Quantitative Verification of Numerical Stability for Kalman Filters

Alexandros Evangelidis and David Parker

School of Computer Science, University of Birmingham, UK
{a.evangelidis, d.a.parker}@cs.bham.ac.uk

Abstract. Kalman filters are widely used for estimating the state of a system based on noisy or inaccurate sensor readings, for example in the control and navigation of vehicles or robots. However, numerical instability may lead to divergence of the filter, and establishing robustness against such issues can be challenging. We propose novel formal verification techniques and software to perform a rigorous quantitative analysis of the effectiveness of Kalman filters. We present a general framework for modelling Kalman filter implementations operating on linear discrete-time stochastic systems, and techniques to systematically construct a Markov model of the filter's operation using truncation and discretisation of the stochastic noise model. Numerical stability properties are then verified using probabilistic model checking. We evaluate the scalability and accuracy of our approach on two distinct probabilistic kinematic models and several implementations of Kalman filters.

1 Introduction

Estimating the state of a continuously changing system based on uncertain information about its dynamics is a crucial task in many application domains ranging from control systems to econometrics. One of the most popular algorithms for tackling this problem is the *Kalman filter* [16], which essentially computes an optimal state estimate of a noisy linear discrete-time system, under certain assumptions, with the optimality criterion being defined as the minimisation of the mean squared estimation error.

However, despite the robust mathematical foundations underpinning the Kalman filter, developing an operational filter in practice is considered a very hard task since it requires a significant amount of engineering expertise [20]. This is because the underlying theory makes assumptions which are not necessarily met in practice, such as there being precise knowledge of the system and the noise models, and that infinite precision arithmetic is used [12,24]. Avoidance of numerical problems, such as round-off errors, remains a prominent issue in filter implementations [11,12,24,26]. Our goal in this paper is to develop techniques that allow the detection of possible failures in filters due to numerical instability arising as a result of these assumptions.

The Kalman filter repeatedly performs two steps. The first occurs before the next measurements are available and relies on prior information. This is called

the *time update* (or prediction step) and propagates the "current" state estimate forward in time, along with the uncertainty associated with it. These variables are defined as the a priori state estimate $\hat{x}^-$ and estimation-error covariance matrix P^-, respectively. The second step is called the *measurement update* (or correction step) and occurs when the next state measurements are available. The Kalman filter then uses the newly obtained information to update the a priori $\hat{x}^-$ and P^- to their a posteriori counterparts, denoted $\hat{x}^+$ and P^+, which are adjusted using the so-called optimal *Kalman gain* matrix K.

The part of the filter that could hinder its numerical stability, and so cause it to produce erroneous results, is the propagation of the estimation-error covariance matrix P in the time and measurement updates [4,12,20]. This is because the computation of the Kalman gain depends upon the correct computation of P and round-off or computational errors could accumulate in its computation, causing the filter either to diverge or slow its convergence [12]. While, from a mathematical point of view, the estimation-error covariance matrix P should maintain certain properties such as its symmetry and positive semidefiniteness to be considered valid, subtle numerical problems can destroy those properties resulting in a covariance matrix which is theoretically impossible [17]. Out of the two update steps in which the filter operates, the covariance update in the correction step is considered to be the *"most troublesome"* [20]. In fact, the covariance update can be expressed with three different but algebraically equivalent forms, and all of them can result in numerical problems [4].

To address the aforementioned challenges, we present a general framework for modelling and verifying different filter implementations operating on linear discrete-time stochastic systems. It consists of a modelling abstraction which maps the system model whose state is to be estimated and a filter implementation to a discrete-time Markov chain (DTMC). This framework is general enough to handle the creation of various different filter variants. The filter implementation to be verified is specified in a mainstream programming language (we use Java) since it needs access to linear algebra data types and operations.

Once the DTMC has been constructed, we verify numerical stability properties of the Kalman filter being modelled using properties expressed in a reward-based extension [10] of the temporal logic PCTL (probabilistic computation tree logic) [13]. This requires generation of non-trivial reward structures for the DTMC computed using linear algebra computations on the matrices and vectors used in the execution of the Kalman filter implementation. The latter is of more general interest in terms of the applicability of our approach to analyse complex numerical properties via probabilistic model checking.

We have implemented this framework within a software tool called VerFilter, built on top of the probabilistic model checker PRISM [18]. The tool takes the filter implementation, a description of the system model being estimated and several extra parameters: the maximum time the model will run, the number of intervals the noise distribution will be truncated into, and the numerical precision, in terms of the number of decimal places, to which the floating-point numbers which are used throughout the model will be rounded.

The decision to let the user specify these parameters is particularly important in the modelling and verification of stochastic linear dynamical systems, where the states of the model, which comprise of floating-point numbers, as well as the labelling of the states, are the result of complex numerical linear algebra operations. Lowering the numerical precision usually means faster execution times at the possible cost of affecting the accuracy of the verification result. This decision is further motivated by the fact that many filter implementations run on embedded systems with stringent computational requirements [24], and being able to produce performance guarantees is crucial.

We demonstrate the applicability of our approach by verifying two distinct filter implementations: the conventional Kalman filter and the Carlson-Schmidt square-root filter. This allows us to evaluate the trade-offs of one versus the other. In fact, our tool has been tested on five implementations, but we restrict our attention to these two due to space restrictions. For the system models, we use *kinematic state models*, since they are used extensively in the areas of navigation and tracking [4,19]. We evaluate our approach with two distinct models. We demonstrate that our approach can successfully analyse a range of useful properties relating to the numerical stability of Kalman filters, and we evaluate the scalability and accuracy of the techniques.

Related Work. Studies of Kalman filter numerical stability outside of formal verification are discussed above and in more detail in the next section. To the best of our knowledge, there is no prior work applying probabilistic model checking to the verification of Kalman filters. Perhaps the closest is the use of non-probabilistic model checking on a variant of the filter algorithm is the work by [21], which applied model checking to target estimation algorithms in the context of antimissile interception. In general, applying formal methods in state estimation programs is an issue which has concerned researchers over the years. For example, [23,25] combined program synthesis with property verification in order to automate the generation of Kalman filter code based on a given specification, along with proofs about specific properties in the code. Other work relevant to the above includes [22], which used the language ACL2 to verify the loop invariant of a specific instance of the Kalman filter algorithm.

2 Preliminaries

2.1 The Kalman filter

The Kalman filter tracks the state of a linear stochastic discrete-time system of the following form:

$$x_{k+1} = F_k x_k + w_k \qquad z_k = H_k x_k + v_k \tag{1}$$

where x_k is the $(n \times 1)$ system state vector at discrete time instant k, F_k is a square $(n \times n)$ state transition matrix, which relates the system state vector x_k between successive time steps in the absence of noise. In addition, z_k is the $(m \times 1)$

measurement vector, H_k is the $(m \times n)$ measurement matrix, which relates the measurement with the state vector. Finally, w_k and v_k represent the process and measurement noises, with covariance matrices Q_k and R_k, respectively. Given the above system and under certain assumptions, the Kalman filter is an optimal estimator in terms of minimising the mean squared estimation error.

The task of the Kalman filter is to find the optimal Kalman gain matrix K_k in terms of minimising the sum of estimation-error variances, which can be obtained by summing the elements of the main diagonal of the a posteriori estimation-error covariance matrix P^+. The estimation process begins by initialising $\hat{x}_0^+ = \mathbb{E}[x_0]$, and $P_0^+ = \mathbb{E}[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T]$. Then, the conventional Kalman filter algorithm proceeds by iterating between two steps. The time update is given as:

$$\hat{x}_{k+1}^- = F_k \hat{x}_k^+ \qquad P_{k+1}^- = F_k P_k^+ F_k^T + Q_k \tag{2}$$

The measurement update is given as:

$$y_{k+1} = z_{k+1} - H_{k+1}\hat{x}_{k+1}^- \qquad S_{k+1} = H_{k+1} P_{k+1}^- H_{k+1}^T + R_{k+1} \tag{3}$$

$$K_{k+1} = P_{k+1}^- H_{k+1}^T S_{k+1}^{-1} \tag{4}$$

$$\hat{x}_{k+1}^+ = \hat{x}_{k+1}^- + K_{k+1} y_{k+1} \qquad P_{k+1}^+ = (I - K_{k+1} H_{k+1}) P_{k+1}^- \tag{5}$$

2.2 Numerical Instability of the Kalman filter

In order for P to be statistically valid it must be (symmetric) positive definite. Briefly, this means that all of its eigenvalues are positive real numbers. This is for two reasons. First, from a modelling perspective, if its eigenvalues were zero, this would translate to a filter which completely trusts its estimates and consequently would avoid taking into account the subsequent measurements, placing all of its "belief" in the system model [4]. Second, from a numerical stability perspective, it does not suffice for the eigenvalues of P to be greater than zero, because if they are in close proximity to zero, then round-off errors could cause them to become negative, rendering it totally invalid [2,12,15].

In fact, the three equivalent forms to express the covariance measurement update are susceptible to numerical errors [4] and cannot guarantee the numerical stability of P. For example, the covariance update $P_k^+ = (I - K_k H_k) P_k^-$ is generally not preferred because it is too sensitive to round-off errors [4], which means neither the symmetry nor the positive definiteness of P_k can be guaranteed. That is because this update takes the product of nonsymmetric and symmetric matrices, a form which has been characterised as undesirable [20].

Alternatively, changing the covariance measurement update equation to $P_k^+ = P_k^- - K_k S_k K_k^T$ could potentially pose a *"serious numerical problem"* [20], such as P_k losing positive definiteness. Finally, while *Joseph's stabilised form* [8], given by $P_k^+ = (I - K_k H_k) P_k^- (I - K_k H_k)^T + K_k R_k K_k^T$, is considered to preserve the numerical robustness of P^+, it is not totally insensitive to numerical errors [4]. An additional disadvantage is the high computational complexity, which is $O(n^3)$ [12,20], since the number of arithmetic operations such as additions and multiplications is considerably higher compared to the simpler form.

To ameliorate these numerical problems, an alternative form of expressing the covariance time and measurement updates is using so-called *square-root filters*. These are generally considered superior to conventional filter implementations mainly because of their ability to increase the numerical stability of the propagation of the estimation-error covariance matrix P, and have often been described as outstanding [17,20]. It should be noted that the term square-root filter is mostly used to refer to the measurement update of the Kalman filter algorithm, since it is this part that can cause numerical problems [11]. They were motivated by the need for increased numerical precision because of word lengths of limited size in the 1960s [24] and by the concern with respect to the numerical accuracy of P in the measurement update of the Kalman filter equations [11]. Potter [5] proposed the idea of the so-called square-root filters and this idea was evolved ever since. The idea, which was limited to noiseless systems, is that P is factored into its square root C, such that $P = CC^T$, and as a result C is propagated through the measurement update equations, instead of P. Replacing P with its square-root factor C has the effect of doubling the numerical precision of the filter, thus making it particularly suitable for matrices which are not well-conditioned or when increased precision cannot be obtained from the hardware [11,12,20,24].

2.3 The Carlson-Schmidt Square-Root Filter

The Carlson-Schmidt filter is a form of a square-root filter which relies on the decomposition of P into its Cholesky factors in the time and measurement update equations. The Carlson part of the filtering algorithm, originally given by Carlson [9], corresponds to the measurement update, while the Schmidt part corresponds to the time update of the Kalman filter equations, respectively. Carlson's algorithm is capable of handling noise and, like Potter's algorithm, processes measurements as scalars. It factors P into the product of an upper-triangular Cholesky factor and its transpose such that $P = CC^T$. Note that unlike Potter's initial square-root filter where the factor C is not required to be triangular, in Carlson's square-root implementation the Cholesky factor C is an upper-triangular matrix. Maintaining C in upper-triangular form has been shown to provide several advantages in terms of storage and computational speed compared to Potter's algorithm [9,20]. While the choice between a lower and upper-triangular Cholesky factor C is arbitrary [20], Carlson motivated the preference to choose an upper-triangular Cholesky factor by the fact that in the time update part of the algorithm, fewer retriangularisation operations are required especially when someone designs a filter to be applied in a tracking or in a navigation problem, respectively [9].

3 Quantitative Verification of Kalman Filters

In this section, we describe our approach to modelling and verifying the numerical stability of Kalman filter implementations. This is based on the construction

and analysis of a probabilistic model (a discrete-time Markov chain) representing the behaviour of a particular Kalman filter executing in the context of estimating the state of a linear stochastic discrete-time system. The probabilistic model is automatically constructed based on a specification of the filter and the system whose state it is trying to estimate. Numerical stability properties are then verified using probabilistic model checking queries. We describe these phases in the following two sections.

3.1 Constructing Probabilistic Models of Kalman Filter Execution

We define a high-level modelling abstraction which can be instantiated to construct models of various different Kalman filter implementations. The modelling abstraction comprises three components: the first and second correspond to the system and measurement models along with their associated noise distributions; the third is the Kalman filter implementation itself used to estimate the state of the system model in the presence of uncertainty. The first two of these are defined mathematically along the lines described in Section 2.1. The third is specified in detail using a mainstream programming language, since it requires linear algebra data types and operations. Our implementation (see Section 4) uses Java and associated numerical libraries.

The DTMC which represents the evolution of the system model along with the filter estimates is not a *static* process. Rather it occurs in a *dynamic* fashion, involving the interaction of several components. For example, we do not assume that the measurements emitted from the system model are already given to us or that the filter estimates are already predetermined. Rather, as the system model evolves from state to state, the Kalman filter executes and tries to estimate its true state, imitating a real-time tracking scenario.

DTMC States and Transitions. The variables which define the Markov chain's states correspond to the system, measurement and filter models. All of these variables can be made independent of the filter implementations. For example, in a square-root filter implementation, C^+ can be either reconstructed or not in each time step, before being passed into the Markov chain's state, which demonstrates the modularity and extensibility of our approach.

The evolution of the states of the Markov chain corresponds to the system model perturbed by different noise values. Each of the Markov chain's states stores the 'true" values of the system model's state and the noisy measurements emitted at each time step k. These variables, along with the a posteriori state estimate and the estimation-error covariance, are included in the state of the Markov chain because they are needed for verification purposes. Then, before the Markov chain transitions to the next state (between time k and $k+1$), the time update of the corresponding filter variant is invoked. Both of the a priori variables depend on their a posteriori counterparts.

Specifically, once we are in a state for time instant k, our goal is to compute in the next state at time $k+1$ both the system model's updated state vector and the a posteriori variables of the respective filter, $\hat{x}^+$ and P^+. The a priori variables

of the Kalman filter types are encapsulated between these two updates as an intermediate step. Note that $\hat{x}$ and P are essentially the same variables which are used in the computation of both the a priori and a posteriori state estimates and estimation-error covariance matrices, respectively. What distinguishes x's semantics is whether the measurement z has been processed. This allows us to concretely define the notion of time k in each of the Markov chain's states.

In particular, a time instant k in the Markov chain can be thought of as encompassing: (i) state variables *before* the measurement is processed; and (ii) state variables *after* the measurement has been processed. Combining this temporal order into one state allows us to save storage by merging what would otherwise require two states to be represented.

The number of outgoing transitions and their probability values are determined by a *granularity level* of the noise, that we denote `gLevel`. The Gaussian distribution of the noise is discretised into `gLevel` disjoint intervals. The intervals used for each granularity level are shown in Table 1.

The measure used to determine these intervals is the standard deviation σ, which is a common practice in statistical contexts; see for example the so-called $68-95-99.7$ rule, which states that, assuming the data are normally distributed, then 68%, 95% and 99.7% of them will fall between one, two and three standard deviations of the mean, respectively. This statement can be expressed probabilistically as well by computing the cumulative distribution function (CDF) of a normally distributed random variable X, usually by converting it to its *standard* counterpart and using the so-called standard normal tables. While computing the probability that a noise value will fall inside an interval is relatively easy, the computation of its expected value is slightly more difficult. This is because we can choose to either truncate the distribution to intervals which contain the mean value of the distribution, which is the easier case, or to intervals which do not. For the first case, the expected value will be 0, which is the mean of distribution; for the second, this is not true.

Usually, for those cases, one might use a simple heuristic such as dividing the sum of the two endpoints of the interval by two, which is actually quite common. However, this might not be representative of the actual expected value since it does not weigh the values lying inside the interval according to the corresponding value of the density correctly. In other words, since the mean is also interpreted as the *"centre of gravity"* of the distribution [6], in the case of truncated intervals which do not contain the mean, more accurate techniques are needed. The probabilities of the Markov chain for a given granularity level are computed by first standardising the random variable, the noise in our case, and then evaluating its CDF at the two endpoints of the corresponding interval. Then, by subtracting them, we obtain the probability that it will fall within a certain interval.

Once the probabilities have been computed, it remains to find the expected value of the random variable for the corresponding intervals. In order to avoid the situation described earlier, and obtain the mean in a more accurate way, we have used the *truncated normal distribution* to compute the mean for the respective

Table 1: Intervals according to the granularity level.

gLevel	Intervals
2	$[-\infty..\mu]$, $[\mu.. + \infty]$
3	$[-\infty.. - 2\sigma]$, $[-2\sigma.. + 2\sigma]$, $[+2\sigma.. + \infty]$
4	$[-\infty.. - 2\sigma]$, $[-2\sigma..\mu]$, $[\mu.. + 2\sigma]$, $[+2\sigma.. + \infty]$
5	$[-\infty.. - 2\sigma]$, $[-2\sigma.. - \sigma]$, $[-\sigma.. + \sigma]$, $[+\sigma.. + 2\sigma]$, $[+2\sigma.. + \infty]$
6	$[-\infty.. - 2\sigma]$, $[-2\sigma.. - \sigma]$, $[-\sigma..\mu]$, $[\mu.. + \sigma]$, $[+\sigma.. + 2\sigma]$, $[+2\sigma.. + \infty]$

intervals. Formally, if a normal random variable X is normally distributed and lies within an interval $[a..b]$, where $-\infty \leq a \leq b \leq +\infty$, then X conditioned on $a < X < b$ has a truncated normal distribution. The PDF of a normally truncated random variable X is characterised by four parameters: (i-ii) the mean μ and standard deviation σ of the *original* distribution and (iii-iv) the lower and upper truncation points, a and b. Compactly, the mean value of the noise for a corresponding interval can be expressed as the conditional mean, $E[X|a < X < b]$, given by the following formula [14]:

$$E[X|a < X < b] = \mu + \sigma \frac{\phi(\frac{a-\mu}{\sigma}) - \phi(\frac{b-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})} \tag{6}$$

Note that in the expression above, ϕ and Φ denote the PDF and CDF of the standard normal distribution, respectively. Also note that the denominator has already been computed in the previous step, when the transition probabilities were computed. As a result, the computation of the transition probabilities and the conditional mean values for each of the corresponding intervals can be done in a unified manner.

3.2 Verification of Numerical Stability

Next, we discuss how to capture numerical stability properties for our Kalman filter models (see the earlier summary in Section 2) using the probabilistic temporal logic [10] of the PRISM model checker [18]. We explain the properties below, as we introduce them, and refer the reader to [10] for full details of the logic.

Verifying positive definiteness. In order to construct this property, we perform an eigenvalue-eigenvector decomposition of P^+ into the matrices $[V, D]$. The eigenvalues are obtained from the diagonal matrix D, and their positivity is determined and used to label each state of the Markov chain accordingly: we use an atomic proposition *isPD* for states in which $P+$ is positive definite.

We can then specify the probability that the matrix remains positive definite for the duration of execution of the filter using the formula $\mathtt{P}_{=?}[\,\Box\, \mathit{isPD}\,]$, where the temporal logic operator $\Box$, which is often referred to as as "always" or "globally", is used to represent invariance.

Examining the condition number of the estimation-error covariance matrix. The verification of certain numerical properties, such as those related to positive definiteness, is a challenging task and should be treated with caution. This is because, while convenient, focusing the verification on whether an event will occur or not, might not capture inherent numerical difficulties related to the numerical stability of state estimation algorithms. In other words, it does not suffice to check whether P^+ is positive definite or not by checking its eigenvalues because, as mentioned earlier, if they are in close proximity to zero, then round-off errors could cause them to become negative [12].

For example, it is often the case that estimation practitioners want to detect matrices that are close to becoming *singular*, a concept which is often referred to as *"detecting near singularity"* [7]. In other words, since a positive definite matrix is nonsingular, one wants to determine the "goodness" of P^+ in terms of its "closeness" to singularity, within some level of tolerance, usually the machine precision [12]. A matrix is said to be *well-conditioned* if it is "far" from singularity, while *ill-conditioned* describes the opposite. In order to quantify the goodness of P^+, we use the so-called *condition number*, which is a concept used in numerical linear algebra to provide an indication of the sensitivity of the solution of a linear equation (e.g. $Ax = b$), with respect to perturbations in b [12,20]. In our case, this concept is used to obtain a measure of goodness of P^+.

The condition number of P^+ is given as $\kappa(P^+) = \sigma_{max}/\sigma_{min}$, where σ_{max} and σ_{min} are the maximum and minimum singular values, respectively [11,20]. These can be obtained by performing the singular value decomposition (SVD) of P^+. A "small" condition number indicates that the matrix is well-conditioned and nonsingular, while a "large" condition number indicates the exact opposite. Note that the smallest condition number is 1 when $\sigma_{max} = \sigma_{min}$.

We express this property as the formula $\mathtt{R}^{cond}_{=?}[\,\mathtt{I}^{=k}\,]$, which gives the expected value of the condition number after k time steps. We assign the condition number to each state of the DTMC using a reward function *cond* and we set k to be `maxTime`, the period of time for which we verify the respective filter variant.

Providing bounds on numerical errors. Another useful aspect of the condition number is that it can be used to obtain an estimate of the precision loss that numerical computations could cause to P^+. For instance, for a single precision and a double precision floating-point number format, the precision is about 7 and 16 decimal digits, respectively. Since our computations take place in the decimal number system, the logarithm of the condition number (e.g. $log_{10}(\kappa(P^+))$), gives us the ability to define more concretely when a condition number will be considered "large" or "small" [3,20,24]. For example, a $log_{10}(\kappa(P^+)) > 6$ and a $log_{10}(\kappa(P^+)) > 15$ could cause numerical problems in the estimation-error covariance computation and render P^+ as ill-conditioned when implemented in a single and a double precision floating-point number format, respectively.

So, to verify this property we construct a closed interval whose endpoints will be based on the appropriate values of the numerical quantity of $log_{10}(\kappa(P^+))$. This lets us label states whose $log_{10}(\kappa(P^+))$ value will fall within "acceptable" values in the interval, when, for instance, double precision is used. We then use

Table 2: User inputs for each of the models.

Input	Description	Used in:	Type
$\hat{x}_0^+$	A posteriori state estimate vector	Filter	`RealVector`
P_0^+	A posteriori estimation-error covariance matrix	Filter	`RealMatrix`
x	State vector	System	`RealVector`
w	Process noise vector	System	`RealVector`
v	Measurement noise vector	System	`RealVector`
F	State transition matrix	Shared	`RealMatrix`
Q	Process noise covariance matrix	Filter	`RealMatrix`
H	Measurement matrix	Shared	`RealMatrix`
R	Measurement noise covariance matrix	Shared	`RealMatrix`
`gLevel`	Granularity of the noise	Shared	`int`
`decPlaces`	Number of decimal places	Shared	`int`
`maxTime`	Maximum time the model will run	Shared	`int`
`filterType`	Type of filter variant	Shared	`int`

the property $\mathtt{P}_{=?}[\,\Box\, \mathit{isCondWithin}\,]$, in a similar fashion to the first property above, where *isCondWithin* labels the "acceptable" states. A probability value of less than 1 should raise an alarm that numerical errors may be encountered.

4 Tool Support: VerFilter

Next, we provide some details about the tool, VerFilter, which is the software implementation of the framework defined in Section 3. The VerFilter tool is written in the Java programming language in order to be seamlessly integrated with the PRISM libraries, which are written in Java as well. The tool and supporting files for the results in the next section are available from [27].

VerFilter Inputs. In Table 2 we show the user inputs available to VerFilter, by distinguishing which of those refer to the system and measurement model, which refer specifically to the filter models and which are shared between them. The `RealVector` and `RealMatrix` shown in Table 2 are implemented as one-dimensional and two-dimensional arrays of type `double`, respectively. VerFilter also takes as inputs four extra parameters: (i) `gLevel` which takes an integer between 2 and 6, and has been discussed in Section 3.1; (ii) `decPlaces` which allows the user to specify an integer between 2 and 15, the number of decimal places, to which the numerical values used in the computations will be rounded; (iii) `maxTime` which is an integer and determines the maximum time the model will run; and (iv) `filterType` which is the type of filter to be executed.

VerFilter Algorithms. In this paper, we focus on two of our filter variants: the conventional Kalman filter (`CKFilter`) and the Carlson-Schmidt square-root filter (`SRFilter`). In VerFilter, several of the numerical linear algebra computations for implementing Kalman filters are done using the Apache Commons Math library [1], while other parts have been manually implemented. In `CKFilter`, for

example, the library is used for "basic" matrix operations and for the eigen and singular value decomposition of P. For `SRFilter`, algorithms implemented manually include the upper-triangular Cholesky factorisation and Carlson's measurement update with Schmidt's time update using *Householder transformations*.

5 Experimental Results

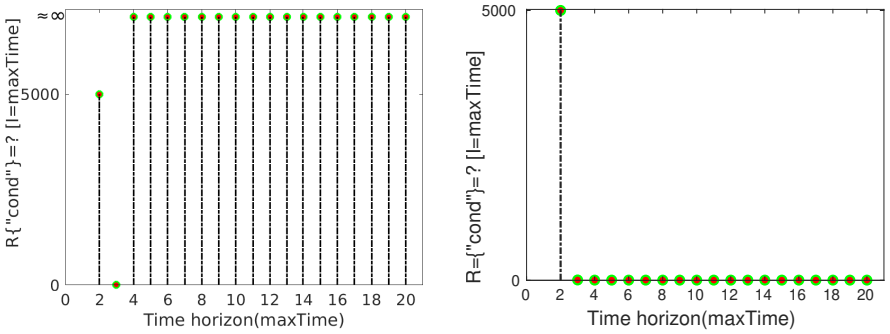
We now illustrate results from the implementation of our techniques on the two filters `CKFilter` and `SRFilter` mentioned above. For the system models in our experiments, we use two distinct *kinematic state models* which describe the motion of objects as a function of time. For the first, the *discrete white noise acceleration model* (DWNA), the initial estimation-error covariance matrix P_0^+ is defined as $\begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$. Defining P_0^+ as a diagonal matrix is quite common, since it is initially unknown whether the state variables are correlated to each other. The process noise covariance matrix is given by $Q = \Gamma\sigma_w^2\Gamma^T$ where the noise gain matrix $\Gamma = [\frac{1}{2}\Delta t^2 \ \Delta t]^T$ is initialised by setting the sampling interval Δt to 1, which results in $\Gamma = [0.5 \ 1]^T$. The variance σ_w^2 is set to 0.001 initially. For the second model, the *continuous white noise acceleration model* (CWNA), σ_w^2 is initially set to 0.001. Note that each of these models results in a different process noise covariance matrix Q. For more details on these models, see [27].

5.1 Verification of Kalman Filter Implementations

In the first set of experiments, shown in Fig. 1, we analyse the condition number of P^+, in order to verify that it remains well-conditioned in terms of maintaining its nonsingularity as it is being propagated forward in time (as discussed in Section 3.2). This property is verified against two inputs which we vary; the first is the numerical precision in terms of the number of decimal places, which we vary from 3 to 6 inclusive. The second input is the time horizon of the model which in our case is measured in discrete time steps and is varied from 2 to 20.

Our goal is twofold. Firstly, we examine whether an increase in the numerical precision has a meaningful effect on how accurately the condition number is computed. This is important since, as we show in Section 5.2, a decrease in the numerical precision usually makes verification more efficient. Being able to consider an appropriate threshold above which an increase in the numerical precision will not have an effect on the property to be verified can determine the applicability of these verification mechanisms in realistic settings. Secondly, we examine whether letting the model evolve for a greater amount of time could have an impact on the property that is being verified.

The first observation between Fig. 1a and 1b is that the increased numerical precision actually determines the verification result. For example, we note that for `maxTime` values in the range of $[4 - 20]$, when the input to our model for the numerical precision is 3 decimal places, the instantaneous reward jumps to infinity. An infinite reward in this case means that the condition number of P^+



(a) 3 decimal places (b) 4 - 6 decimal places

Fig. 1: Condition number of $P+$ over time under various degrees of precision.

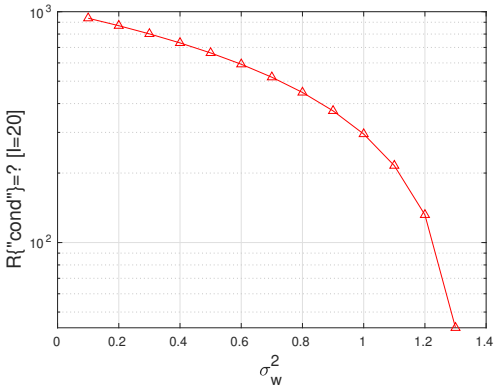


Fig. 2: Verifying goodness of P^+

is $\approx$ 1.009e+16, which practically means that P^+ is "computationally" singular and consequently positive definiteness is not being preserved. Conversely, when we increase the numerical precision to a value > 4, positive definiteness is preserved and the instantaneous reward assigned to the states fluctuates around small values close to zero. Another interesting observation is that the instantaneous rewards stabilise to a value of ≈ 3, irrespective of whether the numerical precision is 4, 5 or 6. In fact, the actual absolute difference of the rewards over the states in which positive definiteness is preserved between a numerical precision of 5 and 6 decimal places, is ≈ 0.1.

In the second set of experiments the system model is a CWNA kinematic model. Our goal is to examine how VerFilter can be used to examine heuristic-based approaches and ad-hoc methods such as artificial noise injection in terms of their usefulness in correcting potential numerical problems in P^+. This is also helpful in situations where it is challenging to determine the elements of Q, by

Table 3: Comparison between two filter variants.

CKFilter	**SRFilter**	**CKFilter**	**SRFilter**
$\mathtt{P}_{=?}[\,\Box\, isPD\,]$	$\mathtt{P}_{=?}[\,\Box\, isPD\,]$	$\mathtt{R}^{cond}_{=?}[\,\mathtt{I}^{=maxTime}\,]$	$\mathtt{R}^{cond}_{=?}[\,\mathtt{I}^{=maxTime}\,]$
1	1	5001	69.88
1	1	6.85	2.48
0	1	$+\infty$	2.01
0	1	$+\infty$	1.94
0	1	$+\infty$	1.94
0	1	$+\infty$	1.94

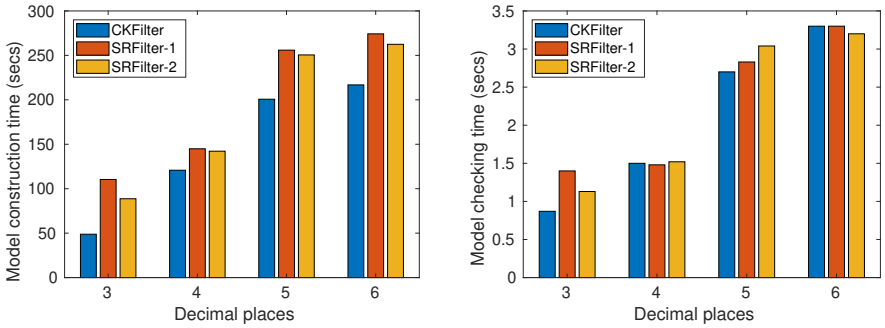
performing an automatic search over those values which will produce an optimal performance, in this case in terms of the numerical robustness of P.

To this end, we verify whether P^+ will remain well-conditioned or not, by varying the elements of Q. The noise variance σ_w^2, which determines the elements of Q, is the input to our model, P^+ is being verified against. We do not vary the maximum time; rather, we let the Markov chain evolve to a fixed `maxTime` value of 20 time steps, which corresponds to $\approx 1 \times 10^6$ states.

In Fig. 2 we show the effects of increasing the variance of the noise by small increments, which is then multiplied with the elements of Q. The first point of the plot $(0.1, 1000)$, means that for a value of $\sigma_w^2 = 0.1$, the corresponding instantaneous reward which corresponds to the condition number of P^+ in a set of states where `maxTime`=20, is 1000. As we increase σ_w^2, the "quality" of P^+ increases, reaching a condition number of ≈ 43.

In summary, for this particular example, the optimal $\sigma_w^2 = 1.3$. It is important to note that when performing verification on Markov chains whose trajectories evolve over multiple states, to verify that the positive definiteness of P^+ is not destroyed between successive states (i.e. successive time steps). To this end, it is advisable to use a property of the form $\mathtt{P}_{=?}[\,\Box\, isPD\,]$ and reject models in which the previous property is not satisfied with probability one.

In Table 3 we compare two of the filter variants available in VerFilter; the `CKFilter` and the `SRFilter`. In this set of experiments, the setup is similar to the first one. First, our purpose is to demonstrate the correctness of our approach by comparing the condition numbers of P^+ and C^+, respectively. The superiority of the `SRFilter` compared to `CKFilter`, is demonstrated from the fact that for the same set of parameters the numerical robustness of P^+ is preserved. This can be seen by comparing the computed results of the reward-based properties as shown in the third and fourth column of Table 3. We note that when choosing the `CKFilter`, the reward value shoots up to $+\infty$, representing an estimation-error covariance matrix in which the PD property is destroyed, while in the `SRFilter` case the corresponding reward value settles around the small value of 1.94. This is also evident by observing the first and second columns of Table 3 which tell us whether the PD invariant will be maintained in all the states of the model. Notably, the PD property in the `CKFilter` does not hold for every state, in fact



(a) Model construction time (b) Model checking time

Fig. 3: Time comparisons between three filters.

the probability is zero, while for the `SRFilter` the PD property holds for every state with probability one.

5.2 Scalability analysis

In this section, we report on the scalability of our approach in terms of the model construction and model checking time, across three filter variants. The model has been generated by letting the Markov chain evolve to a fixed `maxTime` value of 20 time steps, which corresponds to $\approx 1 \times 10^6$ states. The rationale behind this section is to emphasise the careful analysis that needs to be performed to systematically evaluate the trade-offs between the accuracy of the verification result and the fastness of the verification algorithms.

In Fig. 3 we show the time comparisons, for varying degrees of precision, between a model which encodes the conventional Kalman filter (`CKFilter`), and our two implementations of the Carlson-Schmidt square-root filter with (`SRFilter-1`) and without (`SRFilter-2`) reconstruction of the estimation-error covariance matrix, respectively. The model checking time refers to the total time it takes to verify the first and second property of Section 3.2. These sets of experiments were run on a 16GB RAM machine with an i7 processor at 1.80GHz, running Ubuntu 18.04.

By observing Fig. 3a it is apparent that the increased numerical precision affects the construction time of the models. The average model construction time of the three filter variants increased by a factor of ≈ 3 from 3 to 6 decimal places. Specifically, the average time is ≈ 83 seconds for 3 decimal places compared to ≈ 249 seconds, when 6 decimal places were used. Moreover, the construction of the `CKFilter` was the fastest in all the degrees of precision considered, however, as it was noted in Section 5.1 it produces an inaccurate verification result when the number of decimal places is 3.

Conversely, the construction times of the two square-root filters were about the same, and it seems that the extra computational step ($P = CC^T$) did not have a significant effect on the performance of the model construction. However, it should be borne in mind that these experiments were conducted on systems represented by two-dimensional matrices. The model checking times are shown in Fig. 3b and one can observe that they follow a similar pattern with the model construction times shown earlier, in terms of the increase in time from 3 to 6 decimal places. For instance, the average model checking time increases by a factor of ≈ 3 when 6 decimal places are used, compared to 3.

Another observation is that the model checking time appears to be independent of the type of the filter used. This can be seen from the limited variability the model checking time experiences between the three filter variants, since for the degrees of precision considered, it remains at approximately the same level. This is in contrast to the model construction time which appears to be affected by the filter type, since it is considerably less for the `CKFilter` compared to its square-root variants. In fact, for a precision of 6 decimal places, and once `CKFilter` is chosen as an input we experience a drop in the model construction time of about 53 seconds. However, for the same amount of precision, the time it takes to model check all the three filters is around 3 seconds.

6 Conclusion

We have presented a framework for the modelling and verification of Kalman filter implementations. It is general enough to analyse a variety of different implementations, and various system models, and to study a range of numerical issues which may hinder the effective deployment of the filters in practice. We have implemented the techniques in a tool and illustrated its applicability and scalability with a range of experiments. Due to space limitations, we showed results for two filters, the conventional Kalman filter and for the Carlson-Schmidt square-root filter, but our implementation already supports three others.

In general, the evaluation of Kalman filters in terms of their performance has attracted considerable attention, since the early days of their development. However, formal methods such as probabilistic model checking have not been used for their verification. This is, to the best of our knowledge, the first work where these types of problems are applied to a probabilistic verification setting. Our main contribution in this work is that we show that probabilistic verification can be a promising alternative in verifying these types of systems.

Acknowledgements. This work has been partially supported by an EPSRC-funded Ph.D. studentship (award ref: 1576386) and the PRINCESS project (contract FA8750-16-C-0045) funded by the DARPA BRASS programme.

References

1. Math - Commons-Math: The Apache Commons Mathematics Library, http://commons.apache.org/math/
2. Anderson, B., Moore, J.: Optimal Filtering. Dover Books on Electrical Engineering, Dover Publications (2012)
3. Bar-Shalom, Y.: Tracking and Data Association. Academic Press Professional, Inc., San Diego, CA, USA (1987)
4. Bar-Shalom, Y., Li, X.R.: Estimation with Applications to Tracking and Navigation. John Wiley & Sons, Inc., New York, NY, USA (2001). https://doi.org/10.1002/0471221279
5. Battin, R.H.: Astronautical guidance. Electronic sciences, McGraw-Hill (1964)
6. Bertsekas, D., Tsitsiklis, J.: Introduction to Probability. Athena Scientific optimization and computation series, Athena Scientific (2008)
7. Bierman, G.J.: Factorization Methods for Discrete Sequential Estimation (1977)
8. Bucy, R.S., Joseph, P.D.: processes with applications to guidance. Interscience Publishers New York (1968)
9. Carlson, N.A.: Fast triangular formulation of the square root filter. AIAA Journal **11**(9), 1259–1265 (1973). https://doi.org/10.2514/3.6907
10. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: Bernardo, M., Issarny, V. (eds.) Formal Methods for Eternal Networked Software Systems (SFM'11). LNCS, vol. 6659, pp. 53–113. Springer (2011). https://doi.org/10.1007/978-3-642-21455-4_3
11. Gibbs, B.P.: Advanced Kalman Filtering, Least Squares and Modeling: A Practical Handbook. John Wiley & Sons, Inc. (2011). https://doi.org/0.1002/9780470890042
12. Grewal, M.S., Andrews, A.P.: Kalman Filtering: Theory and Practice Using MATLAB. Wiley-IEEE Press, 4th edn. (2014)
13. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing **6**(5), 512–535 (1994). https://doi.org/10.1007/BF01211866
14. Johnson, N.L., Kotz, S., Balakrishnan, N.: Continuous univariate distributions. New York: Wiley (1994)
15. Kailath, T.: Linear Systems. Prentice-Hall, Englewood Cliffs, N.J (1980)
16. Kalman, R.E.: A new approach to linear filtering and prediction problems. ASME Journal of Basic Engineering (1960)
17. Kaminski, P., Bryson, A., Schmidt, S.: Discrete square root filtering: A survey of current techniques. IEEE Transactions on Automatic Control **16**(6), 727–736 (December 1971). https://doi.org/10.1109/TAC.1971.1099816
18. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proc. 23rd International Conference on Computer Aided Verification (CAV'11). LNCS, vol. 6806, pp. 585–591. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_47
19. Li, X.R., Jilkov, V.P.: Survey of maneuvering target tracking. part i. dynamic models. IEEE Transactions on Aerospace and Electronic Systems **39**(4), 1333–1364 (Oct 2003). https://doi.org/10.1109/TAES.2003.1261132
20. Maybeck, P.S.: Stochastic models, estimation, and control: Volume 1. Mathematics in science and engineering, Elsevier Science, Burlington, MA (1982)
21. Moulin, M., Gluhovsky, L., Bendersky, E.: Formal verification of maneuvering target tracking. AIAA Guidance, Navigation, and Control Conference and Exhibit (2003). https://doi.org/10.2514/6.2003-5716

22. R. Gamboa, J. Cowles, J.V.B.: On the verification of synthesized kalman filters. In: 4th International Workshop on the ACL2 Theorem Prover and Its Applications. (2003)
23. Roşu, G., Venkatesan, R.P., Whittle, J., Leuştean, L.: Certifying Optimality of State Estimation Programs, pp. 301–314. Springer Berlin Heidelberg, Berlin, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45069-630
24. Simon, D.: Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches. Wiley-Interscience (2006)
25. Whittle, J., Schumann, J.: Automating the implementation of kalman filter algorithms. ACM Trans. Math. Softw. **30**(4), 434–453 (Dec 2004). https://doi.org/10.1145/1039813.1039816
26. Zarchan, P., Musoff, H.: Fundamentals of Kalman filtering : a practical approach. American Institute of Aeronautics and Astronautics, Reston, VA, 4 edn. (2015)
27. Supporting material, www.prismmodelchecker.org/files/fm19kf/