Software Analysis Unifying Particle Filtering and Marginalized Particle Filtering

Václav Šmídl
Institute of Information Theory and Automation,
Prague, Czech Republic.
smidl@utia.cas.cz

Abstract – *Particle filtering has evolved into wide range of techniques giving rise to many implementations and specialized algorithms. In theory, all these techniques are closely related, however this fact is usually ignored in software implementations. In this paper, particle filtering is studied together with marginalized particle filtering and a generic software scheme unifying these two areas is proposed. It is presented in general terms of object-oriented programming so that it may be implemented in existing Bayesian filtering toolboxes that are briefly reviewed. The power of the approach is illustrated on a new variant of the marginalized particle filter. A range of new variants of the filter is obtained by plugging this class into the proposed software structure. The framework and the illustrative example is implemented in the BDM library.*

Keywords: Marginalized particle filter, software analysis, Bayesian filtering

1 Introduction

Implementation of algorithms of Bayesian filtering, or filtering in general, are of two kind: generic, or application specific. The latter is much more common in practical applications due specific restrictions on computational time or hardware platform. Typical examples are implementation of Extended Kalman filter for low level control of an electrical drive [6], or marginalized particle filter for robot navigation [10]. The generic implementations are less common, their purpose being educational or deployment of modified version of the algorithm within a chosen domain.

Software analysis is crucial for both kinds of implementations, the stress is however on different aspect. The key problem of the application-specific implementation is to adjust the chosen algorithm to the fit the application domain and its restrictions. The key problem of software analysis for generic implementation is logical consistency and certain "elegance" in a way that potential users find the implementation easy to use.

In this paper, we are concerned with a generic implementation of Bayesian filtering in general, with special attention paid to implementation of the marginalized particle filter (MPF) [9], also known as Rao-Blackwellized particle filter [3]. This type of filter is a combination of an analytic filter with a particle filter, both components which are already available in many toolboxes of Bayesian filtering. Majority of generic toolboxes is designed using object-oriented approach, hence we will study the filters from this perspective. At the time of writing, the following noteworthy toolboxes are designed using object oriented approach:

Bayes++, BFL, BDM C++ libraries of Bayesian filters, available from `http://bayesclasses.sourceforge.net/`, `http://www.orocos.org/bfl`, and `http://mys.utia.cas.cz:1800/trac/bdm/`, respectively.

Kalmtool, NEF Matlab toolboxes for Bayesian filtering, available from `http://www.iau.dtu.dk/research/control/kalmtool.html`, and `http://nft.kky.zcu.cz/nef.html`, respectively.

None of these currently provides marginalized particle filter in its general form. Discussion how to implement the MPF in these toolboxes will be provided. A reference implementation is available within the Bayesian Decision-Making library (BDM).

1.1 Issues of Object-oriented approach

The following features of object-oriented approach [1] are attractive for implementation Bayesian filtering: (i) *inheritance*, which allows to create new algorithms by an extension of the previous algorithm, (ii) *polymorphism*, which allows writing generic algorithms, and (iii) *encapsulation* of data structures (*attributes*) and *methods* that operate on these, which improves scalability of the solution. The danger of this approach lies in overusing these tools. The most susceptible to overusing is the mechanism of inheritance. When two classes having something in common are to be implemented, it is possible to create a class representing their common attributes or methods and design the two original classes as its specialization. This way, the common

parts are implemented only in one place, however, at the price of maintaining an extra class the meaning of which may be unclear to others. Too many level of abstraction can make the software design hard to understand and learn. Finding the right balance in this issue is a delicate but very important task.

In this paper, we present only conceptual levels of abstraction, from which the actual implementation is expected to vary. Differences in implementation may depend on the chosen implementation language and what other classes are already implemented in the toolbox.

2 Theory of Bayesian Filtering

We are concerned with the classical problem of inference of state variables parametrizing a sequence of observation models in the following manner:

$$y_t \sim p(y_t|x_t, u_t), \quad x_t \sim p(x_t|x_{t-1}, u_t). \tag{1}$$

Here, x_t is a vector known as the state variable, y_t are the observations and u_t is a vector of observed exogenous input. Variable $d_t = [y'_t, u'_t]$ aggregate all observable quantities. By *Bayesian Filtering*, we mean the recursive evaluation of the filtering distribution, $p(x_t|d_{1:t})$, using Bayes' rule [7, 3]:

$$p(x_t|d_{1:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|d_{1:t-1})\, dx_{t-1}, \tag{2}$$

$$p(x_t|d_{1:t}) = \frac{p(y_t|x_t, u_t)p(x_t|d_{1:t-1}),}{p(y_t|u_t, d_{1:t-1})} \tag{3}$$

$$p(y_t|u_t, d_{1:t-1}) = \int p(y_t|x_t, u_t)p(x_t|d_{1:t-1})dx_t \tag{4}$$

where $p(x_1|)$ is the prior distribution, and $d_{1:t} = [d_1, \ldots, d_t]$ denotes the set of all observations. The integration in (2), and elsewhere in this paper, is over the whole support of the involved probability density functions (pdf). The denominator of (3) can be interpreted in two possible ways: (i) as a density on y_t, which is known as predictive density from time $t-1$, or (ii) as a numerical quantity with substituted observed values of y_t and u_t, which is known as marginal likelihood or evidence.

2.1 Particle Filtering (PF)

Particle filtering (PF) [3] refers to a range of techniques for generating an empirical approximation of $p(x_{1:t}|d_{1:t})$, where $x_{1:t} = [x_1, \ldots, x_t]$ is the state trajectory:

$$p(x_{1:t}|d_{1:t}) \approx \frac{1}{n}\sum_{i=1}^{n} \delta(x_{1:t} - x_{1:t}^{(i)}), \tag{5}$$

where $x_{1:t}^{(i)}$, $i = 1, \ldots, n$ are i.i.d. samples from the posterior and $\delta(\cdot)$ denotes the Dirac δ-function. Therefore, this approach is feasible only if we can sample from the exact posterior, $p(x_{1:t}|d_{1:t})$. If this is not the case, we can draw samples from a chosen proposal distribution (importance function), $q(x_{1:t}|d_{1:t})$, as follows:

$$\begin{aligned} p(x_{1:t}|d_{1:t}) &= \frac{p(x_{1:t}|d_{1:t})}{q(x_{1:t}|d_{1:t})} q(x_{1:t}|d_{1:t}) \\ &\approx \frac{p(x_{1:t}|d_{1:t})}{q(x_{1:t}|d_{1:t})} \frac{1}{n}\sum_{i=1}^{n} \delta(x_{1:t} - x_{1:t}^{(i)}). \end{aligned} \tag{6}$$

Using the sifting property of the Dirac δ-function, the approximation can be written in the form of a *weighted* empirical distribution, as follows:

$$p(x_{1:t}|d_{1:t}) \approx \sum_{i=1}^{n} w_t^{(i)} \delta(x_{1:t} - x_{1:t}^{(i)}), \tag{7}$$

$$w_t^{(i)} \propto \frac{p(x_{1:t}^{(i)}|d_{1:t})}{q(x_{1:t}^{(i)}|d_{1:t})}. \tag{8}$$

Under this *importance sampling* procedure, the true posterior distribution need only be evaluated point-wise. Furthermore, normalizing constant of $p(\cdot)$ is not required, since (7) can be normalized trivially via a constant $c = \sum_{i=1}^{n} w_t^{(i)}$.

The challenge for on-line algorithms is to achieve recursive generation of the samples and evaluation of the importance weights. Using (1) and standard Bayesian calculus, (8) may be written in the following recursive form:

$$w_t^{(i)} \propto \frac{p(y_t|x_t^{(i)}, u_t)p(x_t^{(i)}|x_{t-1}^{(i)})}{q(x_t^{(i)}|x_{1:t-1}^{(i)}, d_{1:t})} w_{t-1}^{(i)}. \tag{9}$$

where, now, $x_t^{(i)}$ are drawn from the denominator of (9), which can be chosen as $p(x_t|x_{t-1})$.

2.1.1 Choosing the number of particles

Convergence of the particle filter to the true posterior density is proved for $n \to \infty$. Thus, the accuracy of approximation grows with more particles, as does the associated computational cost. Optimal trade-off between accuracy and cost can be done experimentally, or it can also be optimized on-line, see [4] and followup work.

2.1.2 Proposal distribution

The choice of the right importance function is often key to success of a particle filter in the a particular problem. Substantial amount of research was dedicated to this issue, yielding a range of options, see [12] for survey. An interesting direction is to run a local filter within a particle filter and use its posterior density as a proposal. In fact, other proposal distributions can be interpreted as approximate Bayesian filtering.

2.1.3 Adaptive Monte Carlo

Another approach to improvement of proposal densities is adaptive Monte Carlo technique [2], where the proposal density is parametrized by an unknown parameter which is estimated recursively with growing number of generated particles. Thus, this approach combines choice of the number of particles and improvement of the proposal.

2.2 Marginalized Particle Filtering (MPF)

The main advantage of importance sampling is its generality. However, it may be computationally prohibitive to draw samples from the possibly high dimensional state space of x_t. Furthermore, it is necessary to generate large numbers of such particles in these cases in order to achieve an acceptable error of approximation. These problems can be overcome in cases where the structure of the model (1) allows analytical marginalization over a subset, $x_{1,t}$, of the full state vector $x'_t = [x'_{1,t}, x'_{2,t}]$ [3, 9]. Therefore, we consider the factorization

$$p(x_{1:t}|d_{1:t}) = p(x_{1,1:t}|x_{2,1:t}, d_{1:t})p(x_{2,1:t}|d_{1:t}),$$

where $p(x_{1,1:t}|x_{2,1:t}, d_{1:t})$ is analytically tractable, while $p(x_{2,1:t}|d_{1:t})$ is not. We replace the latter by a weighted empirical distribution, in analogy to (6), yielding

$$p(x_{1:t}|d_{1:t}) \approx \sum_{i=1}^{n} w_t^{(i)} p(x_{1,1:t}|x_{2,1:t}^{(i)}, d_{1:t})\delta(x_{2,1:t} - x_{2,1:t}^{(i)}), \tag{10}$$

$$w_t^{(i)} \propto \frac{p(x_{2,1:t}^{(i)}|d_{1:t})}{q(x_{2,1:t}^{(i)}|d_{1:t})}. \tag{11}$$

Note that we now only have to sample from the space of $x_{2,t}$. The weights can, once again, be evaluated recursively:

$$w_t^{(i)} \propto \frac{p(d_t, x_{2,t}^{(i)}|x_{2,t-1}^{(i)}, d_{1:t-1})}{q(x_{2,t}|x_{2,1:t-1}^{(i)}, d_{1:t})} w_{t-1}^{(i)}. \tag{12}$$

Note that term $p(d_t, x_{2,t}^{(i)}|x_{2,t-1}^{(i)}, d_{1:t-1})$ in numerator of (12) is equivalent to the normalizing constant of a Bayesian filter that treats $x_{2,t}^{(i)}$ as observations:

$$p(x_{1,t}|x_{2,1:t}, d_{1:t}) = \frac{p(d_t|x_{1,t}, x_{2,t})p(x_{2,t}|x_{1,t}, x_{t-1})p(x_{1,t}|x_{t-1})}{p(d_t, x_{2,t}|x_{2,1:t-1}, d_{1,t-1})} \times p(x_{1,t-1}|x_{2,1:t-1}, d_{1,t-1}) \tag{13}$$

2.2.1 Special cases

In this section we analyze special cases, where (13) is analytically tractable with a reasonable candidate for the proposal function :

1. conditionally independent evolution of $x_{2,t}$, i.e. $p(x_{2,t}|x_{1,t-1}, x_{2,t-1}) = p(x_{2,t}|x_{2,t-1})$, then

$$p(d_t, x_{2,t}^{(i)}|x_{2,t-1}^{(i)}, d_{1:t-1}) = p(d_t|x_{2,t}^{(i)}, d_{1:t-1})p(x_{2,t}^{(i)}|x_{2,t-1}^{(i)}). \tag{14}$$

   where integration over $x_{1,t}$ is performed only for $p(d_t|x_{2,t}^{(i)}, d_{1:t-1})$. This is a commonly encountered case [13], in which standard sampling from the parameter evolution, $p(x_{2,t}|x_{2,t-1})$, can be used.

2. analytically tractable subspace, $x_{1,t}$ allows further partitioning of $x_{1,t} = [x_{1d,t}, x_{1x,t}]$, for which (14) also arise, with each element in the chain rule marginalized independently, i.e. $p(d_t|\cdot)$ over $x_{1d,t}$ and $p(x_2^{(i)}|\cdot)$ over $x_{2x,t}$. The latter density can be again used as the proposal. This is the case studied in [8].

3. integration can not be simplified, and must be performed over the whole space $x_{1,t}$, however, the predictive density $p(d_t, x_{2,t}|x_{2,t-1}^{(i)}, d_{1:t-1})$ is analytically tractable. In that case, the proposal density can be obtained by marginalization over d_t. An example of this case is given in Section 4.1.

2.3 Software representation

The number of possible configurations of a particle filter is rather high, since techniques in Sections 2.1.1–2.1.3 can be mutually combined. Extension to the Marginalized particle filter adds an extra degree of freedom, increasing all previous choices can be combined with additional choices of analytical filters. Having different filter for each possible combination of variants leads to combinatorial explosion.

The task is to find a software scheme that allows to separate these degrees of freedom in a systematic way, reducing combinatorial explosion.

3 Software Analysis of Bayesian Filtering

A range of styles can be used to design a software package for Bayesian filtering. The most general approach is based on detailed analysis of the Bayesian theory which results in a complex set of classes. Some of these classes may be unnecessary for specific purpose, which leads to various simplifications. Toolboxes mentioned in Section 1 are good examples of various levels of simplifications.

The general mathematical formalism of Bayesian calculus is rather simple. All operations are performed on probability density functions (pdfs), denoted $p(a|c)$, where a, c are multivariate random variables. On these data structures are defined the following operations:

marginalization: $p(a|c) = \int p(a, b|c)db$.

chain rule: $p(a, b|c) = p(a|b, c)p(b|c) = p(b|a, c)p(a|c)$.

Bayes rule: $p(a|b, c) = \frac{p(b|a,c)p(a|c)}{p(b|c)}$ where $p(b|c) = \int p(b|a, c)p(a|c)da$.

In principle, all Bayesian algorithms are derived using just these operations. It is tempting to establish a class for pdfs and three operations on them, however, this approach has some pitfalls. The main cause of this is analytical tractability of the integrations and the resulting need for approximations. Software packages thus face the challenge how to represent a range of approximations.

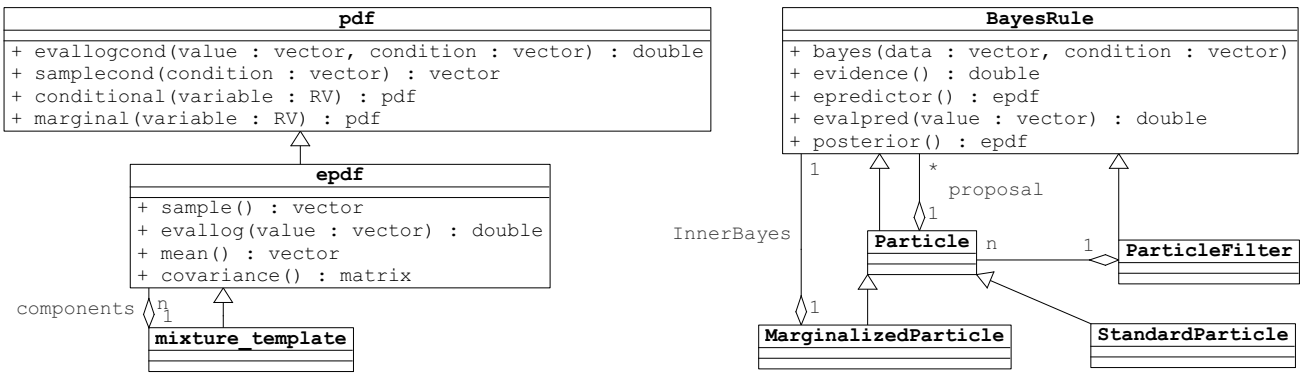

Figure 1: UML-like schematics of the proposed software analysis

3.1 Typical software designs

The central class for all Bayesian object oriented toolboxes is a class "Filter" which correspond to the recursive Bayes rule. A typical implementation is to define Filter as an abstract class with a method which accepts conditional pdfs as arguments and updates posterior pdf. For example, this is implemented by Bayes++, BFL, and similarly in NEF where the pdfs are aggregated into a single structure called model. The resulting pdf is sometimes represented by a class (BFL, NEF) or by its statistics without any further generalization (Bayes++). The remaining Bayesian operations, i.e. marginalization and chain rule are not implemented in any of the mentioned toolboxes.

The advantage of this approach is its close relation to the standard Markov model (1), where each part of the model is clearly distinguished. Such design is also easy to implement when the filter is assumed to use only one approximation, which is typical of standard textbook filters: extended Kalman filter or particle filter. When the problem statement exactly fits these requirements, this approach is sufficient and appropriate. The operations of marginalization and chain rule are not missed since these are trivial for the unconditional Normal and Empirical pdf.

However, this approach is difficult to extend to non-standard techniques, such as forgetting in exponential family, or Marginalized Particle Filters. For example, forgetting in exponential family is based on direct approximation of the Bayes rule without an explicit parameter evolution model [5]. Moreover, some auxiliary variables within the filters may require specialized classes which create a complex hierarchy of inheritance. For example, a proposal for particle filtering using EKF as its local filter (Section 2.1.2) is located in the sixth level of abstraction from the root pdf class in BFL.

3.2 Elementary Classes of Bayesian Calculus

In order to improve expressiveness of a software toolbox, we conjecture that it is necessary to implement more of the Bayesian calculus within the root classes. Here, we describe software analysis that is used in the BDM library. The root classes are implemented as non-pure[1] interfaces, i.e. using only methods without any attributes. A subset of the root classes and their most important methods is displayed in Figure 1.

Class `pdf`: represents conditional pdf, which supports common operations such as sampling and evaluation of its value in a given point. For these operations, it is necessary to know also the value of the conditioning variable. Contrary to other packages, marginalization and conditioning (complement of marginalization within the chain rule) are also defined.
Note that operations `marginalize()` and `condition()` represent only decomposition within a chain rule. Composition of joint density from the decomposed pdfs is provided by offspring called `mprod`.

Class `epdf`: is a specialization of class `pdf` for empty set of conditioning variables. In such a case, it is possible to define more operations, such as evaluation of moments (methods `mean()` and `covariance()`) with numeric output.

Class `RV`: serves as an identifier of random variables and their realizations. The marginalization and conditioning operators require to know which variable in the joint density needs to be marginalized. Methods of class `RV` include creation of unions and intersections.

Class `BayesRule`: implements recursive version of the Bayes rule. The update operation is called `bayes()` and it requires only numerical values in arguments: the vector of observation y_t and the vector of all conditioning variables. This single method implements both the time update and data-update steps (2) and (3). It is assumed that if the knowledge of model (1) is required, it will be represented as attributes within an offspring of this class. This policy gives more control to offspring over the models they accept or require.
The posterior density is provided by an interface

[1] Some of its methods are not virtual.

method `posterior()` returning an offspring of epdf. For more complex filters, the resulting epdf is itself an interface class handling access to internal attributes.
The method `bayes()` has an additional role, which is to evaluate the value of logarithm of the marginal likelihood or evidence (4), which is accesible by method `logevidence()`. Given the importance of this value for model selection it is surprising how many software toolboxes do not evaluate this value.
By design, this class has also enough information to evaluate (4) at $t+1$, i.e. $f(y_{t+1}|u_{t+1})$ (method `predictor()`). A simplified method `epredictor()` is defined when the model has no variables in conditioning or when their numerical value is known. These methods are important for applications in adaptive control, but their relevance in filtering will be shown later.
In contrast to other toolboxes, the class is not named "Filter" since it also covers recursive estimators of stationary parameters.

These classes are specialized into specific families such as: Gaussian and Student `epdfs`, or Kalman filter and Exponential family filters for `BayesRule`. Care is taken that all abstraction layers have also theoretical meaning. An example is the `BayesRule` for exponential family, which is an abstract class that extends the `BayesRule` by methods associated with forgetting (discounting), such as scheduling of the discount factor.

3.3 Particle filtering

In this Section, we argue that wide range of particle filter variants can be build from the basic classes in Section 3.2. First, lets have a look at the proposal function. The difference between the trivial proposal, $q(x_t|x_{t-1}, d_{1:t}) \equiv p(x_t|x_{t-1})$, and some smarter proposals is the addition of the data into the conditioning. Clearly a proposal function is a posterior density resulting from some application of the Bayes rule. Therefore, it seems more suitable to model the proposal by a Bayesian filter (`BayesRule`) rather than a conditional density (`pdf`). The sampling from it is done in two steps: (i) update the posterior, (ii) sample from the posterior. A complication of this approach stem from the fact that the resulting sample is reused as mean of the prior for the proposal in the next step [11]. This step needs to be implemented either by the `BayesRule` of the proposal, yielding an extra layer of abstraction, or by the particle filter, reducing its generality. At the time of writing, the first approach seems to be a slightly better option.

Second suggestion of this Section is to implement each particle as a special case of the `BayesRule`. This implies that function of the the whole filter is separated in two levels, the filter-wide level and the particle level. Potentially, each particle may contain proposal density which forms the third level of `BayesRule` objects. This hierarchy is shown in Figure 2 left in informal way, and Figure 1 in UML notation.

The `ParticleFilter` level has the following role:

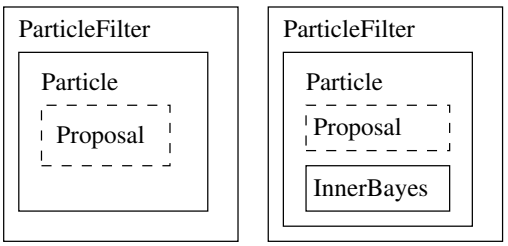

Figure 2: Hierarchy of `BayesRule` levels in the proposed particle filtering scheme.

- Its attributes are an array of `Particles` and an array of their weights
- Method `bayes()` calls `bayes()` of all particles, collect their non-normalized weights via `logevidence()`. This may be done iteratively when variable particle size or adaptive MC strategies are used. Finally, the weights are normalized and resampling takes place.
- All remaining methods (e.g. `posterior()`, `epredictor()`, etc.) are implemented by calling corresponding methods of the `Particles` and weighting them appropriately.

The `Particle` level in the `StandardParticle` implementation has the following role:

- Its prior and posterior density is a Dirac delta function with the value of $x_t^{(i)}$ as its statistics.
- Method `logevidence()` returns contribution of the non-normalized weight in time t, i.e. (9) without $w_{t-1}^{(i)}$.
- Method `epredictor()` returns the observation density with substituted value of $x_t^{(i)}$.

This may seem as an arbitrary construction for a particle filter, but it has the following significant advantages:

- It provides a natural abstraction between particle-specific features, such as the choice of proposal (Section 2.1.2) and filter-specific features, such as the strategy of selection of the number of particles (Section 2.1.1). The only exception from this division is the Auxiliary particle filter which is implemented on the `ParticleFilter` level, due to availability of method `epredictor()` for each particle.
- It unifies marginalized particle filtering with standard particle filtering as the marginalization does not affect the `ParticleFilter` level, but only the `Particle` level.

Note that it is also possible to interpret the standard particle filter as a special case of marginalized particle filter, or the other way around.

3.4 Marginalized Particle Filtering

The particle level in the `MarginalizedParticle` implementation has the following role:

- It contains an extra attribute `InternalBayes` of type `BayesRule` with state variable $x_{1,t}$, and observations $[d_t, x_{2,t}^{(i)}]$, representing (13).
- Prior an posterior density is a product of Dirac function on $x_{2,t}$ and posterior of the internal `InternalBayes`,
- Method `logevidence()` returns `InternalBayes.logevidence()`, minus the likelihood of the proposal.
- Method `epredictor()` returns marginal of the `InternalBayes.epredictor()` on d_t.
- Method `bayes()` may differ for various special cases in Section 2.2.

For example, method `bayes()` for the third—most general—special case of MPF is roughly as follows:

1. Compute predictive density:

$$p(d_t, x_{2,t}|\cdot) = \texttt{InternalBayes.epredictor()};$$

2. Obtain proposal:

$$p(x_{2,t}|\cdot) = p(d_t, x_{2,t}|\cdot).\texttt{marginalize}(x_{2,t}),$$

3. Generate new sample:

$$x_{2,t}^{(i)} = p(x_{2,t}|\cdot).\texttt{sample}();$$

4. Update internal state:

$$\texttt{InternalBayes.bayes}([d_t', x_{2,t}^{(i)\prime}]');$$

5. Return logarithm of evidence:

$$\texttt{InternalBayes.logevidence}() - p(x_{2,t}|\cdot).\texttt{evallog}(x_{2,t}^{(i)});$$

The other special cases from Section 2.2 are simplifications of the above mentioned algorithm. They can be implemented either by specialization of the `MarginalizedParticle`, or by specialization of classes for `InternalBayes`. This decision needs to be again balanced with respect to abstraction level dilemma, Section 1.1. At the time of writing, the former approach seems more advantageous.

4 Discussion

The presented structure of particle filtering implementation may seems to be over-engineered, given that a simple particle filter can be implemented in a few lines of code in scripting environments like Matlab, Python or R. If the task is to implement one variant of the filter, then it is probably the best approach to take. However, with growing number of implemented variants of filters, it soon becomes difficult to maintain. On the contrary, the power of the presented approach is increasing with amount of available code. This will be illustrated on a simple example.

4.1 Illustrative example of use

Consider the following system,

$$\begin{array}{ll} x_t & = g(x_{t-1}) + v_t, \\ y_t & = h(x_t) + w_t, \end{array} \quad \mathrm{Cov}\left(\begin{bmatrix} v_t \\ w_t \end{bmatrix}\right) = \begin{bmatrix} Q & S' \\ S & R \end{bmatrix}, \quad (15)$$

where Q, S, R are unknown parameters. This system is suitable for marginalized particle filtering, since for a given value of x_t, (15) has conjugate prior of the Inverted Wishart type. This estimator is well studied, with predictors and their marginals available in analytical forms of the multivariate Student type. Extensions of this estimator for non-stationary parameters or unknown mean values are available in theory [7], or code (class l of BDM). For detailed treatment of this model with $S = 0$, see [8].

Computation of MPF for system (15) in BDM requires to redefine only one class. The need for redefinition comes from the fact that the ARX class requires residues $x_t - g(x_{t-1})$ and $y_t - h(x_t)$ on the input. Computation of these residues can be implemented either within an offspring of `MarginalizedParticle`, or an offspring of `ARX`. We choose the former, and implement it as a new class `NoiseParticle` which computes the residues and passes them to the `InternalBayes` object, which may be of any class that estimates statistics of residues, e.g. `ARX`. By such a small step we have gained the following flexibility:

- The `ARX` class support estimation of time-varying covariances via forgetting, with various options. Hence, the resulting MPF models estimates not only stationary covariance but also time-varying covariances.
- The `ARX` class has a build-in function for structure estimation, which decides which elements of the covariance are most likely zero. This function is implemented as heuristic Bayesian hypothesis testing.
- The InternalBayes attribute may be changed to be of any (meaningful) type of `BayesRule`. Other meaningful types are exponential family estimators or their approximations, such as Mixtures of exponential family (class `MixEF` in BDM, its recursive form uses online EM, or QB algorithm).
- By plugging the `NoiseParticle` class into different `ParticleFilters`, we obtain standard PF, PF

with variable number of particles, auxiliary PF, etc. The range of options is only limited by the number of available `ParticleFilter` variants which may have been initially implemented for a completely different filter.

Note that it is possible to recognize several un-published versions of MPF within the list above. For all these filters, there is no need to develop and publish any theory – it is already available and implemented.

4.2 Implementation in other toolboxes

The previous Section illustrated the power of the proposed approach in the BDM library. However, it can be easily implemented in all object-oriented toolboxes, such as those in Section 3.1. The key operation that are required—and currently missing in these toolboxes—are operations `logevidence()` and `epredictor()` on the root Bayes filter class, and operation `marginalize()` on the class representing pdfs.

5 Conclusions

The presented software analysis reveals a that a particle filter can be implemented as an interaction of three (or more) layers of Bayesian filters, Section 3. Each of these layers represent one degree of freedom corresponding to (i) standard vs. auxiliary vs. adaptive MC particle filtering, (ii) particle vs. marginalized particle filter, (iii) proposal density filters and internal marginal filters. These degrees of freedom can be arbitrarily combined, giving rise to combinations that has not been published yet. As shown is Section 4.1, new filters arise just by combining existing methods in software without any need for new theory.

The approach is presented in general form to illustrate the main idea, which can be implemented differently in existing toolboxes. Reference implementation of the approach is available within the BDM library `http://mys.utia.cas.cz:1800/trac/bdm/`, with extra layers of abstraction that makes implementation in C++ more convenient. The library is available as an open-source project, with interfaces to Matlab and Python.

Acknowledgment

Support of grants MŠMT 1M0572 and GAČR 102/08/P250 is gratefully acknowledged.

References

[1] G. Booch, R. Maksimchuk, M. Engle, B. Young, J. Conallen, and K. Houston. Object-oriented analysis and design with applications. 2007.

[2] J. Cornebise, E. Moulines, and J. Olsson. Adaptive methods for sequential importance sampling with application to state space models. *Statistics and Computing*, 18(4):461–480, 2008.

[3] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[4] D. Fox. Adapting the sample size in particle filters through KLD-sampling. *The international Journal of robotics research*, 22(12):985, 2003.

[5] R. Kulhavý and M. B. Zarrop. On a general concept of forgetting. *International Journal of Control*, 58(4):905–924, 1993.

[6] Z. Peroutka. Design considerations for sensorless control of PMSM drive based on extended Kalman filter. In *11th European Conference on Power Electronics and Applications (EPE)*, Dresden, Germany, 2005.

[7] V. Peterka. Bayesian approach to system identification. In P. Eykhoff, editor, *Trends and Progress in System identification*, pages 239–304. Pergamon Press, Oxford, 1981.

[8] S. Saha, E. Özkan, V. Šmídl, and F. Gustafsson. Marginalized particle filters for Bayesian estimation of noise parameters. In *Proceedings of the 13th International Conference on Information Fusion*, Edinburgh, UK, 2010.

[9] T. Schön, F. Gustafsson, and P.-J. Nordlund. Marginalized particle filters for mixed linear/nonlinear state-space models. *IEEE Transactions on Signal Processing*, 53:2279–2289, 2005.

[10] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. MIT press, Cambridge, Massachusetts, USA, 2005.

[11] R. Van der Merwe, A. Doucet, N. De Freitas, and E. Wan. The unscented particle filter. *Advances in Neural Information Processing Systems*, pages 584–590, 2001.

[12] M. Šimandl and O. Straka. Sampling densities of particle filter: a survey and comparison. In *Proceedings of the 26th American Control Conference (ACC)*, New York City, USA, 2007.

[13] V. Šmídl and A. Quinn. Variational Bayesian filtering. *IEEE Transactions on Signal Processing*, 56(10):5020–5030, 2008.