given by:

$$\phi_Y^* = \langle \phi_Y(pa_Y) \rangle + \sum_{k \in ch_Y} \langle \phi_{XY}(X_k, cp_k) \rangle$$

where expectations are of course with respect to $q$. We can now reparameterize $\phi_{XY}$ to be functions of the expectations of the variables they depend on, meaning:

$$\widehat{\phi}_Y(\langle \mathbf{u}_i \rangle_{i \in pa_Y}) = \langle \phi_Y(pa_Y) \rangle$$
$$\widehat{\phi}_{XY}(\langle \mathbf{u}_k \rangle, \langle \mathbf{u}_j \rangle_{j \in cp_k}) = \langle \phi_{XY}(X_k, cp_k) \rangle$$

Now we have to compute the expectations of $\mathbf{u}$ under $q$. We see in equation 5 that any variable node $X$ has an associated factor $q_X$ with the same function form as $p(X|pa_X)$. So the expectation of $\mathbf{u}$ can be found using equation 2. If the node is observed then we replace the usual expectation with the calculation of $\mathbf{u}_X(X)$.

### 3.4.1 Definition of the Variational Message Passing algorithm

Let node $Y$ be the parent node of node $X$ is:

$$\mathbf{m}_{Y \to X} = \langle \mathbf{u}_\mathbf{Y} \rangle$$

Then the message from child $X$ to parent $Y$ is given by:

$$\mathbf{m}_{X \to Y} = \widehat{\phi}_{XY}(\langle \mathbf{u}_\mathbf{X} \rangle, \{\mathbf{m}_{i \to X}\}_{i \in cp_Y})$$

We can see here that for $X$ go be able to send a message to parent $Y$, it needs to have been received all the messages from the other parents besides $Y$. In the case a child node is observed, the expectation $\langle \mathbf{u}_A \rangle$ is replaced with $\mathbf{u}_A$. After $Y$ has received all the messages from all the nodes in its Markov blanket (children and parents), then the posterior $q_{Y*}$ can be updated from the computation of the natural parameter vector $\phi_{Y*}$. This is given by:

$$\phi_{Y*} = \widehat{\phi}_Y(\{\mathbf{m}_{i \to Y}\}\, i \in pa_Y) + \sum_{j \in ch_Y} \mathbf{m}_{j \to Y}$$

The quantity we are interested in, is the natural statistic vector $\langle \mathbf{u}_Y \rangle_{q_Y^*}$, which can be computed from the natural parameter vector $\phi_Y^*$. We show in 3.4.1 the overview for the Variational Message Passing algorithm (from [28]):

**Variational Message Passing algorithm**

1. Initialize each moment vector $\langle \mathbf{u}_j(X_j) \rangle$, which effectively initializes each $q_j$ distribution.
2. For each node $X_j$
   - Receive messages from all parents and child nodes. This in turn will necessitate the co-parents of $X_j$ to send messages to their children.
   - Compute natural parameter $\phi_j^*$
   - Compute moment vetor $\langle \mathbf{u}_j(X_j) \rangle$ given the new $\phi_j^*$
3. Compute the lower bound $\mathcal{L}$
4. If the increase in $\mathcal{L}$ is smaller than a threshold or a fixed number of iterations has been reached stop. Otherwise run from step 2.

# 4 Expectation propagation

## 4.1 Introduction

In the context of Bayesian inference, approximation methods for the posterior distribution of interest are critical for efficient computation. Expectation propagation (EP) is a deterministic algorithm that is more efficient than other approximation techniques and achieves higher accuracy in many problems. EP is an extension to Assumed Density Filtering (ADF), which is a sequential one-pass algorithm for approximating the posterior distribution of interest. EP has been used also in time-series filtering and smoothing problems. EP has also been described as a message passing algorithm. We first describe shortly the ADF algorithm and then proceed to defining the EP method.

## 4.2 Assumed Density Filtering

ADF is also known as online Bayesian learning, moment matching and weak marginalization. These are different names for the same concept, seen from different perspectives (statistics, control, artificial intelligence). Let $\mathcal{D}$ be the set of observed data we have and let $\mathbf{x}$ be the set of our hidden variables. The usual problem is to find $p(\mathbf{x}|\mathcal{D})$, useful for estimation of the true state of the system and $p(\mathcal{D})$, also known as the evidence of the model, which is useful for model selection. Assuming the observation density comes from a mixture of two Gaussians, in the following way:

$$p(\mathbf{y}|\mathbf{x}) = (1-w)\mathcal{N}(\mathbf{y};\mathbf{x},\mathbf{I}) + w\mathcal{N}(\mathbf{y},10\mathbf{I})$$
$$\mathcal{N}(\mathbf{y};\mathbf{m},\mathbf{V}) = \frac{exp(-1/2(\mathbf{y}-\mathbf{m})^T\mathbf{V}^{-1}(\mathbf{y}-\mathbf{m}))}{|2\pi\mathbf{V}|^{1/2}}$$

This is known as the clutter problem, and the components of the observation density are related to the latent variables (first) and clutter (second), while $w$ is the clutter ratio (known). Assuming a Gaussian prior distribution over the latent variables:

$$p(\mathbf{x}) \sim (N)(\mathbf{0}, 100\mathbf{Y}_d)$$

where $d$ is the dimensionality of the latent vector. To employ ADF we need to write the posterior distribution as:

$$p(\mathcal{D},\mathbf{x}) = \prod_i t_i(\mathbf{x})$$

where $t_0(\mathbf{x}) = p(\mathbf{x})$ and $t_i(\mathbf{x}) = p(\mathbf{y}_i|\mathbf{x})$. To approximate the posterior, we choose a spherical Gaussian distribution:

$$q(\mathbf{x}) \sim \mathcal{N}(\mathbf{m}_x, v_x\mathbf{I}_d)$$

The last step is to cycle through the terms $t_i$ and incorporate them into the approxmating posterior. To incorporate a term $t_i(\mathbf{x})$, let $\widehat{p}(\mathbf{x})$ be the exact posterior given by:

$$\widehat{p}(\mathbf{x}) = \frac{t_i(\mathbf{x})q^{\sim i}(\mathbf{x})}{\int_{\mathbf{x}} t_i(\mathbf{x})d\mathbf{x}}$$

and now minimize the KL-divergence $KL(\widehat{p}(\mathbf{x})||q(\mathbf{x}))$ constrained by $q(\mathbf{x})$ being in the approximating family. As we chose q to be a spherical Gaussian, the solution is given by matching moments as:

$$\mathbb{E}_q[\mathbf{x}] = \mathbb{E}_{\widehat{p}}[\mathbf{x}]$$
$$\mathbb{E}_q[\mathbf{x}^T\mathbf{x}] = \mathbb{E}_{\widehat{p}}[\mathbf{x}^T\mathbf{x}]$$

So when the approximating posterior is in the exponential family, ADF basically propagates expectations. At each step the normalizing factor $Z_i = \int_{\mathbf{x}} t_i(\mathbf{x}) q^{\sim i}(\mathbf{x}) d(\mathbf{x})$. The product of these $Z_i$ gives an estimate of the evidence, $p(\mathcal{D})$. Thus, we have:

$$Z_i = (1-w)\mathcal{N}(\mathbf{y}_i; \mathbf{m}_x^{\sim i}, (v_x^{\sim i}+1)\mathbf{I}) + w\mathcal{N}(\mathbf{y}_i; \mathbf{0}; 10\mathbf{I})$$

The ADF algorithm is then given by [29]:
**Assumed density filtering algorithm**

1. Initialize $\mathbf{m}_x = \mathbf{0}, v_x = 100$ (the prior). Initialize $s \leftarrow 1$ (scale factor).
2. For each data point $\mathbf{y}_i$, update ($\mathbf{m}_x$,$v_x$,$s$) as in:

$$\begin{aligned} s &= s^{\sim i} \times Z_i \\ r_i &= 1 - \frac{1}{Z_i} w\mathcal{N}(\mathbf{y}_i; \mathbf{0}, 10\mathbf{I}) \\ \mathbf{m}_x &= \mathbf{m}_x^{\sim i} + v_x^{\sim i} r_i \frac{\mathbf{y}_i - \mathbf{m}_x^{\sim i}}{v_x^{\sim i}+1} \\ v_x &= v_x^{\sim i} - r_i \frac{(v_x^{\sim i})^2}{v_x^{\sim i}+1} + r_i(1-r_i)\frac{(v_x^{\sim i})^2 \left\|\mathbf{y}_i - \mathbf{m}_x^{\sim i}\right\|^2}{d(v_x^{\sim i}+1)^2} \end{aligned}$$

In short, this can be seen as computing the probability $r$ for each data point, of not being clutter, then we make an update to the estimate $\mathbf{x}(\mathbf{m}_x)$ and then change the confidence of the estimate, $v_x$. We can see that the algorithm depends strongly of the order of data processing, as the probability of a data point being clutter depends on the current estimate $\mathbf{x}$ which in turn, depends on the data already processed. We now proceed to the expectation propagation algorithm [29].

## 4.3 Expectation propagation

We saw that in ADF we first treat each term $t_i$ exactly and then approximate the posterior that incorporates $t_i$. However, the problem could be solved by first approximating $t_i$ with $\widehat{t}_i$ and then solving for the exact posterior with these approximating $\widehat{t}_i$s. We can see these approximating $\widehat{t}_i$s as the ratio between the posterior that incorporated $t_i$ and old posterior, the one before the incorporation. This means:

$$\widehat{t}_i(\mathbf{x}) = Z_i \frac{q(\mathbf{x})}{q^{\sim i}(\mathbf{x})}$$

A desired property of this approximation is the fact that considering the approximate posterior to be in the exponential family, then all the approximating terms will also be in the exponential family. The algorithm thus computes a Gaussian approximation $\widehat{t}_i(\mathbf{x})$ and then combines these analytically to get a Gaussian posterior on $\mathbf{x}$. In this context, it can be seen that the order does not matter anymore, and the algorithm can be improved by just reiterating the same process multiple times, in any order. Thus, we can now state the Expectaion Propagation algorithm [29]:
**Expectation Propagation**

1. Initialize the approximating terms $\widehat{t}_i$
2. Compute the approximating posterior from the product of the approximating terms:

$$q(\mathbf{x}) = \frac{\prod_i \widehat{t}_i(\mathbf{x})}{\int \prod_i \widehat{t}_i(\mathbf{x}) d\mathbf{x}}$$

3. Until all $\widehat{t}_i$ converge:

   - Choose a $\widehat{t}_i$ to refine
   - Remove the $\widehat{t}_i$ from the posterior to get what we called earlier in text an 'old posterior' $q^{\sim i}(\mathbf{x})$ by division and normalization:

$$q^{\sim i}(\mathbf{x}) \propto \frac{q(\mathbf{x})}{\widehat{t}_i(\mathbf{x})}$$

   - Combine $q^{\sim i}(\mathbf{x})$ and $t_i(\mathbf{x})$, then minimize the KL divergence to get a posterior $q(\mathbf{x})$ with the normalizing coefficient $Z_i$
   - Update $\widehat{t}_i = Z_i \frac{q(\mathbf{x})}{q^{\sim i}(\mathbf{x})}$

4. Use the normalizaion as an approximation to $p(\mathcal{D})$:

$$p(\mathcal{D}) \approx \int \prod_i \widehat{t}_i(\mathbf{x}) d\mathbf{x}$$

The algorithm is not guaranteed to converge, even though it has at least one fixed point, sometimes even many. To better grasp how the algorithm works, we follow [29] and give the example of EP on the clutter problem.

## 4.4 EP on the Clutter problem

1. Each approximating term $\widehat{t}_i$ has the form:

$$\widehat{t}_i(\mathbf{x}) = s_i exp(-\frac{1}{2v_i}(\mathbf{x} - \mathbf{m_i})^T(\mathbf{x} - \mathbf{m}_i))$$

   Initialize then the prior such that: $v_0 = 100$, $\mathbf{m}_0 = 0$, $s_0 = (2\pi v_0)^{-d/2}$. Initialize $v_i = \infty$, $\mathbf{m}_i = 0$ and $s_i = 1$ such that $\widehat{t}_i(\mathbf{x}) = 1$

2. $\mathbf{m}_x = \mathbf{m}_0$, $v_x = v_0$

3. Until all ($\mathbf{m}_i$,$v_i$,$s_i$) converge (changes $< 10^{-4}$): loop with $i = 1$ to $n$

   - Remove $\widehat{t}_i$ from the posterior:

$$(v_x^{\sim i})^{-1} = v_x^{-1} - v_i^{-1}\mathbf{m}_x^{\sim i} = \mathbf{m} + v_x^{\sim i}v_i^{-1}(\mathbf{m}_x - \mathbf{m}_i)$$

   - Recompute ($\mathbf{m}_x$,$v_x$,$Z_i$) from ($\mathbf{m}_x^{\sim i}$,$v_x^{\sim i}$) like in ADF.
   - Update $\widehat{t}_i$:

$$\begin{aligned} v_i^{-1} &= v_x^{-1} - (v_x^{\sim i})^{-1} \\ \mathbf{m}_i &= \mathbf{m}_x^{\sim i} + (v_i + v_x^{\sim i})(v_x^{\sim i})^{-1}(\mathbf{m}_x - \mathbf{m}_x^{\sim i}) \\ s_i &= \frac{Z_i}{(2\pi v)^{d/2}\mathcal{N}(\mathbf{m}_i; \mathbf{m}_x^{\sim i}, (v_i + v_x^{\sim i})\mathbf{I})} \end{aligned}$$

4. Compute the normalizing constant:

$$B = \frac{\mathbf{m}_x^T\mathbf{m}_x}{v_x} - \sum_i \frac{\mathbf{m}_i^T\mathbf{m}_i}{v_i}$$

$$p(\mathcal{D}) \approx (2\pi v_x)^{d/2} exp(B/2) \prod_{i=0}^{n} s_i$$

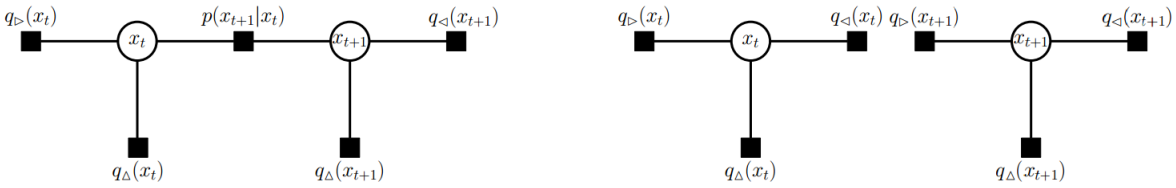


Figure 6: Factor graph (left) and fully factored graph (right) of a general dynamical system.

## 4.5 Expectation Propagation in Gaussian Process Dynamical Systems

In the context of dynamical systems, [33] have shown a new approximate message passing scheme for Bayesian state estimation in Gaussian Process dynamical systems on factor graphs (see Figure 6). The whole algorithm is quite involved but we will give its short description. We start by defining the notation:

**Gaussian EP for Dynamical Systems**

1. **Init**: Set all factors $q_i$ to $\mathcal{N}(\mathbf{0}, \infty\mathbf{I})$; Set $q(\mathbf{x}_1) = p(\mathbf{x}_1)$ and marginals $q(\mathbf{x}_{t\neq 1}) = \mathcal{N}(\mathbf{0}, 10^{10}\mathbf{I})$
2. **repeat**
3. **for** $t = 1$ to $T$ **do**
4. **for** all factors $q_i(\mathbf{x}_t)$, where $i = \triangleright, \triangle, \triangleleft$ **do**
5. Compute cavity distribution $q^{\setminus i}(\mathbf{x}_t) = q(\mathbf{x}_t)/q_i(\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_t|\mu^{\setminus i}, \mathbf{\Sigma}^{\setminus i})$ with
$$\mathbf{\Sigma}^{\setminus i} = (\mathbf{\Sigma}_t^{-1} - \mathbf{\Sigma}_i^{-1})^{-1}, \qquad \mu^{\setminus i} = \mathbf{\Sigma}^{\setminus i}((\mathbf{\Sigma}_t^{-1}\mu_t - \mathbf{\Sigma}_i^{-1}\mu_i)$$
6. Determine moments of $f_i(\mathbf{x}_t)q^{\setminus i}(\mathbf{x}_t)$, e.g. via the derivatives of:
$$\log Z_i(\mu^{\setminus i}, \mathbf{\Sigma}^{\setminus i}) = \log \int f_i(\mathbf{x}_t)q^{\setminus i}(\mathbf{x}_t)d\mathbf{x}_t$$
7. Update the posterior $q(\mathbf{x}_t) \propto \mathcal{N}(\mathbf{x}_t|\mu, \mathbf{\Sigma})$ and the approximate factor $q_i(\mathbf{x}_t)$:
$$\begin{aligned} \mu_t &= \mu^{\setminus i} + \mathbf{\Sigma}^{\setminus i}\nabla_m^T, & \mathbf{\Sigma}_t &= \mathbf{\Sigma}^{\setminus i} - \mathbf{\Sigma}^{\setminus i}(\nabla_m^T\nabla_m - 2\nabla_s)\mathbf{\Sigma}^{\setminus i} \\ \nabla_m &:= d\log Z_i/d\mu^{\setminus i}, & \nabla_s &:= d\log Z_i/d\mathbf{\Sigma}^{\setminus i} \\ q_i(\mathbf{x}_t) &= q(\mathbf{x}_t)/q^{\setminus i}(\mathbf{x}_t) \end{aligned}$$
8. **end for**
9. **end for**
10. **until** Convergence or maximum number of iterations exceeded.

For each node $\mathbf{x}_t$ in the fully factor graph in Figure 6, EP computes three messages $q_{\triangleright}(\mathbf{x}_t), q_{\triangleleft}(\mathbf{x}_t)$ and $q_{\triangle}(\mathbf{x}_t)$. Then the marginal $q(\mathbf{x}_t)$ and messages are updated. First, the *cavity distribution* $q^{\setminus i}(\mathbf{x}_t)$ is computed by removing $q_i(\mathbf{x}_t)$ from the marginal $q(\mathbf{x}_t)$. Then, in the *projection* step, the moments are computed $f_i(\mathbf{x}_t)q^{\setminus i}(\mathbf{x}_t)$, where $f_i$ is the true factor. The moments for the exponentia family can be computed using the derivative of the log-partition function $\log Z_i$ of $f_i(\mathbf{x}_t)q^{\setminus i}(\mathbf{x}_t)$ [29]. Then the moments of the marginal $q(\mathbf{x}_t)$ are set to the moments of

$f_i(\mathbf{x}_t)q^{\setminus i}(\mathbf{x}_t)$ and the message $q_i(\mathbf{x}_t)$ is updated. This procedure is applied for all states $\mathbf{x}_t$, until convergence. To update the posterior $q(\mathbf{x}_t)$ and the messages, EP needs to compute the log-partition function $\log Z_i$, but for nonlinear models this means computing integrals of the form:

$$p(\mathbf{a}) = \int p(\mathbf{a}|\mathbf{x}_t)p(\mathbf{x}_t)d\mathbf{x}_t = \int \mathcal{N}(\mathbf{a}|\mathbf{m}(\mathbf{x}_t), \mathbf{S}(\mathbf{x}_t))\mathcal{N}(\mathbf{x}_t|\mathbf{b}, \mathbf{B})d\mathbf{x}_t$$

where $\mathbf{a} = \mathbf{z}_t$ for the measurement message and $\mathbf{a} = \mathbf{x}_{t+1}$ for the forward and backward messages; $\mathbf{m}(\mathbf{x}_t)$ and $\mathbf{S}(\mathbf{x}_t)$ are the predictive GP means and covariances for the GPDS. Then, the authors in [33] approximate $p(\mathbf{a})$ by a Gaussian distribution, taking into account the implicit linearization in the computation of the derivatives $\nabla_{\mathbf{m}}$ and $\nabla_{\mathbf{s}}$.

# 5 Switching Space State Models (SSSM)

## 5.1 Introduction

Linear dynamical systems (LDS) are powerful tools for describing the underlying dynamics of a process generating a time-series. It was succesfully used in describing human motion [34], moving targets [35], the dance of honey bees [36] and financial markets [37]. But often, some type of time-series show structural changes over time, i.e. completely different dynamic regime than before. Thus, having multiple dynamical systems such that each one describes just a specific subset of the dynamics, makes perfect sense. An alternative to having nonlinear dynamic system for capturing the dynamics of a time-series, a few researchers look into having multiple linear dynamic systems for different regimes of the time-series and a switching variable (or process) that chooses at each time step between them. This is called jump-linear system. In the model presented here, the switching is done probabilistically, based on a discrete-valued *mode* of the system. If the latent mode is a discrete-time Markov process then the model is called Markov-jump linear system or switching linear dynamic system. This uses a HMM for the switching dynamics. However, we have to keep in mind the fact that the change from one regime to the other is time-dependent also, with some regimes being encountered more often, while some are rarely seen. This is an extension to the well-known Hidden Markov Model (HMM), with the difference that the SLDS does not make any Markovian assumption (the HMM makes the assumption that the observations are conditionally independent given the mode). Some approaches for the SLDS include fixing the number of HMM modes [36] or defining a changepoint detection mechanism where each change is to a new, unseen dynamical system [38]. Thus, an important problem in SLDS is the identification of the different modes, or regimes associated with each mode. One solution to this [39] is to project the data into a higher dimensional space and then segment the data into different subspaces. This is an algebraic approach and assumes deterministic dynamics of the system, even though the authors in [39] state that their system is robust to some reasonable amount of noise. Some other interesting line of work, which we will describe in a subsequent section is the work of Ghahramani and Hinton [40] that after assuming that the number of modes is known in advance, they use a variational approach to the segmentation of data and learning the parameters of the distinct LDSs. The research described next, does not fix the number of modes beforehand and also allows the system to go back to previously met regimes, also not assuming deterministic dynamics.

## 5.2 Nonparametric Bayesian learning of Switching Linear Dynamic Systems

The authors in [41] develop a model that includes a hierarchical Dirichlet process (HDP) as the prior on the parameters of the HMM with unknown number of modes. Employing Bayesian nonparametrics they are able to learn an unknown number of dynamical regimes, and then using automatic relevance determination (ARD) they are able to learn which components account for each mode of the overall system dynamics. We have to note that the individual LDSs can have of course different dimensionality. The Bayesian approach employed chooses the prior such that is penalizes more complex models in two ways: a higher number of modes of the overall system is obviously not desirable and thus is penalized more and also the higher dimensionality of the dynamics describing each mode is also penalized more than a lower dimensionality. Instead of hard constraints on the model, they increase the posterior probability for simpler models. They contrast this with the penalized likelihood approach employing Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC) by stating that their approach is purely Bayesian. [Give a short overview of the algorithm]. The SLDS can be described by the equations:

$$
\begin{aligned}
z_t &\sim \pi_{z_{t-1}} \\
\mathbf{x}_t &= A^{(z_t)}\mathbf{x}_{t-1} + \mathbf{e}_t(z_t) \\
\mathbf{y}_t &= C\mathbf{x}_t + \mathbf{w}_t
\end{aligned}
$$

where $z_t$ is the mode that we talked about earlier, at time $t$, and is defined by discrete Markov process with the transition distributions being $\pi_j$. Here they assumed the process noise to be specific to the mode, meaning (as opposed to the measurement noise, which is not):

$$
\mathbf{e}_t(z_t) \sim \mathcal{N}(0, \Sigma_(z_t))
$$

The authors state that the measurement matrix $C$ and the measurement noise $\mathbf{w}$ could both be dependent on the mode, however this can impair the identifiability of the model [Fox]. The research in [Fox] is extensive, thus we will state here just one of the most important consideration of their SLDS learning algorithm, i.e. automatic relevance determination [42] such that model parameters are driven to 0 if their presence is not supported by the data. Their model (HDP-SDLS) is placing independent, zero mean, spherical symmetric Gaussian priors on the columns of the matrix $\mathbf{A}$:

$$
p(\mathbf{A}(k)|\alpha(k)) = \prod_{j=1}^{n} \mathcal{N}(\mathbf{a}_j^(k); 0, \alpha_j^{-(k)} I_n)
$$

All precisions $\alpha_j^(k)$ are given gamma priors. The Gaussians priors penalize non-zero columns of the matrix $\mathbf{A}$ through the precision parameters. Estimation of these hyperparameters in an iterative manner, leads to large $\alpha_j^(k)$ for columns for which the data is insufficient for overcoming the penalty imposed by the prior. When $\alpha_j^(k) \to \infty$ then $\mathbf{a}_j^(k) \to 0$, which means that the $j^{th}$ component does not contribute to the underlying dynamics given by the $k^{th}$ mode. The authors also formulate the algorithm as a message passing algorithm. For details see [41]

## 5.3 Variational learning of Switching State Space Models

The work presented here [40] is the first to perform variational approximation for inference in a SSSM, thus it is fundamental for subsequent developments of the field. As before, we have M real-valued state vectors, $X_t^(m)$ describing the LDSs and one discrete state vector $S_t$, that

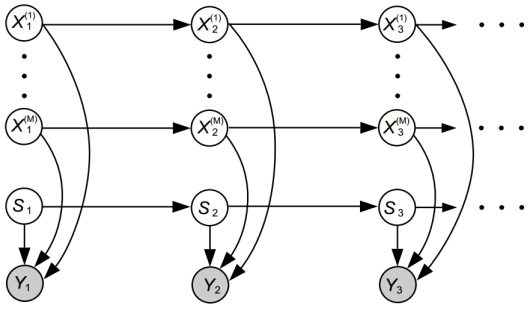


Figure 7: Graphical model for the SSSM.

can take M discrete values:$1, .., M$. This is known as the switch variable. The joint probability of hidden and observed states is thus:

$$p(S_t, X_t^{(1)}, ..X_t^{(M)}, Y_t) = p(S_1)\prod_{t=2}^{T} p(s_t|s_{t-1}) \prod_{m=1}^{M} p(X_1^{(m)}) \prod_{t=2}^{T} p(X_t^{(m)}|X_{t-1}^{(m)})$$
$$\prod_{t=1}^{T} p(Y_t|X_t^{(1)}, ..X_t^{(M)}, S_t)$$

The corresponding graphical model can be seen in Figure 7. When conditioned on a specific value for the switch variable $S_t = m$, the observed variable is a multivariate Gaussian with the output given by the model $m$. The probability of the observation vector $Y_t$ is given by:

$$p(Y_t|X_t^{(1)}, ..X_t^{(M)}, S_t = m) = (2\pi)^{-D/2}|R|^{-1/2} exp\left\{-\frac{1}{2}(Y_t - C^{(m)}X_t^{(m)})'R^{-1}(Y_t - C^{(m)}X_t^{(m)})\right\}$$

where D is the dimensionality of the observation vector, R is the observation noise covariance matrix, and $C^m$ is the output matrix for model $m$. Each state vector $X_t$ evolves independently according to a linear Gaussian dynamics with a different transition matrix, initial state and process noise. The switch state evolves according to the initial state probability $p(S_1)$ and the state transition matrix $p(S_t|S_{t-1})$. An analogy with the *mixture of experts* architecture is obvious, where the switch functions as a gating mechanisms with Markovian dynamics. We have to mention that in this specific paper, the authors use the same dimensionality for each LDS, even though, they say that the extension to distinct dimensionalities is straightforward.

### 5.3.1 Learning

Employing a variational approach, the usual decomposition of the log-posterior is the same as in the previous section on Variational Inference, where in this case we have an additional dependency of $p$ and $q$ on $S_t$. The approximation of the posterior is similarly done by minimization of the KL-divergence between the approximating posterior and the true posterior employing an EM-like algorithm. In this case the decomposition of the approximating posterior $q$ is given by:

$$q(S_t, X_t) = \frac{1}{Z_q}\left[\psi(S_1)\prod_{t=2}^{T}\psi(S_{t-1}, S_t)\right]\prod_{m=1}^{M}\psi(X_1^{(m)})\prod_{t=2}^{T}\psi(X_{t-1}^{(m)}, X_t^{(m)})$$

where $Z_q$ is the normalization coefficient, ensuring that $q$ is a valid pdf and $\psi$ are unnormalized probabilities, called potential functions, given by:

$$\psi(S_1 = m) = p(S_1 = m)q_1^{(m)}$$
$$\psi(S_{t-1}, S_t = m) = p(S_t = m|S_{t-1})q_t^{(m)}$$

where $q_t^{(m)}$ are the variational parameters of the approximating distribution $q$. The terms that include $S_t$ define a Markov chain, while the terms that include $X_t^{(m)}$ define a set of M uncoupled state-space models. The authors removed couplings from the original system resulting from the fact that at time $t$, the observation depends on all hidden variables at time $t$. They keep the couplings between successive states of the hidden variables, and employ forward-backward and Kalman smoothing recursions. This gives a structured variational approximation, by keeping some of the original structure of the system. The uncoupled SSMs in the approximation $q$ have associated potentional functions that are related to the probabilities in the original system. The prior and the transition probabilities are multiplied by a factor that changes the potentials such that they account for the data:

$$\psi(X_1^{(m)}) = p(X_1^{(m)}) \left[p(Y_1|X_1^{(m)}, S_1 = m)\right]^{h_1^{(m)}}$$
$$\psi(X_{t-1}^{(m)}, X_t^{(m)}) = p(X_t^{(m)}|X_{t-1}^{(m)}) \left[p(Y_t|X_t^{(m)}, S_1 = m)\right]^{h_1^{(m)}}$$

where $h_t^{(m)}$ are called *responsabilities* and represent the fact that some specific model $m$ generated some specific observation $Y_t$ (when is close to 1) or not (close to 0). The KL divergence satisfies the fixed point equations for the variational parameters:

$$h_t^{(m)} = q(S_t = m)$$
$$q_t^{(m)} = exp\left\{-\frac{1}{2}\langle(Y_t - C^{(m)}X_t^{(m)})'R^{-1}(Y_t - C^{(m)}X_t^{(m)})\rangle\right\}$$

The re-estimation for the switch process is done using an algorithm similar to Baum-Welch for the HMM. We give next an overview of the algorithm presented in [GH$_switch$] :

**Variational learning for SSSM**

Initialize parameters of the model

Repeat until bound on log likelihood has converged

- **E-step** Repeat until convergence of KL(q——p):
  1. Compute $q_t^{(m)}$ from the prediction error of the model $m$ on observation $Y_t$
  2. Compute $h_t^{(m)}$ using the forward-backward algorithm on the HMM, with observation probabilities $q_t^{(m)}$
  3. For $m = 1$ to $M$
     Run Kalman smoothing recursion, using data weighted by $h_t^{(m)}$
- **M-step**
  1. Re-esimtate parameters for each SSM using data weighted by $h_t^{(m)}$
  2. Re-estimate parameters for the switching process using Baum-Welch update equations

## 5.4 Expectation Propagation for SLDS with message passing

EP has been shown to have great potential for recursive filtering problems. The work of Zoeter and Heskes [43] extends EP to the SLDS model. We first show the factor graph on which the EP agorithm is run and on which the reasoning for this section will take place, in Figure **??**. The notation is as before: $\mathbf{x}, \mathbf{y}, s$ are the latent vector, observation vector and switching variable respectively, while the authors also denote $\mathbf{z}_t \triangleq \{s_t, \mathbf{x}_t\}$ as a single conditionally Gaussian (CG) distributed random variable (for more details and standard operations of CG see [43]). In the graphical model, we see that $\mathbf{y}$ do not appear, as they are always observed and part of the factors. We denote by $\psi$ the potential functions given by:

$$\psi_t(\mathbf{z}_{t-1}, \mathbf{z}_t) \triangleq p(\mathbf{y}, \mathbf{z}_t|\mathbf{z}_{t-1}, \theta) = p(\mathbf{y}_t|\mathbf{x}_t, s_{t-1}, s_t)p(\mathbf{x}_t|\mathbf{x}_{t-1}, s_{t-1}, s_t, \theta)p(s_t|s_{t-1}, \theta)$$
$$\psi_1(\mathbf{z}_0, \mathbf{z}_1) \triangleq p(\mathbf{y}_1, \mathbf{z}_1|\theta) = p(\mathbf{y}_t|\mathbf{x}_1, s_1, \theta)p(\mathbf{x}_1|s_1, \theta)p(s_1|\theta)$$

Approximate one-slice marginals are denoted by $\widehat{q}(\mathbf{z}_t) \sim p(\mathbf{z}_t|\mathbf{y}_{t-1})$ and the forward messages by $\alpha_t(\mathbf{z}_t)$. These forward messages have a similar purpose as in the regular Kalman filter. In the exact filter the messages satisfy: $\alpha_t(\mathbf{z}_t) \propto p(\mathbf{z}_t|\mathbf{y}_{1:t}, \theta)$, but here they are approximations. Having defined the notation we will explain the main points of the algorithm in short. Note that the marginal: $\widehat{p}_t(\mathbf{z}_t) = \sum_{\mathbf{z}_{t-1}} \widehat{p}_t(\mathbf{z}_{t-1}, \mathbf{z}_t)$ is a mixture of Gaussians with M components, and if one would use $\widehat{p}_t(\mathbf{z}_t)$ as the new forward message, even though we would have exact results, at the next step of the recursion the number of components in the joint would increase by a factor M, which would mean an exponential increase in the number of timesteps. To avoid this, $\widehat{p}_t(\mathbf{z}_t)$ is approximated by the CG distribution closest to $\widehat{p}_t(\mathbf{z}_t)$ in the KL sense, i.e.:

$$\widehat{q}_t(\mathbf{z}_t) = Collapse(\widehat{p}_t(\mathbf{z}_t))$$

If just one forward pass is performed, then approximate beliefs $\widehat{q}_t(\mathbf{z}_t)$ are based just on $\mathbf{y}_{1:t}$. Thus, the authors set $\alpha_t(\mathbf{z}_t) = \widehat{q}_t(\mathbf{z}_t)$. For smoothing, we need a backward pass as well, and thus we denote by $\beta_t(\mathbf{z}_t)$ with $t = 1, 2, .., T$ the backward messages, which are similar to the backward messages in the HMM smoother. For the exact case, these are given by:

$$\beta_t(\mathbf{z}_t) \propto p(\mathbf{y}_{t+1:T}|\mathbf{z}_t, \theta)$$

such that

$$\alpha_t(\mathbf{z}_t)\beta_t(\mathbf{z}_t) \propto p(\mathbf{z}_t|\mathbf{y}_{1:t}, \theta) \tag{6}$$

However, in the current scheme we are approximating this quantity with $\widehat{q}_t(\mathbf{z}_t)$. We note that compared to the forward messages, the backward messages cannot always be normalized, as the we see that they are conditioned on $\mathbf{z}_t$, and when integrating over $\mathbf{z}_t$ this does not always give a finite value. Thus, the authors propose to first construct beliefs for the messages, then approximate the beliefs, and then deduce the messages from approximated beliefs. Let us give a more detailed description of this process. First $\beta_T = 1$, then for $t \leq T$, the message $\beta_{t-1}$ is computed as a function of $\beta_t$, the local potential $\psi_t$ and the forward message $\alpha_{t-1}$. Having this quantities, an approximated two-slice posterior belief is given by:

$$\widehat{p}_t(\mathbf{z}_{t-1}, \mathbf{z}_t) \propto \alpha_{t-1}(\mathbf{z}_{t-1})\psi_t(\mathbf{z}_{t-1}, \mathbf{z}_t)\beta_t(\mathbf{z}_t)$$

As in the forward pass the marginal $\widehat{p}_t(\mathbf{z}_{t-1})$ is a conditional mixture of Gaussians and not a simple CG. Because $\widehat{p}_t(\mathbf{z}_{t-1})$ is a proper distribution then $\widehat{q}_t(\mathbf{z}_{t-1})$ is well defined:

$$\widehat{q}_t(\mathbf{z}_{t-1}) = Collapse(\widehat{p}_t(\mathbf{z}_{t-1}))$$

which gives, from Equation 6:

$$\widehat{q}_t(\mathbf{z}_{t-1}) = \alpha_{t-1}(\mathbf{z}_{t-1})\beta_{t-1}(\mathbf{z}_{t-1})$$

The message $\alpha_{t-1}(\mathbf{z}_{t-1})$ is kept fixed in this recursive step, so $\beta t - 1(\mathbf{z}_{t-1})$ can be computed as in the last step of the algorithm. In the next forward pass, the new message $\alpha_t$ is a function of the old message $\alpha_{t-1}$, the local potential $\psi_t$ and the backward message $\beta_t$, by constructing $\widehat{q}_t(\mathbf{z}_t)$ and dividing by $\beta_t(\mathbf{z}_t)$. We give the complete algorithm as described in [43].

**Expectation propagation for the SLDS**:

- Compute a forward pass by performing the following steps for $t = 1, 2, ..., T$ with $t' \triangleq t$ and a backward pass by performing the same steps for $t = T - 1, T - 2, ..., 1$ with $t' \triangleq t - 1$. Possibly iterate forward-backward passes until convergence. At the boundaries keep $\alpha_0 = \beta_T = 1$.
  1. Construct a two-slice belief $\widehat{p}(\mathbf{z}_{t-1}, \mathbf{z}_t) \propto \alpha_{t-1}(\mathbf{z}_{t-1})\psi(\mathbf{z}_{t-1}, \mathbf{z}_t)\beta_t(\mathbf{z}_t)$
  2. Marginalize to obtain a one-slice marginal $\widehat{p}(\mathbf{z}_{t'}) = \sum_{\mathbf{z}_{t''}} \widehat{p}(\mathbf{z}_{t-1}, \mathbf{z}_t)$, with $t'' \triangleq \{t-1, t\}^{''}$.
  3. Find $\widehat{q}_{t'}(\mathbf{z}_{t'})$ that approximates $\widehat{p}_t(\mathbf{z}_{t'})$ best in the KL sense $\widehat{q}_{t'}(\mathbf{z}_{t'}) = \text{Collapse}(\widehat{p}_t(\mathbf{z}_{t'}))$
  4. Infer the new messages by division

$$\alpha_t(\mathbf{z}_t) = \frac{\widehat{q}_t(\mathbf{z}_t)}{\beta_t(\mathbf{z}_t)} \qquad \beta_{t-1}(\mathbf{z}_{t-1}) = \frac{\widehat{q}_{t-1}(\mathbf{z}_{t-1})}{\alpha_{t-1}(\mathbf{z}_{t-1})}$$

# 6 Conclusion

Graphical models naturally lend themselves to message passing algorithms. We saw different schemes for approximate inference making use of message passing algorithms. For example, the Kalman Filter can also be modeled as a message passing algorithm in a factor graph [24], also the forward-backward algorithm, as well as the Viterbi algorithm. These are modeled as special cases of the sum-product algorithm on factor graphs. Particle filters have been also formulated as message passing algorithms [44].As a general idea, in a graphical model the local computation taking place at each node (being a sum, product, or more complex computation) can be summarized in some way by the node and then shared in the system by messages transmitted. This is the core idea of message passing, splitting the overall computation into local computations specific to each node. Of course one of the main assumptions is that the joint distribution factorizes in some way, such that the computation can be split between smaller subgraphs, or even nodes (as in the variational approach). Even though in the variational inference paradigm, the message passing assumes a fully factorized distribution, extensions exist, such that some of the original structure in the graph can be kept [45] (sometimes exact inference can be performed in these subgraphs) and still make use of the variational approximation. When it comes to EP we saw multiple approaches employing different types of message passing schemes, also ADF can be formulated in the spirit of the sum-product algorithm [43]. We saw that for some message passing algorithms the scheduling of the messages matters, while for others not too much. However sometimes, specific scheduling schemes can improve convergence speed. An interesting idea would be to put a constraint on the scheduling of the message passing scheme such that it satisfies some criterion, or even employ heuristics in the scheduling algorithm. We saw also that for different approximations

schemes the messages have different properties, it would be interesting to analyze what kind of properties should the messages have such that they optimize some measure. In other words, what kind of information should be propagated through the graphical model such that inference is efficient and accurate. Is there a general rule for this ? What about the type of messages, how many types should be used ? What should be their characteristics ? Another interesting aspect of message passing algorithms is that they are naturally suited for large-scale distributed computing, as the nodes (or sets of nodes) could be split between different computers and then each local computation would be independent of the others, with a minimum overhead of transmitting structured information over the network.

# References


[1] S. E. Fienberg *et al.*, "When did bayesian inference become" bayesian"?," *Bayesian analysis*, vol. 1, no. 1, pp. 1–40, 2006.

[2] J. Neyman, "Outline of a theory of statistical estimation based on the classical theory of probability," *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 236, no. 767, pp. 333–380, 1937.

[3] A. J. Krener, "Kalman-bucy and minimax filtering," *Automatic Control, IEEE Transactions on*, vol. 25, no. 2, pp. 291–292, 1980.

[4] V. N. Vapnik and A. J. Chervonenkis, "Theory of pattern recognition," 1974.

[5] H. Akaike, "Information theory and an extension of the maximum likelihood principle," in *Selected Papers of Hirotugu Akaike*, pp. 199–213, Springer, 1998.

[6] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, no. 5, pp. 465–471, 1978.

[7] R. A. Fisher, "Statistical methods and scientific inference.," 1956.

[8] G. A. Barnard, "Pivotal models and the fiducial argument," *International Statistical Review/Revue Internationale de Statistique*, pp. 309–323, 1995.

[9] V. Vapnik, "Statistical learning theory. 1998," 1998.

[10] A. H. Jazwinski, *Stochastic Processes and Filtering Theory*. Academic Press, 1970.

[11] Y.-C. Ho and R. Lee, "A bayesian approach to problems in stochastic estimation and control," *Automatic Control, IEEE Transactions on*, vol. 9, no. 4, pp. 333–339, 1964.

[12] G. Kallianpur, *Stochastic filtering theory*. Springer, 1980.

[13] V. Peterka, "Bayesian approach to system identification," *Trends and Progress in System identification*, vol. 1, pp. 239–304, 1981.

[14] B. Anderson, "0., and moore, jb (1979), optimal filtering," 1979.

[15] D. F. Kerridge, "Inaccuracy and inference," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 184–194, 1961.

[16] R. Kulhavỳ, "Recursive nonlinear estimation: A geometric approach," *Automatica*, vol. 26, no. 3, pp. 545–555, 1990.

[17] C. Robert, "The bayesian choice. 2001," 2001.

[18] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[19] B. A. McElhoe, "An assessment of the navigation and course corrections for a manned flyby of mars or venus," *Aerospace and Electronic Systems, IEEE Transactions on*, no. 4, pp. 613–623, 1966.

[20] G. A. Einicke, "Smoothing, filtering and prediction: Estimating the past, present and future," *New York: InTech*, 2012.

[21] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pp. 153–158, IEEE, 2000.

[22] S. J. Julier and J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems," in *AeroSense'97*, pp. 182–193, International Society for Optics and Photonics, 1997.

[23] B. J. Frey and D. J. MacKay, "A revolution: Belief propagation in graphs with cycles," *Advances in neural information processing systems*, pp. 479–485, 1998.

[24] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 498–519, 2001.

[25] M. Wainwright, T. Jaakkola, and A. Willsky, "Tree consistency and bounds on the performance of the max-product algorithm and its generalizations," *Statistics and Computing*, vol. 14, no. 2, pp. 143–166, 2004.

[26] A. T. Ihler, J. Iii, and A. S. Willsky, "Loopy belief propagation: Convergence and effects of message errors," in *Journal of Machine Learning Research*, pp. 905–936, 2005.

[27] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *et al.*, *Introduction to algorithms*, vol. 2. MIT press Cambridge, 2001.

[28] J. M. Winn and C. M. Bishop, "Variational message passing," in *Journal of Machine Learning Research*, pp. 661–694, 2005.

[29] T. P. Minka, "Expectation propagation for approximate bayesian inference," in *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pp. 362–369, Morgan Kaufmann Publishers Inc., 2001.

[30] H. Attias, "A variational bayesian framework for graphical models," *Advances in neural information processing systems*, vol. 12, no. 1-2, pp. 209–215, 2000.

[31] Z. Ghahramani and M. J. Beal, "Propagation algorithms for variational bayesian learning," *Advances in neural information processing systems*, pp. 507–513, 2001.

[32] A. Corduneanu and C. M. Bishop, "Variational bayesian model selection for mixture distributions," in *Artificial intelligence and Statistics*, vol. 2001, pp. 27–34, Morgan Kaufmann Waltham, MA, 2001.

[33] M. Deisenroth and S. Mohamed, "Expectation propagation in gaussian process dynamical systems," in *Advances in Neural Information Processing Systems*, pp. 2609–2617, 2012.

[34] V. Pavlovic, J. M. Rehg, and J. MacCormick, "Learning switching linear models of human motion," in *NIPS*, pp. 981–987, Citeseer, 2000.

[35] E. B. Fox, E. B. Sudderth, and A. S. Willsky, "Hierarchical dirichlet processes for tracking maneuvering targets," in *Information Fusion, 2007 10th International Conference on*, pp. 1–8, IEEE, 2007.

[36] S. M. Oh, J. M. Rehg, T. Balch, and F. Dellaert, "Learning and inferring motion patterns using parametric segmental switching linear dynamic systems," *International Journal of Computer Vision*, vol. 77, no. 1-3, pp. 103–124, 2008.

[37] M. E. P. So, K. Lam, and W. K. Li, "A stochastic volatility model with markov switching," *Journal of Business & Economic Statistics*, vol. 16, no. 2, pp. 244–253, 1998.

[38] X. Xuan and K. Murphy, "Modeling changing dependency structure in multivariate time series," in *Proceedings of the 24th international conference on Machine learning*, pp. 1055–1062, ACM, 2007.

[39] K. Huang, A. Wagner, and Y. Ma, "Identification of hybrid linear time-invariant systems via subspace embedding and segmentation (ses)," in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, vol. 3, pp. 3227–3234, IEEE, 2004.

[40] Z. Ghahramani and G. E. Hinton, "Variational learning for switching state-space models," *Neural computation*, vol. 12, no. 4, pp. 831–864, 2000.

[41] A. S. Willsky, E. B. Sudderth, M. I. Jordan, and E. B. Fox, "Nonparametric bayesian learning of switching linear dynamical systems," in *Advances in Neural Information Processing Systems*, pp. 457–464, 2009.

[42] M. J. Beal, *Variational algorithms for approximate Bayesian inference*. PhD thesis, University of London, 2003.

[43] O. Zoeter and T. Heskes, "Expectation propagation and generalised ep methods for inference in switching kalman filter models," in *Probabilistic Methods for Time-Series Analysis* (D. Barber, A. T. Cemgil, and S. Chiappa, eds.), pp. 181–207, Cambridge University Press, 2011.

[44] J. Dauwels, S. Korl, and H.-A. Loeliger, "Particle methods as message passing," in *Information Theory, 2006 IEEE International Symposium on*, pp. 2052–2056, IEEE, 2006.

[45] D. Geiger, C. Meek, and Y. Wexler, "A variational inference procedure allowing internal structure for overlapping clusters and deterministic constraints.," *J. Artif. Intell. Res.(JAIR)*, vol. 27, pp. 1–23, 2006.