Input-adaptive QMC-Kalman filters for track fitting

Rodolfo G. Esteves and Michael D. McCool
School of Computer Science
University of Waterloo
{rgesteve, mmccool}@cs.uwaterloo.ca

Christiane Lemieux
Department of Statistics and Actuarial Science
University of Waterloo
clemieux@math.uwaterloo.ca

***Abstract*—On-line track reconstruction is one of the bottlenecks of the pattern recognition task in High-Energy Physics (HEP). This problem has been traditionally divided into the sub-tasks of track finding and track fitting. The latter involves estimating the state of a particle inside a detector moving under the influence of a magnetic field. For the last twenty or so years the most popular solution to the track fitting problem has been the Kalman filter (KF). It is well known that the KF is only guaranteed to compute the optimal estimator if the dynamics of the system are linear and subject to Gaussian noise. However, these conditions are not met in the track fitting problem, in particular, the dynamics are strongly non-Gaussian due to effects such as multiple Coulomb scattering and energy loss. A proposed solution is the "Gaussian sum filter" (GSF) which runs a bank of KFs to estimate each of the modes of the noise distributions, modelled here as a mixture of Gaussians. But this solution is limited by the fact that the GSF uses the same distribution for every input dataset. To address this issue, we present in this paper the Input-adaptive KF (IAKF), which makes use of the dynamic code generation features in Intel's ArBB parallel framework to create a GSF that matches the given (observation) noise distribution. The IAKF further deals with non-linearity by having the GSF drive, instead of KFs, the recently proposed Quasi-Monte Carlo KF (QMC-KF), a generalization of the σ-point KF family. Numerical results are shown to validate the performance of the IAKF. The generated code is not only tailored to the data, but takes advantage of several levels of parallelism in multi-core processors.**

***Keywords*-Nonlinear dynamic systems; quasi-Monte Carlo (QMC); Kalman filter (KF)**

I. INTRODUCTION

High-Energy Physics (HEP) studies the fundamental components of matter and radiation, as well as their interactions, thereby addressing questions crucial for the understanding of the Universe. HEP experiments pose unique challenges in design, implementation and data analysis. For one, HEP experiments often have to sort through the massive data streams produced by particle accelerators. Particle accelerators use electromagnetic fields to induce high-momenta on sub-atomic particles, whose collisions among each other generate data that is invaluable to understand matter under extreme conditions. This data often takes the form of *traces*, measurements of various physical aspects of the particle taken along their collision paths at specific detection points (the "stations"). A considerable portion of the data mining that takes place in a HEP experiment is spent in the *track reconstruction* task, which consists of taking the traces and reconstructing the underlying physical process. Track reconstruction is often broken up in the complementary sub-tasks of *track finding* and *track fitting*. Track finding involves associating a set of readings with the likely trajectory of a specific particle. Track fitting then takes those sets and determines the values that best conciliate the experimental readings and the mathematical description of the trajectory. These values, or *state* are usually denoted by $\mathbf{x}_k \in \mathbb{R}^5$, and consist of the exact intersection point of the particle with each of the detectors, the track direction and the curvature. A *measurement* $\mathbf{z}_k$ is taken at each station k, and a collection of stations represents a model of the whole detector.

The mathematical setting of this problem is to consider the particle accelerator as a nonlinear dynamic system [1]:

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k) + \omega_k \tag{1}$$

of which the measurement $\mathbf{z}_k$ is a function, also polluted by noise to form the stochastic "observation" process:

$$\mathbf{z}_k = h(\mathbf{x}_k) + \nu_k \tag{2}$$

It can be seen from the notation that the system is considered as discretized in time with indices k. In the transition between measurement points $k-1$ and k, the state is considered to be subject to *process noise*, which is denoted here by ω_k. Moreover, precise observations are not possible because of the limitations in the measuring model and instruments, a circumstance which is considered by introducing the *measurement noise* ν_k. The precise probability distributions of ω_k and ν_k heavily depend on the application, but they are commonly taken to be mutually independent.

The goal of track fitting is then to *filter* out the noise, i.e., to calculate the posterior distribution $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ at all measurement indices k, where $\mathbf{z}_{1:k}$ denotes a sequence of observations $\{\mathbf{z}_i\}_{i=1}^{k}$. The optimal solution to this problem is given by the recursive Bayesian estimation algorithm [2], which recursively updates the posterior density of the system state as new observations arrive. However, it is worth noting that this recursive solution is only tractable for linear, Gaussian systems, in which case the closed-form recursive solution to the Bayesian integral equations is the well-known Kalman filter (KF). Because in HEP experiments process noise arises from interactions between charged particles and

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-156-4

detector materials, the linearity and Gaussianity assumptions the KF relies on are hardly ever met. The measurement noise v_k is also rarely Gaussian, since in real detectors there is an ever-present possibility of outlying or ambiguous observations. Therefore, approximate solutions need to be applied to this problem.

Approximate filtering for general dynamic state space systems can be roughly categorized in two approaches: deterministic and Sequential Monte Carlo-based. The first approach, which is the focus of this paper, generalizes the KF, trying to keep its simplicity and well-understood theory [1], [2], [3]. However, these generalizations suffer from the fact that filters must be manually constructed from the input data, or that the same filter is applied to every input dataset, which degrades the quality of the estimator or require inordinate user effort. To overcome these disadvantages, we present the Input-adaptive KF (IAKF), an approximate solution to the filtering problem based on the KF where an automatically constructed Gaussian mixture models the measurement noise and a set of non-linear KFs are used for the actual filtering. We note that some ideas preliminary to this paper have been presented in [4], but a paper-length exposition has never been published.

The organization of this paper is as follows: Section II introduces the variants of the KF designed to handle non-linear dynamics in the presence of non-Gaussian noise. Our proposed method, the IAKF is presented in Section III, where we describe its operation and demonstrate its statistical performance. The most novel feature of the IAKF is that it is implemented by relying heavily on dynamic code generation, an aspect we explore in section IV. After a brief introduction to this technology, we present the dynamic code generation facilities of ArBB, and how we use them to implement the IAKF. We also compare previous approaches to code generation for filtering with our work. Finally, we conclude in Section V.

II. NON-LINEAR GENERALIZATIONS TO THE KF

The KF is a prediction/correction scheme with the predicted state and observation being calculated from the estimate at the previous measurement point (or from the prior distribution $p(x_0)$):

$$\begin{aligned}\mathbf{x}_{k|k-1} &= f(\mathbf{x}_{k-1|k-1}, \omega_k)\\ \mathbf{z}_{k|k-1} &= h(\mathbf{x}_{k|k-1}, v_k)\end{aligned}$$

where ω_k and v_k are independent and normally-distributed, with zero mean and covariances Q and R respectively. f is a stochastic transition kernel from state $\mathbf{x}_{k-1}$ to state $\mathbf{x}_k$ and h is a stochastic non-linear mapping from the current state to the observation. $\mathbf{x}_{k-1|k-1}$ is the state estimate given the observations $\mathbf{z}_{1:k-1}$, and $\mathbf{x}_{k|k-1}$ is the *predicted* estimate for the next state given the same trace.

These predictions are then updated with the *innovation*, the difference between predicted and actually observed measurement, modulated with a correcting factor K_k, the *Kalman gain*, to render the next state estimate $\mathbf{x}_{k|k}$ and corresponding covariance $\mathbf{P}_{k|k}$:

$$\begin{aligned}\mathbf{x}_{k|k} &= \mathbf{x}_{k|k-1} - K_k(\mathbf{z}_k - \mathbf{z}_{k|k-1})\\ \mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - K_k(P_z)_{k-1}K_k^T\end{aligned}$$

The KF is *optimal* (in the least-squares sense) provided process and measurement mappings are linear, and the associated noise is Gaussian. However, as mentioned in Section I, this is not the case in HEP experiments. Below, we present two general approaches dealing with non-linearities and non-Gaussianity, particularly relevant to track fitting.

A. Dealing with non-linearities

When considering non-linearities, a common simplifying assumption is that

$$\begin{aligned}p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}) &= \mathcal{N}(\mathbf{x}_{k-1|k-1}, \mathbf{P}_{k-1|k-1})\\ p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) &= \mathcal{N}(\mathbf{x}_{k|k-1}, \mathbf{P}_{k|k-1})\end{aligned}$$

which makes the filtering distribution $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ normally distributed as well.

The Extended Kalman filter (EKF) is by far the most popular technique for state estimation on non-linear dynamic systems. Despite its advantages, the EKF comes at the cost of considerable shortcomings, most of which result from the EKF's linearization of process and measurement equations around the previous estimate. This does not take into account the statistical properties of the noise, and may ultimately cause the divergence of the filter [1].

Alternative approaches go back to the definition of first- and second-moment of the filtering distribution:

$$\begin{aligned}\mathbf{x}_{k|k-1} &= \int \mathbf{x}_k p(\mathbf{x}_k|\mathbf{z}_{1:k-1})\mathrm{d}\mathbf{x}_k\\ &= \int f(\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})\mathrm{d}\mathbf{x}_{k-1} \qquad (3)\end{aligned}$$

The second moment follows the same pattern and is ommitted here due to space restrictions.

In this paper we present the Input-adaptive KF, which makes use of the QMC-KF algorithm to construct filters that are tailored to the input data, thereby improving the robustness of a track-fitting system as shown in Section III-B. The QMC-KF numerically approximates (3) using Monte Carlo or quasi-Monte Carlo (QMC) integration. QMC-KF [2], a generalization of σ-point filters, relies on the approximation

$$\mathbf{x}_{k|k-1} = \sum_{i=1}^{n} f(\mathbf{x}_k^{(i)}) \qquad (4)$$

where $[\mathbf{x}_k^{(i)}]_{i=1}^n$ is a low-discrepancy point set of the appropriate dimensionality (in this work, as in [2], we use randomized Halton point sets) under some transformation that maps to a Gaussian distribution. This can be done in several ways, and we explain our choices in Section III-B.

Copyright (c) IARIA, 2011.

B. Dealing with non-Gaussianity

Gaussian distributions are hardly appropriate for the phenomena studied in HEP experiments. Measurements include outliers and ambiguity that introduce tails in the measurement error distribution v_k, whereas the biggest contributors to process noise, energy loss and multiple scattering, are highly non-Gaussian. Forcing a Gaussian distribution to describe these effects greatly reduces the amount of information contained in the true densities, especially in the case of multimodal densities.

A common approach to avoid information loss while remaining within the KF framework consists of modelling the non-Gaussian distributions by *Gaussian mixtures*. For example, measurement outliers can be handled by a Gaussian mixture with a "core" component describing the "regular" measurements, and one or more components describing the outlier-induced tails [5] (e.g., by a mixture of Gaussians sharing the mean but with different covariances). Likewise, ambiguous measurements can be modeled by a mixture of Gaussians with one component per possible value, i.e., with the mean set to the possible value and identical variances, thus concurrently using all possible meanings of the ambiguous measurement. As for process noise, we can model the tails of the multiple scattering for low-energy particles, or the highly asymmetric energy loss of electrons by suitable Gaussian sums [6]. In principle, every distribution involved in the filtering process (state priors, measurement and process noise) can be modeled as a Gaussian mixture. Taking this notion as a guideline, Alspach and Sorenson [3] proposed the Gaussian-sum filter (GSF), where every component of the mixture is propagated and updated by a standard KF. This is to say that the GSF consists of a bank of Kalman filters running in parallel.

III. The Input-adaptive Kalman filter

All the techniques described in the above section assume that the same filter will be applied to every input, since parameters chosen for the filter are fixed at program construction time. This does not reflect the actual system under consideration, and may therefore result in inaccurate or divergent filters. Here we address this issue by dynamically building the filter based on the input data. In this section we present the Input-adaptive KF (IAKF), a GSF driving QMC-KFs with a preprocessing step of data clustering.

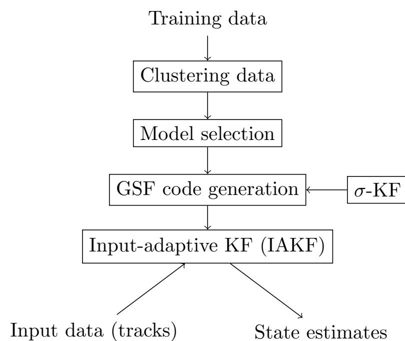


Figure 1. Conceptual diagram of the IAKF

A. Operation of the IAKF

Figure 1. shows the conceptual diagram of the IAKF, whose general operation is as follows: a sample of the tracks at the first station is read as a training set, assumed to have been generated by a Gaussian mixture density

$$p(x_0) = \sum_{i=1}^{l} \alpha_i \mathcal{N}(\mu_i, C_i) \tag{5}$$

We fit the training data into the Gaussian mixture, which models the notion that the measurement noise is multimodal due to outliers. It is premature to decide *a priori* how many terms l the mixture is to have because of the lack of information on the outlier distribution. Therefore we run a parameter-estimation task on several candidate component counts and use a model selection criterion to choose the best fit among them. The parameter estimation procedure determines the values for mixture parameters and proportions. Having determined the appropriate number of components, we generate the corresponding QMC-KF instances, which will be "baked in" the final filter. Finally that resulting filter is run on the input data to perform the actual state estimation. It is worth noting that the mixing proportion of the Gaussian sum needs to be reweighted at every iteration to maintain an accurate estimate.

The model selection criteria we use to determine how many components the Gaussian mixture should have is the Bayesian information criterion (BIC). For parameter estimation, we have chosen the Expectation Maximization (EM) algorithm, a commonly used maximum-likelihood technique to estimate the parameters of a distribution of a specified form from training data. As the computational load of the GSF is directly related to the number of components in the mixture, we run EM for a low component count (2 to 5).

In the GSF, care must be taken to control the possible combinatorial explosion of Gaussian terms in the posteriors. As the IAKF has a fixed number of components, when any of the constituent QMC-KF estimates becomes multimodal, the smaller components are dropped as to maintain a constant component count.

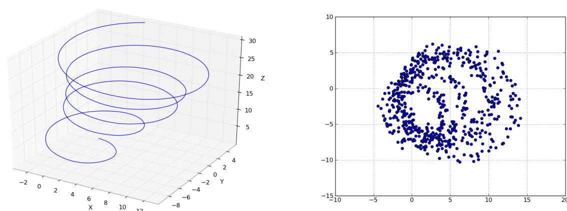
B. Numerical evaluation

To validate the computational performance of the IAKF implementation, we test our system on simulated data. We use the methodology proposed in [7] to simulate the transit of a charged particle in a magnetic field. Therefore, the function f takes the form of a 4th-order Runge-Kutta solution (with fixed-step size) to the equation

$$d\mathbf{p} = \kappa q(\mathbf{v} \times \mathbf{B}) ds / |\mathbf{v}|$$

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-156-4

where $\mathbf{p}$ is the momentum of the particle, q its charge, $\mathbf{v}$ its velocity, $dt = ds/|\mathbf{v}|$ the trajectory length, $\mathbf{B}$ the magnetic field, and κ is a constant. The state is the 5-tuple $(x, y, t_x, t_y, q/|\mathbf{p}|)$ where $(x,y)_z$ denote the intersection of the trajectory with detecting surface z, $t_x(z) = dx/dz$ and $t_y(z) = dy/dz$ indicate the particle's direction at that point. The observation function h mimics the way silicon micro-strip detectors carry out measurements, projecting the x,y-coordinates at the intersections with the stations z. An illustration of a simplified version of the above system is shown in Figure 2. The simplifications consist of using an homogeneous magnetic field and a (uni-modal) Gaussian noise for both system (a) and measurement (b) noise.



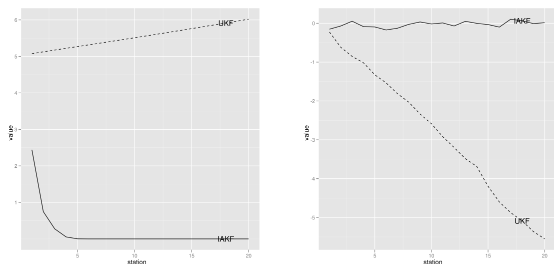
(a) Dynamics process f (motion of charged particle in a magnetic field)

(b) Measurement process h

Figure 2. Non-linear functions f and h for track fitting

The actual experimental setup consists of adding 2-, 3- and 4-component Gaussians mixture additive noise to the measurement process on a 7-station detector. The dynamics of the system are encoded in the Runge-Kutta solver described above.

We test three filters: the Unscented Kalman filter (UKF) and the IAKF driving both QMC-KFs and traditional Monte Carlo filters (i.e., a non-linear filter making use of Monte Carlo integration of the Kalman filter recurrences). Our QMC-KF filters are fed by a 1024-randomized Halton sequence of state dimension 5. The particular randomization method is to use an Owen-type scrambling [8].



(a) Covariance trace

(b) State estimation residuals

Figure 3. Performance of 2-component IAKF (solid) vs UKF (dashed)

Figure 3 illustrates the relative filter efficiency of the UKF vs the 2-component IAKF/QMC on simulated runs. It averages the RMSE (i.e., the trace of the covariance matrix $\mathbf{P}_k$) on 1024 tracks on 20 stations subject to a 2-component Gaussian mixture "outlier" measurement noise (i.e., two components with the same mean but different covariance). It is shown that the IAKF (solid line) presents a significantly smaller and steadier filtering error than the UKF (dashed line), thus providing more accurate and stable state estimation.

IV. Run-time code generation of the IAKF

When developing a program, there is a constant tension between information and time: often, decisions about a program structure have to be made and implemented well before any input is actually seen; when input data becomes available to the program, it is too late for the characteristics of that data to shape the program's execution in a deep level. A system that could interpret the specific input and *generate* code specifically tailored to that input and the host environment would see improved performance. Such a system clearly has the need to programmatically manipulate program code. This is made easier by a simple, regular syntax, so systems based on Lisp or its variants are highly favoured in this type of application. A common solution to allow the creation and execution of code at run-time in languages with richer syntax is to provide a special function (usually called `eval`) that takes a string containing the source code to be executed, and executes it immediately. Incrementally building strings that constitute correct and complete programs is difficult enough, but having to deal with dependencies, variables and other bookkeeping that allows communication between host and created program constitutes a cost that far outweights the possible benefits.

A more tractable solution has been found to use a two-stage approach, where the code to be generated is restricted to a small domain-specific language (DSL). Instead of feeding text written in this language to an `eval`-like function, a full language processing infrastructure processes the DSL source and somehow injects it to the main code, which is written in a general-purpose language. The best-known example of this form of DSL is the pair `lex`/`yacc` for lexer/parser generation, which generates code from the DSL at compile-time. The recent proliferation of just-in-time (JIT) compilers has made this two-stage approach possible at run-time, but the maintenance of two separate source translation infrastructures is still cumbersome. Recent work addresses this problem by using an "embedded" variation of DSL, which uses some facilities of a general-purpose language, to *host* a more specific language. Examples of this technique can be found in a number of modern languages, like the Spirit [9] parser generator C++ library, the LINQ language-integrated data-query framework for Microsoft's .NET, and most famously, numerous examples in the Ruby language, like Ruby on Rails. Embedded domain-specific languages (EDSLs) and JIT compilers are a good fit, as it is

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-156-4

much easier to develop a compiler for a simple language than to take on the task of developing an incremental compiler for a full language.

A central issue when developing EDSLs is the choice of data structure to represent programs or program fragments. As stated above, the most natural to use is a string containing the source of the program to be executed at the next stage. This representation, however, is rife with problems, as the contents of a string are effectively out of the control of the programming language. An alternative is the macro-like approach incorporated in the Intel Array Building Blocks (ArBB) library, which is used in this paper and will be described next.

A. Code generation in ArBB

ArBB, formerly Ct, library [10] is a retargetable dynamic compilation framework whose main purpose is to facilitate the coding of programs that are to take advantage of modern multi- and many-core architectures. As such, it provides a set of implicitly data-parallel collection data structures and computational patterns (including *map*, *reduce* and *prefix sum*). A programmer can make use of this abstract notation and target several levels of parallelism present in multi-core homogeneous systems, as well as potentially heterogeneous accelerator-based many-core architectures (for example, Graphics Processing Units) and cluster-based systems.

A more complete description of ArBB is best found elsewhere [11]. Here, we focus on the code-generating aspect. ArBB is a two-stage system, where the programmer works within the usual confines of the C++ language (the *uncaptured* environment), but specifies the functions that are to be run in parallel using library-provided data types, functions and control flow (the *captured* environment). The rationale for this two-level architecture is that captured code can be compiled to make maximum use of the system it is running on: vectorized ALUs, multiple processors, accelerators and other possible system configurations.

ArBB provides mirrors of C++ control structures in the form of the ‘keywords’ `_if`, `_for`, `_while` and `_do`. This inclusion allows the programmer to express the effect of the computation serially, even though this computation will ultimately be performed in parallel. For example, the kernel:

```
void nr_sqrt_kernel(arbb::f32 a, arbb::f32& s)
{
    _if (a < 0) { return -1; } _end_if;
    arbb::f32 sprev = arbb::f32(0);
    s = a;

    _while (arbb::abs(xprev -x) > tol) {
        sprev = s
        s = 0.5 * (sprev + a / sprev)
    } _end_while;
}
```

can be the argument of a `map` operator to find the square root of each (positive) element in an ArBB array. Each invocation of `nr_sqrt_kernel` will execute concurrently.

An interesting side effect of this two-level execution approach can be seen when C++ control flow is used in *captured* mode. The following code:

```
unsigned int unroll_factor = 4;

void fill_vec_kernel(arbb::dense<arbb::f32>& v,
    arbb::f32 val) {
 for(size_t i=0; i < unroll_factor; i++)
    { v[i]=val; }
}
```

has the same effect as unrolling the loop:

```
void fill_vec_kernel(arbb::dense<arbb::f32>& v,
    arbb::f32 val)
{
 v[0] = val; v[1] = val; v[2] = val; v[3] = val;
}
```

In what follows, we will use this sort of template-based generation to produce QMC-KFs customised to the input. It is pertinent to note that, while *C++ templates* are one form of code generation and programmatic manipulation (known as *metaprogramming*), they are an altogether different mechanism from the one described above for ArBB. C++ template-based metaprogramming is a *compile-time* construct that can be used *in addition* to the run-time facilities we have explained.

B. The ArBB implementation of the IAKF

We have built an ArBB implementation of the IAKF, which determines the number of components `NUM_KF` by the procedure described in Section III and uses this number to generate as many QMC-GSF as necessary, in a manner sketched by:

```
for k = 0; k < NUM_KF; k++) {
  _for( i = N - 1, i >= 0, i-- ){
    // ... magnetic field setup ...
    filter(ts,ss,xInfo, ts.hitsX2.row(i), w,T,C);
    filter(ts,ss,yInfo, ts.hitsY2.row(i), w,T,C);
    for( int j = 0; j < 3; j++ ){
      H2[j] = H1[j];
      H1[j] = H0[j];
    }
    z2 = z1; z1 = z0;
  } _end_for;
}
```

The `filter` kernels are wrapper functions over `maps`, which result in a parallelization strategy both over tracks and over concurrent QMC-KFs. Below we present the results of some numerical experiments. As an experiment we ran the filter with 1- and 2-component mixtures to assess the computational load.

Figure 4 illustrates a benchmark of our system, running on a dual-Core Intel i3 at 2.27 GHz. It can be seen from that the running time is roughly proportional to the number of components. Beyond that, it is more interesting to note that the program is *scalable*, i.e., that the running time consistently decreases as the number of cores increases. This is greatly desirable, as core count is only likely to increase given the current hardware trends.

C. Other approaches to KF code generation

The automatic generation of data analysis programs has been explored for at least a decade. For example, the AutoBayes program synthesis system [12] generates C++ code from a declarative specification of the statistical model via deductive synthesis directed by code templates. As such, it most resembles the `lex`/`yacc`-approach of DSLs described above, where a separate programming language infrastructure is required.

Another example of a system that synthesizes KF and variants code is AutoFILTER [13], which similarly to AutoBayes, outputs C++ code from a textual specification specialized to the description of noise distributions and differential equations. An interesting variation that AutoFILTER includes is that it links against the libraries from the Octave linear algebra system, within which its output is supposed to be used. This illustrates another aspect of code generation within C++, its interoperability.

In contrast to the above-mentioned systems, our work only requires a standard C++ compiler. Furthermore, our generated code takes full advantage of the parallel features of the architecture the program is running on. On the other hand, significant work has gone into the automated verification and certification of the code generated by AutoBayes, an aspect of considerable importance, which is not covered in this paper. The interoperability aspect of AutoFILTER is also a straightforward addition to our system.

V. Conclusion

In this work we proposed the Input-adaptive Kalman filter (IAKF), a member of the deterministic, approximate, non-linear filter family. In contrast with traditional methods, the IAKF adapts to the input, running as many filters as necessary to best fit the input data. This feature, validated by numerical results, makes its estimates more accurate. Furthermore, the IAKF is more robust to changes in data than its non-adaptive counterparts. To implement the IAKF, we make use of the run-time code generation and compilation afforded us by modern parallelism libraries. It is our contention that furnishing the end-programmer with the ability to tailor the program in data-driven programs, as inferencing systems must be, allows for simple and straightforward implementation of programs that deal better with realistic scenarios. Moreover, the improved running time that modern parallel hardware offers can be put to good use in more realistic models and greater variety of data.

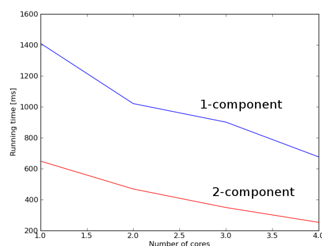


Figure 4. Running time of the QSF with 1- (in blue) and 2-component (in red) MoG v_k

References

[1] R. van der Merwe and E. Wan, "Sigma-point Kalman filters for probabilistic inference in dynamic state-space models," in *Workshop on Advances in Machine Learning*, 2003.

[2] D. Guo and X. Wang, "Quasi-Monte Carlo filtering in nonlinear dynamic systems," *IEEE Transactions on signal processing*, vol. 54, 2006.

[3] D. L. Alspach and H. W. Sorenson, "Nonlinear Bayesian estimation using Gaussian sum approximations," *IEEE Transactions on automatic control*, vol. 17, no. 4, 1972.

[4] R. G. Esteves, C. Lemieux, and M. D. McCool, "Run-time generation of qmc-kalman filters for track fitting (abstract)," in *Monte Carlo and Quasi-Monte Carlo methods*, 2010.

[5] R. Frühwirth, "Track fitting with long-tailed noise: a Bayesian approach," *Computer Physics Communications*, vol. 85, 1995.

[6] W. Adam, R. Frühwirth, A. Strandlie, and T. Todorov, "Reconstruction of electron tracks with the Gaussian-sum filter," CERN, Tech. Rep. CERN-CMS-RN-2003-001, 2003.

[7] S. Gorbunov and I. Kisel, "An analytic formula for track extrapolation in an inhomogeneous magnetic field," in *International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, 2005.

[8] C. Lemieux, *Monte Carlo and quasi-Monte Carlo Sampling*. Springer, 2009.

[9] J. de Guzman and H. Kaiser, "Boost.Spirit homepage," http://boost-spirit.com, 2010.

[10] "Intel Array Building Blocks homepage," http://software.intel.com/en-us/articles/intel-array-building-blocks, 2011.

[11] M. McCool, "Intel Array Building Blocks: A retargetable, dynamic compiler and embedded language," in *Code Generation and Optimization*, 2011.

[12] B. Fischer, J. Schumann, and T. Pressburger, "Generating data analysis programs from statistical models," in *Workshop on Semantics, Applications, and Implementation of Program Generation*, 2000.

[13] J. Richardson and E. Wilson, "Flexible generation of Kalman filter code," in *IEEE Aerospace Conference*, 2006.

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-156-4