7

IMPLEMENTATION METHODS

There is a great difference between theory and practice.

Giacomo Antonelli (1806–1876)[1]

7.1 CHAPTER FOCUS

Up to this point, we have discussed what Kalman filters are and how they are supposed to behave. Their theoretical performance has been shown to be characterized by the covariance matrix of estimation uncertainty, which is computed as the solution of a matrix Riccati differential equation or difference equation.

However, soon after the Kalman filter was first implemented on computers, it was discovered that the observed mean-squared estimation errors were often much larger than the values predicted by the covariance matrix, even with simulated data. The variances of the filter estimation errors were observed to diverge from their theoretical values, and the solutions obtained for the Riccati equation were observed to have negative variances, an embarrassing example of a theoretical impossibility. The problem was eventually determined to be caused by computer roundoff, and alternative implementation methods were developed for dealing with it.

[1]In a letter to the Austrian Ambassador, as quoted by Lytton Strachey in *Eminent Victorians* [1], Cardinal Antonelli was addressing the issue of papal infallibility, but the same might be said about the infallibility of numerical processing systems.

This chapter is primarily concerned with

1. how computer roundoff can degrade Kalman filter performance,
2. alternative implementation methods that are more robust against roundoff errors, and
3. the relative computational costs of these alternative implementations.

7.1.1 Main Points to Be Covered

The main points to be covered in this chapter are the following:

1. Computer roundoff errors can and do seriously degrade the performance of Kalman filters.
2. Solution of the matrix Riccati equation is a major cause of numerical difficulties in the conventional Kalman filter implementation, from the standpoint of computational load as well as from the standpoint of computational errors.
3. Unchecked error propagation in the solution of the Riccati equation is a major cause of degradation in filter performance.
4. Asymmetry of the covariance matrix of state estimation uncertainty is a symptom of numerical degradation and a cause of numerical instability and measures to symmetrize the result can be beneficial.
5. Numerical solution of the Riccati equation tends to be more robust against roundoff errors if the so-called "square roots" of the covariance matrix are used as the dependent variables.
6. Numerical methods for solving the Riccati equation in terms of these matrix "square roots" are called *factorization methods*, and the resulting Kalman filter implementations are collectively called *"square-root" filtering*.
7. Information filtering is an alternative state vector implementation that improves numerical stability properties. It is especially useful for problems with very large initial estimation uncertainty.

7.1.2 Topics Not Covered

1. *Parametric Sensitivity Analysis.* The focus here is on numerically stable implementation methods for the Kalman filter. Numerical analysis of *all* errors that influence the performance of the Kalman filter would include the effects of errors in the assumed values of all model parameters, such as Q, R, H, and Φ. These errors also include truncation effects due to finite precision. The sensitivities of performance to these types of modeling errors can be modeled mathematically, but this is not done here.
2. *Smoothing Implementations.* There have been significant improvements in smoother implementation methods beyond those presented in Chapter 5. The interested reader is referred to the surveys by Meditch [2] (methods up to

1973) and McReynolds [3] (up to 1990) and to earlier results by Bierman [4] and by Watanabe and Tzafestas [5].

3. *Parallel Computer Architectures for Kalman Filtering.* The operation of the Kalman filter can be speeded up, if necessary, by performing some operations in parallel. The algorithm listings in this chapter indicate those loops that can be performed in parallel, but no serious attempt is made to define specialized algorithms to exploit concurrent processing capabilities. An overview of theoretical approaches to this problem is presented by Jover and Kailath [6].

7.2 COMPUTER ROUNDOFF

Roundoff errors are a side effect of computer arithmetic using fixed- or floating-point data words with a fixed number of bits. Computer roundoff is a fact of life for most computing environments.

Example 7.1 (Roundoff Errors) In binary representation, the rational numbers are transformed into sums of powers of 2, as follows:

$$\begin{aligned}1 &= 2^0\\ 3 &= 2^0 + 2^1\\ \frac{1}{3} &= \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \frac{1}{256} + \cdots\\ &= 0_b 010101010101010101010101010 \ldots,\end{aligned}$$

where the subscript "b" represents the "binary point" in binary representation (so as not to be confused with the "decimal point" in decimal representation). When 1 is divided by 3 in an IEEE/ANSI standard [7] single-precision floating-point arithmetic, the 1 and the 3 can be represented precisely, but their ratio cannot. The binary representation is limited to 24 bits of mantissa.[2] The above result is then rounded to the 24-bit approximation (starting with the leading "1"):

$$\begin{aligned}\frac{1}{3} &\approx 0_b 0101010101010101010101011\\ &= \frac{11184811}{33554432}\\ &= \frac{1}{3} - \frac{1}{100663296},\end{aligned}$$

giving an approximation error magnitude of about 10^{-8} and a relative approximation error of about 3×10^{-8}. The difference between the true value of the result and the value approximated by the processor is called *roundoff error*.

[2]The mantissa is the part of the binary representation starting with the leading nonzero bit. Because the leading significant bit is always a "1," it can be omitted and replaced by the sign bit. Even including the sign bit, there are effectively 24 bits available for representing the magnitude of the mantissa.

7.2.1 Unit Roundoff Error

Computer roundoff for floating-point arithmetic is often characterized by a single parameter $\varepsilon_{\text{roundoff}}$, called the *unit roundoff error*, and defined in different sources as the largest number such that either

$$1 + \varepsilon_{\text{roundoff}} \equiv 1 \text{ in machine precision} \tag{7.1}$$

or

$$1 + \varepsilon_{\text{roundoff}}/2 \equiv 1 \text{ in machine precision.} \tag{7.2}$$

The name "`eps`" in MATLAB®is the parameter satisfying the second of these equations. Its value may be found by typing "`eps`$\langle\hookleftarrow\rangle$" (i.e., typing "`eps`" without a following semicolon, followed by hitting the "Return" or "Enter" key) in the MATLAB command window. Entering "`-log2(eps)`" should return the number of bits in the mantissa of the standard data word.

7.2.2 Effects of Roundoff on Kalman Filter Performance

7.2.2.1 Early Discoveries Around the time the Kalman filter was introduced in 1960, the International Business Machines Corporation (IBM) was introducing its 7000-series transisterized computers with 36-bit floating-point arithmetic units. This was considered quite revolutionary at the time, but—even with all this precision—computer roundoff in the first Kalman filter implementations was a serious problem. Early accounts by Schmidt [8, 9] and Battin [10] emphasize the difficulties encountered with computer roundoff, and the serious risk of failure it entailed for in-flight implementations in the Apollo moon missions. The problem was eventually solved satisfactorily by finding new implementation methods, and even better methods would be discovered after the last Apollo missions in the early 1970s.

Many of these roundoff problems occurred on computers with much shorter wordlengths than those available for modern MATLAB implementations and with less accurate implementations of bit-level arithmetic than current microprocessors adhering to current ANSI standards [7].

However, the next example (modified after one from Dyer and McReynolds [11] credited to R. J. Hanson) demonstrates that roundoff can still be a problem in Kalman filter implementations in MATLAB environments and how a problem that is well conditioned, as posed, can be made ill-conditioned by the filter implementation.

Example 7.2 Let I_n denote the $n \times n$ identity matrix. Consider the filtering problem with measurement sensitivity matrix

$$H = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1+\delta \end{bmatrix}$$

and covariance matrices

$$P_0 = I_3 \quad \text{and} \quad R = \delta^2 I_2,$$

where $\delta^2 < \varepsilon_{\text{roundoff}}$ but $\delta > \varepsilon_{\text{roundoff}}$. In this case, although H clearly has rank = 2 in machine precision, the product HP_0H^{T} *with roundoff* will equal

$$\begin{bmatrix} 3 & 3+\delta \\ 3+\delta & 3+2\delta \end{bmatrix},$$

which is singular. The result is unchanged when R is added to HP_0H^{T}. In this case, then, the filter observational update fails because the matrix $HP_0H^{\text{T}} + R$ is not invertible.

Sneak Preview of Alternative Implementations Figure 7.1 illustrates how the standard Kalman filter and some of the alternative implementation methods perform on the variably ill-conditioned problem of Example 6.2 (implemented as MATLAB m-file `shootout.m` on the accompanying diskette) as the conditioning parameter $\delta \to 0$. All solution methods were implemented in the same precision (64-bit floating point) in MATLAB. The labels on the curves in this plot correspond to the names of the corresponding m-file implementations on the accompanying diskette. These are also the names of the authors of the corresponding methods, the details of which will be presented further on.

For this particular example, the accuracies of the methods labeled "Carlson" and "Bierman" appear to degrade more gracefully than the others as $\delta \to \varepsilon$, the machine precision limit. The Carlson and Bierman solutions still maintain about nine digits ($\approx$30 bits) of accuracy at $\delta \approx \sqrt{\varepsilon}$, when the other methods have essentially no bits of accuracy in the computed solution.

This one example, by itself, does not prove the general superiority of the Carlson and Bierman solutions for the observational updates of the Riccati equation. The full implementation will require a compatible method for performing the temporal update, as well. However, the observational update had been the principal source of difficulty with the conventional implementation.

7.2.3 Terminology of Numerical Error Analysis

We first need to define some general terms used in characterizing the influence of roundoff errors on the accuracy of the numerical solution to a given computation problem.

7.2.3.1 Robustness and Numerical Stability These terms are used to describe qualitative properties of arithmetic problem-solving methods. *Robustness* refers to the relative insensitivity of the solution to errors of some sort. *Numerical stability* refers to robustness against roundoff errors.

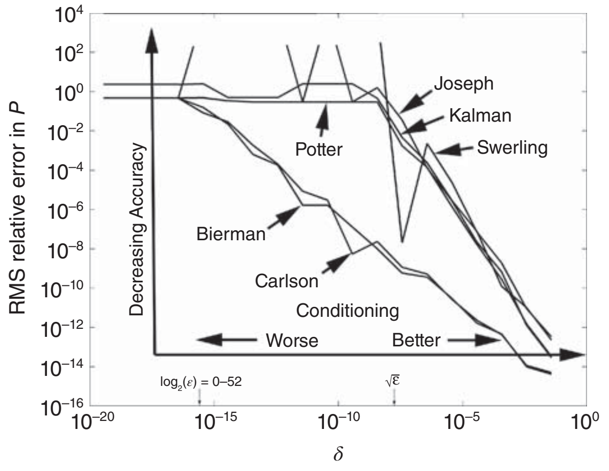

Figure 7.1 Degradation of the Riccati equation observational updates with problem conditioning.

7.2.3.2 Precision versus Numerical Stability Relative roundoff errors can be reduced by using more precision (i.e., more bits in the mantissa of the data format), but the accuracy of the result is also influenced by the accuracy of the initial parameters used and the procedural details of the implementation method. Mathematically equivalent implementation methods can have very different numerical stabilities at the same precision.

7.2.3.3 Numerical Stability Comparisons Numerical stability comparisons can be slippery. Robustness and stability of solution methods are matters of degree, but implementation methods cannot always be totally ordered according to these attributes. Some methods are considered more robust than others, but their relative robustness can also depend upon intrinsic properties of the problem being solved.

7.2.3.4 Iii-Conditioned and Well-Conditioned Problems In the analysis of numerical problem-solving methods, the qualitative term *conditioning* is used to describe the sensitivity of the error in the output (solution) to variations in the input data (problem). This sensitivity generally depends on the input data and the solution method.

A problem is called *well conditioned* if the solution is not “badly” sensitive to the input data and *ill-conditioned* if the sensitivity is “bad.” The definition of what is bad generally depends on the uncertainties of the input data and the numerical precision being used in the implementation. One might, for example, describe a matrix A as being “ill-conditioned with respect to inversion” if A is “close” to being singular. The definition of “close” in this example could mean within the uncertainties in the values of the elements of A or within machine precision.

Example 7.3 (Condition Number of a Matrix) The sensitivity of the solution x of the linear problem $Ax = b$ to uncertainties in the input data (A and b) and roundoff errors is characterized by the condition number of A, which can be defined as the ratio

$$\text{cond}(A) = \frac{\max_x \|Ax\|/\|x\|}{\min_x \|Ax\|/\|x\|} \tag{7.3}$$

if A is nonsingular and as ∞ if A is singular. It also equals the ratio of the largest and smallest characteristic values of A. Note that the condition number will always be ≥ 1 because max $\geq$ min. As a general rule in matrix inversion, condition numbers close to 1 are a good omen, and increasingly larger values are cause for increasing concern over the validity of the results.

The relative error in the computed solution $\hat{x}$ of the equation $Ax = b$ is defined as the ratio $\|\hat{x} - x\|/\|x\|$ of the magnitude of the error to the magnitude of x.

As a rule of thumb, the maximum relative error in the computed solution is bounded above by $c_A \varepsilon_{\text{roundoff}}\,\text{cond}(A)$, where $\varepsilon_{\text{roundoff}}$ is the unit roundoff error in computer arithmetic (defined in Section 7.2.1) and the positive constant c_A depends on the dimension of A. The problem of computing x, given A and b, is considered ill-conditioned if adding 1 to the condition number of A in computer arithmetic has no effect. That is, the logical expression $1 + \text{cond}(A) = \text{cond}(A)$ evaluates to true.

Consider an example with the coefficient matrix

$$A = \begin{bmatrix} 1 & L & 0 \\ 0 & 1 & L \\ 0 & 0 & 1 \end{bmatrix},$$

where

$$\begin{aligned} L &= 2^{64} \\ &= 18,446,744,073,709,551,616, \end{aligned}$$

which is such that computing L^2 would cause overflow in ANSI standard single-precision arithmetic.

The condition number of A will then be

$$\text{cond}(A) \approx 3.40282 \times 10^{38}.$$

This is about 31 orders of magnitude beyond where the rule-of-thumb test for ill-conditioning would fail in this precision ($\approx 2 \times 10^7$). One would then consider A extremely ill-conditioned for inversion (which it is) even though its determinant equals 1.

Programming Note: For the general linear equation problem $Ax = b$, it is not necessary to invert A explicitly in the process of solving for x, and numerical stability is generally improved if matrix inversion is avoided. The MATLAB matrix divide (using x = A\b) does this.

7.2.4 Ill-Conditioned Kalman Filtering Problems

For Kalman filtering problems, the solution of the associated Riccati equation should equal the covariance matrix of actual estimation uncertainty, which should be optimal with respect to all quadratic loss functions. The computation of the Kalman (optimal) gain depends on it. If this does not happen, the problem is considered ill-conditioned. Factors that contribute to such ill-conditioning include the following:

1. Large uncertainties in the values of the matrix parameters Φ, Q, H, or R. Such modeling errors are not accounted for in the derivation of the Kalman filter.
2. Large ranges of the actual values of these matrix parameters, the measurements, or the state variables—all of which can result from poor choices of scaling or dimensional units.
3. Ill-conditioning of the intermediate result $R^* = HPH^{\mathrm{T}} + R$ for inversion in the Kalman gain formula.
4. Ill-conditioned theoretical solutions of the matrix Riccati equation—without considering numerical solution errors. With numerical errors, the solution may become indefinite, which can destabilize the filter estimation error.
5. Large matrix dimensions. The number of arithmetic operations grows as the square or cube of matrix dimensions, and each operation can introduce roundoff errors.
6. Poor machine precision, which makes the relative roundoff errors larger.

Some of these factors are unavoidable in many applications. Keep in mind that they do not *necessarily* make the Kalman filtering problem hopeless. However, they are cause for concern—and for considering alternative implementation methods.

7.3 EFFECTS OF ROUNDOFF ERRORS ON KALMAN FILTERS

7.3.1 Quantifying the Effects of Roundoff Errors on Kalman Filtering

Although there was early experimental evidence of divergence due to roundoff errors, it has been difficult to obtain general principles describing how it is related to characteristics of the implementation. There are some general (but somewhat weak) principles relating roundoff errors to characteristics of the computer on which the filter is implemented and to properties of the filter parameters. These include the results of Verhaegen and Van Dooren [12] on the numerical analysis of various implementation methods in Kalman filtering. These results provide upper bounds on the propagation of roundoff errors as functions of the norms and singular values of key matrix variables. They show that some implementations have better bounds than others. In particular, they show that certain "symmetrization" procedures are provably beneficial and that the so-called "square-root" filter implementations have generally better error propagation bounds than the conventional Kalman filter equations.

Let us examine the ways that roundoff errors propagate in the computation of the Kalman filter variables and how they influence the accuracy of results in the Kalman filter. Finally, we provide some examples that demonstrate common failure modes.

7.3.2 Roundoff Error Propagation in Kalman Filters

7.3.2.1 Heuristic Analysis We begin with a heuristic look at roundoff error propagation, from the viewpoint of the data flow in the Kalman filter, to show how roundoff errors in the Riccati equation solution are not controlled by feedback like roundoff errors in the estimate. Consider the matrix-level data flow diagram of the Kalman filter that is shown in Figure 7.2. This figure shows the data flow at the level of vectors and matrices, with operations of addition ($\oplus$), multiplication ($\otimes$), and inversion ($I\div$). Matrix transposition need not be considered a data operation in this context, because it can be implemented by index changes in subsequent operations. This data flow diagram is fairly representative of the straightforward Kalman filter algorithm, the way it was originally presented by Kalman, and as it might be implemented in MATLAB by a moderately conscientious programmer. That is, the diagram shows how partial results (including the Kalman gain, $\overline{K}$) might be saved and reused. Note that the internal data flow can be separated into two semi-independent loops within the dashed boxes. The variable propagated around one loop is the state estimate. The variable propagated around the other loop is the covariance matrix of estimation uncertainty. (The diagram also shows some of the loop "shortcuts" resulting from reuse of partial results, but the basic data flows are still loops.)

7.3.2.2 Feedback in the Estimation Loop The uppermost of these loops, labeled EST. LOOP, is essentially a feedback error correction loop with gain ($\overline{K}$) computed in the other loop (labeled GAIN LOOP). The difference between the expected value $H\hat{x}$ of the observation z (based on the current estimate $\hat{x}$ of the state vector) and the observed value is used in correcting the estimate $\hat{x}$. Errors in $\hat{x}$ will be corrected by this loop, so long as the gain is correct. This applies to errors in $\hat{x}$ introduced by roundoff as well as those due to noise and a priori estimation errors. Therefore, roundoff errors in the estimation loop are compensated by the feedback mechanism, so long as the loop gain is correct. That gain is computed in the other loop.

7.3.2.3 No Feedback in the Gain Loop This is the loop in which the Riccati equation is solved for the covariance matrix of estimation uncertainty (P), and the Kalman gain is computed as an intermediate result. It is not stabilized by feedback, the way that the estimation loop is stabilized. There is no external reference for correcting the "estimate" of P. Consequently, there is no way of detecting and correcting the effects of roundoff errors. They propagate and accumulate unchecked. This loop also includes many more roundoff operations than the estimation loop, as evidenced by the greater number of matrix multiplies ($\otimes$) in the loop. The computations involved in evaluating the filter gains are, therefore, more suspect as sources of roundoff error propagation in this "conventional" implementation of the Kalman filter. It has been shown by Potter [13] that the gain loop, by itself,

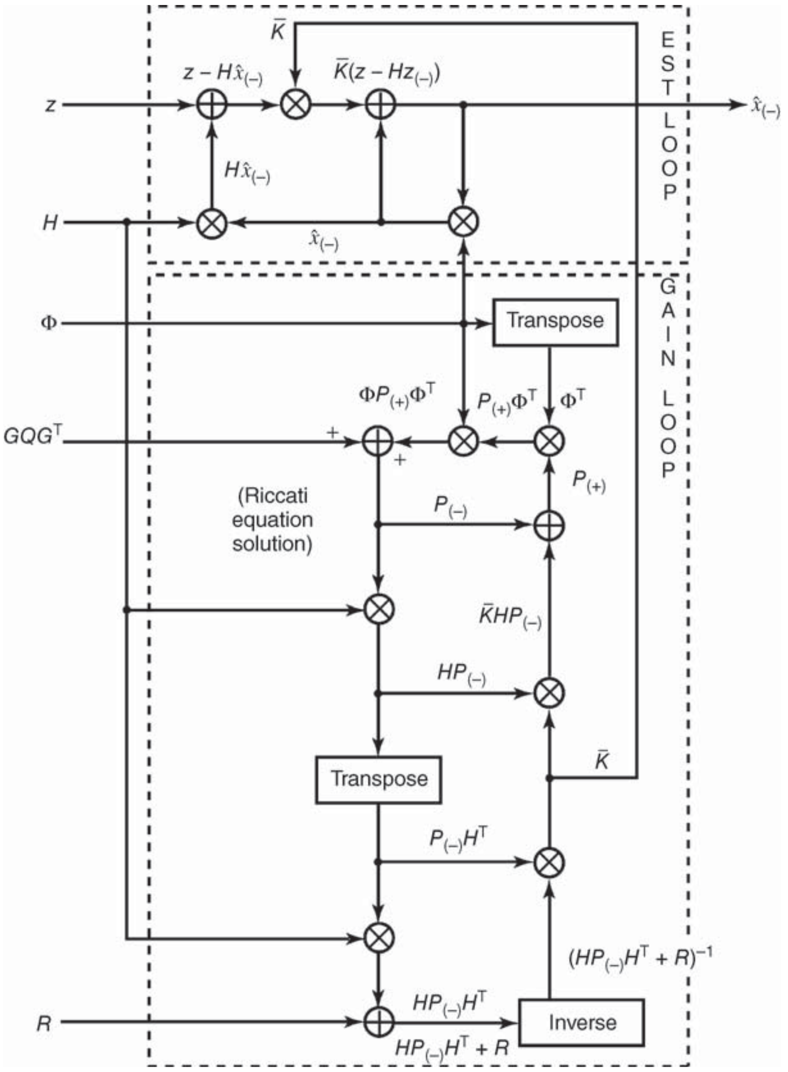

Figure 7.2 Kalman filter data flow.

is not unstable. However, even bounded errors in the computed value of P may momentarily destabilize the estimation loop.

Example 7.4 An illustration of the effects that negative characteristic values of the computed covariance matrix P can have on the estimation errors is shown as follows.

Roundoff errors can cause the computed value of P to have a negative characteristic value. The Riccati equation is stable, and the problem will eventually rectify itself. However, the effect on the actual estimation error can be a more serious problem.

Because P is a factor in the Kalman gain $\overline{K}$, a negative characteristic value of P can cause the gain in the prediction error feedback loop to have the wrong sign. However, in this transient condition, the estimation loop is momentarily destabilized. In this illustration, the estimate $\hat{x}$ converges toward the true value x until the gain changes sign. Then the error diverges momentarily. The gain computations may eventually recover with the correct sign, but the accumulated error due to divergence is not accounted for in the gain computations. The gain is not as big as it should be, and convergence is slower than it should be.

7.3.2.4 Numerical Analysis Because the a priori value of P is the one used in computing the Kalman gain, it suffices to consider just the error propagation of that value. It is convenient, as well, to consider the roundoff error propagation for $x_{(-)}$.

A first-order roundoff error propagation model is of the form

$$\delta x_{k+1(-)} = f_1(\delta x_{k(-)}, \delta P_{k(-)}) + \Delta x_{k+1}, \tag{7.4}$$

$$\delta P_{k+1(-)} = f_2(\delta P_{k(-)}) + \Delta P_{k+1(-)}, \tag{7.5}$$

where the δ term refers to the accumulated error and the Δ term refers to the added roundoff errors on each recursion step. This model ignores higher order terms in the error variables. The forms of the appropriate error propagation functions are given in Table 7.1. Error equations for the Kalman gain are also given, although the errors in $\overline{K}_k$ depend only on the errors in x and P—they are not propagated independently. These error propagation function values are from the paper by Verhaegen and Van Dooren [12]. (Many of these results have also appeared in earlier publications.) These expressions represent the first-order error in the updated a prior variables on the $(k+1)$th temporal epoch in terms of the first-order errors in the kth temporal epoch and the errors added in the update process.

TABLE 7.1 First-Order Error Propagation Models

Roundoff Error in Filter Variable	Error Model (by Filter Type)	
	Conventional Implementation	Square-Root Covariance
$\delta x_{k+1(-)}$	$A_1[\delta x_{k(-)} + \delta P_{k(-)} A_2 (z - H x_{k(-)})] + \Delta x_{k+1}$	
$\delta \overline{K}_k$	$A_1 \delta P_{k(-)}$	
$\delta P_{k+1(-)}$	$A_1 \delta P_{k(-)} A_1^{\mathrm{T}} + \Delta P_{k+1}$ $+\Phi(\delta P_{k(-)} - \delta P_{k(-)}^{\mathrm{T}})\Phi^{\mathrm{T}}$ $-\Phi(\delta P_{k(-)} - \delta P_{k(-)}^{\mathrm{T}})A_1^{\mathrm{T}}$	$A_1 \delta P_{k(-)} A_1^{\mathrm{T}}$ $+\Delta P_{k+1}$

Notes: $A_1 = \Phi - \overline{K}_k H$; $A_2 = H^{\mathrm{T}}[H P_k H^{\mathrm{T}} + R]^{-1}$.

Roundoff Error Propagation Table 7.1 compares two filter implementation types, in terms of their first-order error propagation characteristics. One implementation type is called *conventional*. That corresponds to the straightforward implementation of the equations as they were originally derived in previous chapters, excluding the "Joseph-stabilized" implementation mentioned in Chapter 5. The other type is called *square root*, the type of implementation presented in this chapter. A further breakdown of these implementation types will be defined in later sections.

Propagation of Antisymmetry Errors Note the two terms in Table 7.1 involving the antisymmetry error $\delta P_{k(-)} - \delta P^{\mathrm{T}}_{k(-)}$ in the covariance matrix P, which tends to confirm in theory what had been discovered in practice. Early computers had very little memory capacity, and programmers had learned to save time and memory by computing only the unique parts of symmetric matrix expressions such as $\Phi P \Phi^{\mathrm{T}}, HPH^{\mathrm{T}}, HPH^{\mathrm{T}} + R$, or $(HPH^{\mathrm{T}} + R)^{-1}$. To their surprise and delight, this was also found to improve error propagation. It has also been found to be beneficial in MATLAB implementations to maintain symmetry of P by evaluating the MATLAB expression `P=.5*(P+P')` on every cycle of the Riccati equation.

Added Roundoff Error The roundoff error (Δ) that is added on each cycle of the Kalman filter is considered in Table 7.2. The tabulated formulas are upper bounds on these random errors.

TABLE 7.2 Upper Bounds on Added Roundoff Errors

	Upper Bounds (by Filter Type)	
Norm of Roundoff Errors	Conventional Implementation	Square-Root Covariance
$\lvert\Delta x_{k+1(-)}\rvert$	$\varepsilon_1(\lvert A_1\rvert\lvert x_{k(-)}\rvert + \lvert\overline{K}_k\rvert\lvert z_k\rvert) + \lvert\Delta\overline{K}_k\rvert(\lvert H\rvert\lvert x_{k(-)}\rvert + \lvert z_k\rvert)$	$\varepsilon_4(\lvert A_1\rvert\lvert x_{k(-)}\rvert + \lvert\overline{K}_k\rvert\lvert z_k\rvert) + \lvert\Delta\overline{K}_k\rvert(\lvert H\rvert\lvert x_{k(-)}\rvert + \lvert z_k\rvert)$
$\lvert\Delta\overline{K}_k\rvert$	$\varepsilon_2\kappa^2(R^*)\lvert\overline{K}_k\rvert$	$\varepsilon_5\kappa(R^*)[\lambda_m^{-1}(R^*)\lvert C_{p(k+1)}\rvert + \lvert\overline{K}_k C_{R^*}\rvert + \lvert A_3\rvert/\lambda_1(R^*)]$
$\lvert\Delta P_{k+1(-)}\rvert$	$\varepsilon_3\kappa^2(R^*)\lvert P_{k+1(-)}\rvert$	$\dfrac{\varepsilon_6[1+\kappa(R^*)]\lvert P_{k+1}\rvert\lvert A_3\rvert}{\lvert C_{p(k+1)}\rvert}$

Notes: $\varepsilon_1, \ldots, \varepsilon_6$ are constant multiples of ε, the unit roundoff error; $A_1 = \Phi - \overline{K}_k H; A_3 = [(\overline{K}_k C_{R^*})\lvert C_{p(k+1)}]; R^* = HP_{k(-)}H^{\mathrm{T}} + R; R^* = C_{R^*}C^{\mathrm{T}}_{R^*}$ (triangular Cholesky decomposition); $P_{k+1(-)} = C_{p(k+1)}C^{\mathrm{T}}_{p(k+1)}$ (triangular Cholesky decomposition); $\lambda_1(R^*) \geq \lambda_2(R^*) \geq \cdots \geq \lambda_\ell(R^*) \geq 0$ are the characteristic values of R^*; $\kappa(R^*) = \lambda_1(R^*)/\lambda_\ell(R^*)$ is the condition number of R^*.

The important points which these tables demonstrate are the following:

1. These expressions show the same first-order error propagation in the state update errors for both filter types (covariance and square-root forms). These include terms coupling the errors in the covariance matrix into the state estimate and gain.
2. The error propagation expression for the conventional Kalman filter includes aforementioned terms proportional to the antisymmetric part of P. One must consider the effects of roundoff errors added in the computation of x, $\overline{K}$, and P as well as those propagated from the previous temporal epoch. In this case, Verhaegen and Van Dooren have obtained upper bounds on the norms of the added errors Δx, $\Delta\overline{K}$, and ΔP, as shown in Table 7.2. These upper bounds give a crude approximation of the dependence of roundoff error propagation on the characteristics of the unit roundoff error (ε) and the parameters of the Kalman filter model. Here, the bounds on the added state estimation error are similar for the two filter types, but the bounds on the added covariance error ΔP are better for the square-root filter. (The factor is something like the condition number of the matrix E.) In this case, one cannot relate the difference in performance to such factors as asymmetry of P.

The efficacy of various implementation methods for reducing the effects of roundoff errors have also been studied experimentally for some applications. The paper by Verhaegen and Van Dooren [12] includes results of this type as well as numerical analyses of other implementations (information filters and Chandrasekhar filters). Similar comparisons of square-root filters with conventional Kalman filters (and Joseph-stabilized filters) have been made by Thornton and Bierman [14].

7.3.3 Examples of Filter Divergence

The following simple examples show how roundoff errors can cause the Kalman filter results to diverge from their expected values.

Example 7.5 (Roundoff Errors Due to Large a Priori Uncertainty) If users have very little confidence in the *a priori* estimate for a Kalman filter, they tend to make the initial covariance of estimation uncertainty very large. This has its limitations, however.

Consider the scalar parameter estimation problem ($\Phi = I, Q = 0, \ell = n = 1$) in which the initial variance of estimation uncertainty $P_0 \gg R$, the variance of measurement uncertainty. Suppose that the measurement sensitivity $H = 1$ and that P_0 is so much greater than R so that, in the floating-point machine precision, the result of adding R to P_0—with roundoff—is P_0. That is, $R < \varepsilon P_0$. In that case, the values computed in the Kalman filter calculations will be as shown in the table below:

		Value	
Observation Number	Expression	Exact	Rounded
1	P_0H^{T}	P_0	P_0
1	HP_0H^{T}	P_0	P_0
1	$HP_0H^{\mathrm{T}} + R$	$P_0 + R$	P_0
1	$\overline{K}_1 = P_0H^{\mathrm{T}}(HP_0H^{\mathrm{T}} + R)^{-1}$	$\dfrac{P_0}{P_0 + R}$	1
1	$P_1 = P_0 - \overline{K}_1HP_0$	$\dfrac{P_0R}{P_0 + R}$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$
k	$\overline{K}_k = P_{k-1}H^{\mathrm{T}}(HP_{k-1}H^{\mathrm{T}} + R)^{-1}$	$\dfrac{P_0}{kP_0 + R}$	0
	$P_k = P_{k-1} - \overline{K}_kHP_{k-1}$	$\dfrac{P_0R}{kP_0 + R}$	0

The rounded value of the calculated variance of estimation uncertainty is zero after the first measurement update and remains zero thereafter. As a result, the calculated value of the Kalman gain is also zero after the first update. The exact (roundoff-free) value of the Kalman gain is $\approx 1/k$, where k is the observation number. After 10 observations,

1. the calculated variance of estimation uncertainty is zero;
2. the actual variance of estimation uncertainty is $P_0R/(P_0 + R) \approx R$ (the value after the first observation and after which the computed Kalman gains were zeroed), and
3. the *theoretical* variance in the exact case (no roundoff) would have been $P_0R/(10P_0 + R) \approx \frac{1}{10}R$.

The ill-conditioning in this example is due to the misscaling between the a priori state estimation uncertainty and the measurement uncertainty.

7.4 FACTORIZATION METHODS FOR “SQUARE-ROOT” FILTERING

7.4.1 Background

The historical background and notational ambiguity of “square-root” filtering were discussed in Chapter 1.

7.4.2 Types of Cholesky Factors

We need to distinguish three different types of Cholesky factors for symmetric (and therefore square), positive-definite matrices:

1. *Cholesky factors* are triangular (and therefore square) with positive diagonal entries. They may be either
   (a) lower triangular (i.e., with zeros above the diagonal), or
   (b) upper triangular (i.e., with zeros below the diagonal).
2. *Generalized Cholesky factors* C of a matrix P are not required to be triangular—or even square—but their symmetric product must satisfy the equation $CC^{\mathrm{T}} = P$. All Cholesky factors are also generalized Cholesky factors, but not all generalized Cholesky factors are Cholesky factors. Some of the methods used in "square-root" filtering are for converting generalized Cholesky factors into true Cholesky factors.
3. *Modified Cholesky factorization* has the form

$$P = UDU^{\mathrm{T}},$$

   where U is a *unit triangular matrix* (i.e., with ones on its diagonal) and D is a diagonal matrix with positive diagonal entries. Modified Cholesky factors are neither ordinary Cholesky factors nor generalized Cholesky factors.

Square-root filtering reformulates the Riccati equation to replace the covariance matrix with its "square root" (actually, a Cholesky factor—either straight, generalized, or modified). Either way, this makes a significant difference in the numerical stability of the solution, even though it becomes necessary to "square" the result (actually, using the symmetric product) to recover the covariance matrix for analysis.

However, there are many ways to implement "square root" formulations of the Kalman filter, depending on which type of factoring is used. There are also many ways such matrix factoring methods can be used in different parts of the Kalman filter.

7.4.3 Overview of Matrix Factorization Tricks

See Chapter 1 for distinctions between matrix decompositions, factorizations, and triangularizations. A further distinction between *decomposition* and *factorization* is made by Dongarra et al. [15], who use the term *factorization* to refer to an arithmetic process for performing a product decomposition of a matrix in which not all factors are preserved. These include *triangularization,* used to denote QR factorizations (in the sense of Dongarra et al.) involving a triangular factor that is preserved and an orthogonal factor that is not preserved.

7.4.3.1 Applications to Kalman Filtering The more numerically stable implementations of the Kalman filter use one or more of the following techniques to solve the associated Riccati equation:

1. Factoring the *covariance matrix of state estimation uncertainty* P (the dependent variable of the Riccati equation) into Cholesky factors (triangular

with positive diagonal elements), generalized Cholesky factors, or modified Cholesky factors (unit triangular and diagonal factors).

2. Factoring the *covariance matrix of measurement noise* R to reduce the computational complexity of the observational update implementation. These methods effectively "decorrelate" the components of the transformed measurement noise vector.
3. Taking the symmetric *matrix square roots of elementary matrices*. A symmetric elementary matrix has the form $I - \sigma v v^{\mathrm{T}}$, where I is the $n \times n$ identity matrix, σ is a scalar, and v is an n-vector. The symmetric square root of an elementary matrix is also an elementary matrix with the same v but a different value for σ.
4. Factoring *general matrices* as products of triangular and orthogonal matrices. Two general methods are used in Kalman filtering:
   (a) *Triangularization (QR decomposition)* methods were originally developed for more numerically stable solutions of systems of linear equations. They factor a matrix into the product of an orthogonal matrix Q and a triangular matrix R. In the application to Kalman filtering, only the triangular factor is needed. We will call the QR decomposition triangularization, because Q and R already have special meanings in Kalman filtering. The two triangularization methods used in Kalman filtering are
      i. *Givens rotations* [164] triangularize a matrix by operating on one element at a time. (A *modified Givens method* due to Gentleman [163] generates *diagonal* and *unit triangular* factors.)
      ii. *Householder transformations* [21] triangularize a matrix by operating on one row or column at a time.
   (b) *Gram–Schmidt orthonormalization* is another general method for factoring a general matrix into a product of an orthogonal matrix and a triangular matrix. Usually, the triangular factor is not saved. In the application to Kalman filtering, only the triangular factor is saved.
5. *Rank 1 modification algorithms*. A "rank 1 modification" of a symmetric positive-definite $n \times n$ matrix M has the form $M \pm v v^{\mathrm{T}}$, where v is an n-vector (and therefore has matrix rank equal to 1). The algorithms compute a Cholesky factor of the modification $M \pm v v^{\mathrm{T}}$, given v and a Cholesky factor of M.
6. *Block matrix factorizations* of matrix expressions in the Riccati equation. The general approach uses two different factorizations to represent the two sides of an equation, such as

$$
\begin{aligned}
CC^{\mathrm{T}} &= AA^{\mathrm{T}} + BB^{\mathrm{T}} \\
&= \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} A^{\mathrm{T}} \\ B^{\mathrm{T}} \end{bmatrix}.
\end{aligned}
$$

The alternative generalized Cholesky factors C and $[A\ B]$ must then be related by orthogonal transformations (triangularizations). A *QR* decomposition of

[A B] will yield a corresponding solution of the Riccati equation in terms of a Cholesky factor of the covariance matrix.

In the example used above, [A B] would be called a “1×2” block-partitioned matrix, because there are one row and two columns of blocks (matrices) in the partitioning. Different block dimensions are used to solve different problems:

1. The *discrete-time temporal update* equation is solved in “square-root” form by using alternative 1×2 block-partitioned generalized Cholesky factors.
2. The *observational update* equation is solved in “square-root” form by using alternative 2×2 block-partitioned generalized Cholesky factors and modified Cholesky factors representing the observational update equation.
3. The *combined temporal/observational update* equations are solved in “square-root” form by using alternative 2×3 block-partitioned generalized Cholesky factors of the combined temporal and observational update equations.

The different implementations of the Kalman filter based on these approaches are presented in Sections 7.5.2–7.7.2 and 7.7. They make use of the general numerical procedures presented in Sections 7.4.4–7.4.7.

7.4.4 Cholesky Decomposition Methods and Applications

7.4.4.1 Symmetric Products and Generalized Cholesky Factors The product of a matrix C with its own transpose in the form $CC^{\mathrm{T}} = M$ is called the *symmetric product* of C and C is called the *generalized Cholesky factor* of M (Section B.6 of Appendix B on the Wiley website). Strictly speaking, a generalized Cholesky factor is not a matrix square root, although the terms are often used interchangeably in the literature. (A matrix square root S of M is a solution of $M = SS = S^2$, without the transpose.)

All symmetric nonnegative-definite matrices (such as covariance matrices) have generalized Cholesky factors, but the generalized Cholesky factor of a given symmetric nonnegative-definite matrix is *not unique*. For any *orthogonal matrix* $\mathcal{J}$ (i.e., such that $\mathcal{J}\mathcal{J}^{\mathrm{T}} = I$), the product $\Gamma = C\mathcal{J}$ satisfies the equation

$$\Gamma\Gamma^{\mathrm{T}} = C\mathcal{J}\mathcal{J}^{\mathrm{T}}C^{\mathrm{T}} = CC^{\mathrm{T}} = M.$$

That is, $\Gamma = C\mathcal{J}$ is also a generalized Cholesky factor of M. Transformations of one generalized Cholesky factor into another are important for alternative Kalman filter implementations.

Applications to Kalman Filtering Cholesky decomposition methods produce triangular matrix factors (Cholesky factors), and the sparseness of these factors can be exploited in the implementation of the Kalman filter equations. These methods are used for the following purposes:

1. in the decomposition of covariance matrices (P, R, and Q) for implementation of square-root filters;
2. in "decorrelating" measurement errors between components of vector-valued measurements, so that the components may be processed sequentially as independent scalar-valued measurements (Section 7.4.4.3);
3. as part of a numerically stable method for computing matrix expressions containing the factor $(HPH^{\mathrm{T}} + R)^{-1}$ in the conventional form of the Kalman filter (this matrix inversion can be obviated by the decorrelation methods, however); and
4. in Monte Carlo analysis of Kalman filters by simulation, in which generalized Cholesky factors are used for generating independent random sequences of vectors with pre-specified means and covariance matrices (see Section 4.3.9).

7.4.4.2 Decomposition Algorithms

Symmetric Square Roots The *singular-value decomposition* of a symmetric positive-definite matrix P has the form

$$P = \mathcal{E}\Lambda\mathcal{E}^{\mathrm{T}} \tag{7.6}$$

$\mathcal{E}$ = an orthogonal matrix,

Λ = a diagonal matrix with positive diagonal values $\lambda_j > 0$.

The "eigenstructure" of P is characterized by the columns e_j of $\mathcal{E}$ and the diagonal elements λ_j of Λ. The e_j are unit vectors parallel to the principal axes of the equiprobability hyperellipse of the Gaussian distribution with covariance P, and the λ_j are the corresponding variances of the probability distribution along these axes. Furthermore, the values e_j and λ_j define the eigenvector–eigenvalue decomposition of P as

$$P = \sum_{j=1}^{n} \lambda_j \, e_j e_j^{\mathrm{T}}. \tag{7.7}$$

They also define a symmetric generalized Cholesky factor of P as

$$C \overset{\mathrm{def}}{=} \sum_{j=1}^{n} \sqrt{\lambda_j} \, e_j e_j^{\mathrm{T}}, \tag{7.8}$$

such that $C^{\mathrm{T}}C = CC^{\mathrm{T}} = P$. The first of these equalities follows from the symmetry of C (i.e., $C^{\mathrm{T}} = C$), and the second from

$$CC^{\mathrm{T}} = \left[\sum_{j=1}^{n} \sqrt{\lambda_j} \, e_j e_j^{\mathrm{T}}\right] \left[\sum_{k=1}^{n} \sqrt{\lambda_k} \, e_k e_k^{\mathrm{T}}\right]^{\mathrm{T}} \tag{7.9}$$

$$= \sum_{j=1}^{n} \sum_{k=1}^{n} \sqrt{\lambda_j}\,\sqrt{\lambda_k}\, e_j \{e_j^{\mathrm{T}}\, e_k\} e_k^{\mathrm{T}} \tag{7.10}$$

$$\{e_j^{\mathrm{T}}\, e_k\} = \begin{cases} 0, & j \neq k \\ 1, & j = k \end{cases} \tag{7.11}$$

$$CC^{\mathrm{T}} = \sum_{j=1}^{n} \left[\sqrt{\lambda_j}\right]^2 e_j e_j^{\mathrm{T}} \tag{7.12}$$

$$CC^{\mathrm{T}} = \sum_{j=1}^{n} \lambda_j\, e_j e_j^{\mathrm{T}} \tag{7.13}$$

$$= P. \tag{7.14}$$

The MATLAB function `svd` performs singular-value decompositions of general matrices. If P is symmetric positive definite, the MATLAB command “`[E,Lambda,X] = svd(P);`” returns the eigenvectors e_j of P as the columns of `E`, and the eigenvalues λ_j of P as the diagonal elements of `Lambda`. The symmetric positive-definite generalized Cholesky factor C of a symmetric positive-definite matrix P can then be produced by MATLAB commands

```
[E,Lambda,X] = svd(P);
sqrtD    = sqrt(Lambda);
C        = E*sqrtD*E';
```

Symmetric matrix square roots are not that popular in Kalman filtering, because there are much simpler algorithms for computing triangular Cholesky factors, but symmetric generalized Cholesky factors may be useful in sample-based methods for nonlinear filtering.

Triangular Cholesky Factors Recall that the *main diagonal* of an $n \times m$ matrix C is the set of elements $\{C_{ii} | 1 \leq i \leq \min(m, n)\}$ and that C is called *triangular* if the elements on one side of its main diagonal are zero. The matrix is called *upper triangular* if its nonzero elements are on and above its main diagonal and *lower triangular* if they are on or below the main diagonal.

A Cholesky decomposition algorithm is a procedure for calculating the elements of a triangular Cholesky factor of a symmetric, nonnegative-definite matrix. It solves the Cholesky decomposition equation $P = CC^{\mathrm{T}}$ for a triangular matrix C, given the matrix P, as illustrated in the following example.

Example 7.6 (**3×3 example**) Consider the 3×3 example for finding a lower triangular Cholesky factor $P = CC^{\mathrm{T}}$ for symmetric P:

$$\begin{bmatrix} p_{11} & p_{21} & p_{31} \\ p_{21} & p_{22} & p_{32} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & 0 & 0 \\ c_{21} & c_{22} & 0 \\ c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} c_{11} & 0 & 0 \\ c_{21} & c_{22} & 0 \\ c_{31} & c_{32} & c_{33} \end{bmatrix}^{\mathrm{T}}$$
$$= \begin{bmatrix} c_{11}^2 & c_{11}c_{21} & c_{11}c_{31} \\ c_{11}c_{21} & c_{21}^2 + c_{22}^2 & c_{21}c_{31} + c_{22}c_{32} \\ c_{11}c_{31} & c_{21}c_{31} + c_{22}c_{32} & c_{31}^2 + c_{32}^2 + c_{33}^2 \end{bmatrix}.$$

The corresponding matrix elements of the left- and right-hand sides of the last matrix equation can be equated as nine scalar equations. However, due to symmetry, only six of these are independent. The six scalar equations can be solved in sequence, making use of previous results. The following solution order steps down the rows and across the columns:

Six Independent Scalar Equations	Solutions Using Prior Results
$p_{11} = c_{11}^2$	$c_{11} = \sqrt{p_{11}}$
$p_{21} = c_{11}c_{21}$	$c_{21} = p_{21}/c_{11}$
$p_{22} = c_{21}^2 + c_{22}^2$	$c_{22} = \sqrt{p_{22} - c_{21}^2}$
$p_{31} = c_{11}c_{31}$	$c_{31} = p_{31}/c_{11}$
$p_{32} = c_{21}c_{31} + c_{22}c_{32}$	$c_{32} = (p_{32} - c_{21}c_{31})/c_{22}$
$p_{33} = c_{31}^2 + c_{32}^2 + c_{33}^2$	$c_{33} = \sqrt{p_{33} - c_{31}^2 - c_{32}^2}$

A solution can also be obtained by stepping across the rows and then down the rows, in the order $c_{11}, c_{21}, c_{31}, c_{22}, c_{32}, c_{33}$.

The general solutions can be put in the form of algorithms looping through the rows and columns of C and using prior results. The example above suggests two algorithmic solutions, one looping in row–column order and one looping in column–row order. There is also the choice of whether the solution C should be lower triangular or upper triangular.

Algorithmic solutions are given in Table 7.3. The one on the left can be implemented as `C = chol(M)`, using the built-in MATLAB function `chol`. The one in the right column is implemented in the m-file `utchol.m`.

Programming Note: MATLAB automatically assigns the value zero to all the unassigned matrix locations. This would not be necessary if subsequent processes treat the resulting Cholesky factor matrix C as triangular and do not bother to add or multiply the zero elements.

7.4.4.3 Modified Cholesky (UD) Decomposition Algorithms

Unit Triangular Matrices An upper triangular matrix U is called *unit upper triangular* if its diagonal elements are all 1 (unity). Similarly, a lower triangular matrix L is called *unit lower triangular* if all of its diagonal elements are unity.

TABLE 7.3 Cholesky Decomposition Algorithms

Given an $m \times m$ symmetric positive-definite matrix M, computes a triangular matrix C such that $M = CC^{\mathrm{T}}$.

Lower Triangular Result

```
for j = 1: m,
 for i = 1 : j,
  sigma = M (i, j);
  for k = 1 : j - 1,
   sigma = sigma - C (i, k)*C (j, k);
  end;
  if i == j
   C (i, j) = sqrt (sigma);
  else
   C (i, j) = sigma/C (j, j)
  end;
 end;
end;
```

Upper Triangular Result

```
for j = m: -1:1,
  for i = j : -1:1,
   sigma = M (i, j);
   for k = j + 1 : m,
    sigma = sigma - C (i, k)*C (j, k);
   end;
   if i == j
    C (i, j) = sqrt (sigma);
   else
    C (i, j) = sigma/C (j, j)
   end;
  end;
end;
```

Computational complexity: $\frac{1}{6}m(m-1)(m+4)$ flops $+ m\sqrt{}$.

UD Decomposition Algorithm The *modified* Cholesky decomposition of a symmetric positive-definite matrix M is a decomposition into products $M = UDU^{\mathrm{T}}$ such that U is unit upper triangular and D is diagonal. It is also called *UD decomposition*.

A procedure for implementing UD decomposition is presented in Table 7.4. This algorithm is implemented in the m-file `modchol.m`. It takes M as input and returns U and D as output. The decomposition can also be implemented in place, overwriting the input array containing M with D (on the diagonal of the array containing M) and U (in the strictly upper triangular part of the array containing M). This algorithm is only slightly different from the upper triangular Cholesky decomposition algorithm presented in Table 7.3. The big difference is that the modified Cholesky decomposition does not require taking scalar square roots.

7.4.4.4 Decorrelating Measurement Noise The decomposition methods developed for factoring the covariance matrix of estimation uncertainty may also be applied to the covariance matrix of measurement uncertainty, R. This operation redefines the measurement vector (via a linear transform of its components) such that its measurement errors are uncorrelated from component to component. That is, the new covariance matrix of measurement uncertainty is a *diagonal* matrix. In that case, the components of the redefined measurement vector can be processed serially as uncorrelated scalar measurements. The reduction in the computational complexity[3] of the Kalman filter from this approach will be covered in Section 7.7.1.

[3]The methodology used for determining the computational complexities of algorithms in this chapter is presented in Section 7.4.4.7.

TABLE 7.4 ***UD* Decomposition Algorithm**

Given M, a symmetric, positive-definite $m \times m$ matrix, U and D, modified Cholesky factors of M, are computed, such that U is a unit upper triangular matrix, D is a diagonal matrix, and $M = UDU^{\mathrm{T}}$.

```
for j = m : -1:1,
  for i = j : -1:1,
    sigma = M (i, j);
    for k = j + 1 : m,
      sigma = sigma - U (i, k) *D (k, k) *U (j, k);
    end;
    if i == j
      D (j, j) = sigma;
      U (j, j) = 1;
    else
      U (i, j) = sigma/D (j, j);
    end;
  end;
end;
```

Computational complexity: $\frac{1}{6}m(m-1)(m+4)$ flops.

Suppose, for example, that

$$z = Hx + \xi \tag{7.15}$$

is an observation with measurement sensitivity matrix H and noise ξ that is correlated from component to component of ξ. That is, the covariance matrix

$$\mathrm{E}\langle \xi \xi^{\mathrm{T}} \rangle = R \tag{7.16}$$

is not a diagonal matrix. Then the scalar components of z cannot be processed serially as scalar observations with statistically independent measurement errors.

However, R can always be factored in the form

$$R = UDU^{\mathrm{T}}, \tag{7.17}$$

where D is a diagonal matrix and U is an upper triangular matrix. Unit triangular matrices have some useful properties:

- The determinant of a unit triangular matrix is 1. Unit triangular matrices are, therefore, always *nonsingular*. In particular, they always have a matrix inverse.
- The inverse of a unit triangular matrix is a unit triangular matrix. The inverse of a unit upper triangular matrix is unit upper triangular, and the inverse of a unit lower triangular matrix is a unit lower triangular matrix.

It is not necessary to compute U^{-1} to perform measurement decorrelation, but it is useful for pedagogical purposes to use U^{-1} to redefine the measurement as

$$\begin{aligned} \acute{z} &= U^{-1}z && (7.18)\\ &= U^{-1}(Hx+\xi) && (7.19)\\ &= (U^{-1}H)x + (U^{-1}\xi) && (7.20)\\ &= \acute{H}x + \acute{\xi}. && (7.21) \end{aligned}$$

That is, this “new” measurement $\acute{z}$ has measurement sensitivity matrix $\acute{H} = U^{-1}H$ and observation error $\acute{\xi} = U^{-1}\xi$. The covariance matrix R' of the observation error $\acute{\xi}$ will be the expected value

$$\begin{aligned} R' &= \mathrm{E}\langle \acute{\xi}\acute{\xi}^{\mathrm{T}} \rangle && (7.22)\\ &= \mathrm{E}\langle (U^{-1}\xi)(U^{-1}\xi)^{\mathrm{T}} \rangle && (7.23)\\ &= \mathrm{E}\langle U^{-1}\xi\xi^{\mathrm{T}}U^{\mathrm{T}-1} \rangle && (7.24)\\ &= U^{-1}\mathrm{E}\langle \xi\xi^{\mathrm{T}} \rangle U^{\mathrm{T}-1} && (7.25)\\ &= U^{-1}RU^{\mathrm{T}-1} && (7.26)\\ &= U^{-1}(UDU^{\mathrm{T}})U^{\mathrm{T}-1} && (7.27)\\ &= D. && (7.28) \end{aligned}$$

That is, this redefined measurement has uncorrelated components of its measurement errors, which is what was needed for serializing the processing of the components of the new vector-valued measurement.

In order to decorrelate the measurement errors, one must solve the unit upper triangular system of equations

$$\begin{aligned} U\acute{z} &= z && (7.29)\\ U\acute{H} &= H && (7.30) \end{aligned}$$

for $\acute{z}$ and $\acute{H}$, given z, H, and U. As noted previously, *it is not necessary to invert* U to solve for $\acute{z}$ and $\acute{H}$.

Solving Unit Triangular Systems It was mentioned above that it is not necessary to invert U to decorrelate measurement errors. In fact, it is only necessary to solve equations of the form $UX = Y$, where U is a unit triangular matrix and X and Y have conformable dimensions. The objective is to solve for X, given Y. It can be done by what is called *back substitution*. The algorithms listed in Table 7.5 perform the solutions by back substitution. The one on the right overwrites Y with $U^{-1}Y$. This feature is useful when several procedures are composed into one special-purpose procedure, such as the decorrelation of vector-valued measurements.

TABLE 7.5 Unit Upper Triangular System Solution

Input: $U, m \times m$ unit upper triangular matrix; $Y, m \times p$ matrix Output: $X := U^{-1}Y$	Input: $U, m \times m$ unit upper triangular matrix; $Y, m \times p$ matrix Output: $Y := U^{-1}Y$ (In-Place)

`for j = 1 : p,`
` for i = m: -1:1,`
`  X (i,j) = Y (i,j);`
`  for k = i + 1 : m,`
`  X (i,j) = X(i,j) - U(i,k)*X(k,j);`
`  end;`
` end;`
`end;`
	`for j = 1 : p,`
	` for i = m: -1:1,`
	`  for k = i + 1 : m,`
	`   Y (i,j) = Y (i,j) - U (i,k)*Y (k,j);`
	`  end;`
	` end;`
	`end;`

Computational complexity: $pm(m-1)/2$ flops.	

Specialization for Measurement Decorrelation A complete procedure for measurement decorrelation is listed in Table 7.6. It performs the *UD* decomposition and upper triangular system solution in place (overwriting H with $U^{-1}H$ and z with $U^{-1}z$), after decomposing R as $R = UDU^{\mathrm{T}}$ in place (overwriting the diagonal of R with $\acute{R} = D$ and overwriting the strictly upper triangular part of R with the strictly upper triangular part of U^{-1}).

7.4.4.5 Symmetric Positive-Definite System Solution Cholesky decomposition provides an efficient and numerically stable method for solving equations of the form $AX = Y$ when A is a symmetric, positive-definite matrix. The modified Cholesky decomposition is even better, because it avoids taking scalar square roots. It is the recommended method for forming the term $[HPH^{\mathrm{T}} + R]^{-1}H$ in the conventional Kalman filter without explicitly inverting a matrix. That is, if one decomposes $HPH^{\mathrm{T}} + R$ as UDU^{T}, then

$$[UDU^{\mathrm{T}}][HPH^{\mathrm{T}} + R]^{-1}H = H. \tag{7.31}$$

It then suffices to solve

$$UDU^{\mathrm{T}}X = H \tag{7.32}$$

for X. This can be done by solving the three problems

$$UX_{[1]} = H \text{ for } X_{[1]}, \tag{7.33}$$

$$DX_{[2]} = X_{[1]} \text{ for } X_{[2]}, \tag{7.34}$$

$$U^{\mathrm{T}}X = X_{[2]} \text{ for } X. \tag{7.35}$$

TABLE 7.6 Measurement Decorrelation Procedure

The vector-valued measurement $z = Hx + v$, with correlated components of the measurement error $E(vv^{\mathrm{T}}) = R$, is transformed to the measurement $\acute{z} = \acute{H}x + \acute{v}$ with uncorrelated components of the measurement error $\acute{v} : E\langle \acute{v}\acute{v}^{\mathrm{T}} \rangle = \mathrm{D}$, a diagonal matrix, by overwriting H with $\acute{H} = U^{-1}H$ and z with $\acute{z} = U^{-1}z$, after decomposing R to UDU^{T}, overwriting the diagonal of R with D.

Symbol	Definition
R	Input: $\ell \times \ell$ covariance matrix of measurement uncertainty Output: D (on diagonal), U (above diagonal)
H	Input: $\ell \times n$ measurement sensitivity matrix Output: overwritten with $\acute{H} = U^{-1}H$
z	Input: measurement ℓ-vector Output: overwritten with $\acute{z} = U^{-1}z$

Procedure:
1. Perform UD decomposition of R in place.
2. Solve $U\acute{z} = z$ and $U\acute{H} = H$ in place.

Computational complexity: $\frac{1}{6}\ell(\ell - 1)(\ell + 4) + \frac{1}{2}\ell(\ell - 1)(n + 1)$ flops.

The first of these is a unit upper triangular system, which was solved in the previous subsection. The second is a system of independent scalar equations, which has a simple solution. The last is a unit lower triangular system, which can be solved by "forward substitution"—a simple modification of back substitution. The computational complexity of this method is m^2p, where m is the row and column dimension of A and p is the column dimension of X and Y.

7.4.4.6 Transforming Covariance Matrices to Information Matrices The information matrix is the inverse of the covariance matrix—and vice versa. Although matrix inversion is generally to be avoided if possible, it is just not possible to avoid it forever. This is one of those problems that require it.

The inversion is not possible unless one of the matrices (either P or Y) is positive definite, in which case both will be positive definite and they will have the same condition number. If they are sufficiently well conditioned, they can be inverted in place by UD decomposition, followed by inversion and recomposition in place. The in-place UD decomposition procedure is listed in Table 7.4. A procedure for inverting the result in place is shown in Table 7.7. A matrix inversion procedure using these two is outlined in Table 7.8. It should be used with caution, however.

7.4.4.7 Computational Complexities Using the general methods outlined in References 16 and 17, one can derive the complexity formulas shown in Table 7.9 for methods using generalized Cholesky factors.

TABLE 7.7 Unit Upper Triangular Matrix Inversion

Input/output: U, an $m \times m$ unit upper triangular matrix (U is overwritten with U^{-1})

```
for i = m : -1:1,
  for j = m : -1 : i + 1,
    U (i, j) = -U (i, j);
    for k = i + 1 : j - 1,
       U (i, j) = U (i, j) - U (i, k)*U (k, j);
    end;
  end;
end;
```

Computational complexity: $m(m-1)(m-2)/6$ flops.

7.4.5 Kalman Implementation with Decorrelation

It was pointed out by Kaminski [18] that the computational efficiency of the conventional Kalman observational update implementation can be improved by processing the components of vector-valued observations sequentially using the error decorrelation algorithm in Table 7.6, if necessary. The computational savings with the measurement decorrelation approach can be evaluated by comparing the rough operations counts of the two approaches using the operations counts for the sequential approach given in Table 7.10. One must multiply by ℓ, the number of operations required for the implementation of the scalar observational update equations, and add the number of operations required for performing the decorrelation.

The computational advantage of the decorrelation approach is

$$\frac{1}{3}\ell^3 - \frac{1}{2}\ell^2 + \frac{7}{6}\ell - \ell n + 2\ell^2 n + \ell n^2 \text{ flops.}$$

That is, it requires that many fewer flops to decorrelate vector-valued measurements and process the components serially.

7.4.6 Symmetric Square Roots of Elementary Matrices

7.4.6.1 Elementary Matrices An elementary matrix is a matrix of the form $I - s\mathbf{v}\mathbf{w}^{\mathrm{T}}$, where I is an identity matrix, s is a scalar, and **v, w** are column vectors of the same row dimension as I. Elementary matrices have the property that their products are also elementary matrices. Their squares are also elementary matrices, with the same vector values (**v, w**) but with different scalar values (s).

TABLE 7.8 Symmetric Positive-Definite Matrix Inversion Procedure

Inverts a symmetric positive-definite matrix in place

Symbol	Description
M	Input: $m \times m$ symmetric positive-definite matrix Output: M is overwritten with M^{-1}
Procedure:	1. Perform UD decomposition of M in place. 2. Invert U in place (in the M-array). 3. Invert D in place: for $i = 1 : m, M(i,i) = 1/M(i,i)$; end; 4. Recompose $M^{-1} = (U^{\mathrm{T}}D^{-1})U^{-1}$ in place:

```
for j=m:-1:1,
  for i=j:-1:1,
    if i==j
      s = M(i,i);
    else
      s = M(i,i)*M(i,j);
    end;
    for k=1:i-1,
      s = s + M(k,i)*M(k,k)*M(k,j);
    end;
    M(i,j) = s;
    M(j,i) = s;
  end;
end;
```

Computational complexity: $(m-1)(2m+5)/6$ flops

7.4.6.2 Symmetric Elementary Matrices An elementary matrix is symmetric if $v = w$. The squares of such matrices have the same format:

$$(I - \sigma \mathbf{v}\mathbf{v}^{\mathrm{T}})^2 = (I - \sigma \mathbf{v}\mathbf{v}^{\mathrm{T}})(I - \sigma \mathbf{v}\mathbf{v}^{\mathrm{T}}) \tag{7.36}$$

$$= I - 2\sigma \mathbf{v}\mathbf{v}^{\mathrm{T}} + \sigma^2|\mathbf{v}|^2\mathbf{v}\mathbf{v}^{\mathrm{T}} \tag{7.37}$$

$$= I - (2\sigma - \sigma^2|\mathbf{v}|^2)\mathbf{v}\mathbf{v}^{\mathrm{T}} \tag{7.38}$$

$$= I - s\mathbf{v}\mathbf{v}^{\mathrm{T}} \tag{7.39}$$

$$s = (2\sigma - \sigma^2|\mathbf{v}|^2). \tag{7.40}$$

7.4.6.3 Symmetric Square Root of a Symmetric Elementary Matrix One can also invert the last equation above and take the square root of the symmetric elementary matrix $(I - s\mathbf{v}\mathbf{v}^{\mathrm{T}})$. This is done by solving the scalar quadratic equation

$$s = 2\sigma - \sigma^2|\mathbf{v}|^2, \tag{7.41}$$

$$\sigma^2|\mathbf{v}|^2 - 2\sigma + s = 0 \tag{7.42}$$

TABLE 7.9 Computational Complexity Formulas

Cholesky decomposition of an $m \times m$ matrix:

$$\mathcal{C}_{\text{Cholesky}} = \sum_{j=1}^{m}\left[m - j + \sum_{i=j}^{m}(m-j)\right] = \frac{1}{3}m^3 + \frac{1}{2}m^2 - \frac{5}{6}m$$

UD decomposition of an $m \times m$ matrix:

$$\mathcal{C}_{UD} = \sum_{j=1}^{m}\left[m - j + \sum_{i=j}^{m}2(m-j)\right] = \frac{2}{3}m^3 + \frac{1}{2}m^2 - \frac{7}{6}m$$

Inversion of an $m \times m$ unit triangular matrix:

$$\mathcal{C}_{\text{UTINV}} = \sum_{i=1}^{m-1}\sum_{j=i+1}^{m}(j-i-1) = \frac{1}{6}m^3 - \frac{1}{2}m^2 + \frac{1}{3}m$$

Measurement decorrelation ($\ell \times n$ H-matrix):

$$\mathcal{C}_{\text{DeCorr}} = \mathcal{C}_{UD} + \sum_{i=1}^{\ell-1}\sum_{k=i+1}^{\ell}(n+1) = \frac{2}{3}\ell^3 + \ell^2 - \frac{5}{3}\ell + \frac{1}{2}\ell^2 n - \frac{1}{2}\ell n$$

Inversion of an $m \times m$ covariance matrix:

$$\mathcal{C}_{\text{COVINV}} = \mathcal{C}_{UD} + \mathcal{C}_{\text{UTINV}} + m + \sum_{i=1}^{m}[i(i-1)(m-i+1)] = m^3 + \frac{1}{2}m^2 + \frac{1}{2}m$$

TABLE 7.10 Operations for Sequential Processing of Measurements

Operation	Flops
$H \times P_{(-)}$	n^2
$H \times [HP_{(-)}]^{\mathrm{T}} + R$	n
$\{H[HP_{(-)}]^T + R\}^{-1}$	1
$\{H[HP_{(-)}]^{\mathrm{T}} + R\}^{-1} \times [HP_{(-)}]$	n
$P_{(-)} - [HP_{(-)}] \times \{H[HP_{(-)}]^{\mathrm{T}} + R\}^{-1}[HP_{(-)}]$	$\frac{1}{2}n^2 + \frac{1}{2}n$
Total (per component) $\times\ell$ components	$(\frac{1}{2}n^2 + \frac{5}{2}n + 1) \times \ell$
+decorrelation complexity	$\frac{2}{3}\ell^3 + \ell^2 - \frac{5}{3}\ell + \frac{1}{2}\ell^2 n - \frac{1}{2}\ell n$
Total	$\frac{2}{3}\ell^3 + \ell^2 - \frac{2}{3}\ell + \frac{1}{2}\ell^2 n + 2\ell + n\frac{1}{2}\ell n^2$

to obtain the solution

$$(I - s\mathbf{v}\mathbf{v}^{\mathrm{T}})^{1/2} = I - \sigma\mathbf{v}\mathbf{v}^{\mathrm{T}}, \tag{7.43}$$

$$\sigma = \frac{1 + \sqrt{1 - s|\mathbf{v}|^2}}{|\mathbf{v}|^2}. \tag{7.44}$$

In order that this square root be a real matrix, it is necessary that the radicand

$$1 - s|\mathbf{v}|^2 \geq 0. \tag{7.45}$$

7.4.7 Triangularization Methods

7.4.7.1 Triangularization Methods for Least-Squares Problems These techniques were originally developed for solving least-squares problems. The overdetermined system

$$Ax = b$$

can be solved efficiently and relatively accurately by finding an orthogonal matrix T such that the product $B = TA$ is a triangular matrix. In that case, the solution to the triangular system of equations

$$Bx = Tb$$

can be solved by backward substitution.

***7.4.7.2 Triangularization (*QR** *Decomposition) of* **A** It is a theorem of linear algebra that any general matrix A can be represented as a product[4]

$$A = C_{k+1(-)}\mathrm{T} \tag{7.46}$$

[4]This is the so-called “QR” decomposition in disguise. It is customarily represented as $A = QR$ (whence the name), where Q is orthogonal and R is triangular. However, as mentioned earlier, we have already

of a triangular matrix $C_{k+1(-)}$ and an orthogonal matrix T. This type of decomposition is called *QR decomposition* or *triangularization*. By means of this triangularization, the symmetric matrix product factorization

$$\begin{aligned} P_{k+1(-)} &= AA^{\mathrm{T}} &(7.47)\\ &= [C_{k+1(-)}T][C_{k+1(-)}T]^{\mathrm{T}} &(7.48)\\ &= C_{k+1(-)}TT^{\mathrm{T}}C_{k+1(-)}^{\mathrm{T}} &(7.49)\\ &= C_{k+1(-)}(TT^{\mathrm{T}})C_{k+1(-)}^{\mathrm{T}} &(7.50)\\ &= C_{k+1(-)}C_{k+1(-)}^{\mathrm{T}} &(7.51)\end{aligned}$$

also defines a triangular Cholesky decomposition of $C_{k+1(-)}$ of $P_{k+1(-)}$. This is the basis for performing temporal updates of Cholesky factors of P.

7.4.7.3 Uses of Triangularization in Kalman Filtering Matrix triangularization methods were originally developed for solving least-squares problems. They are used in Kalman filtering for

- temporal updates of generalized Cholesky factors of the covariance matrix of estimation uncertainty, as described above;
- observational updates of generalized Cholesky factors of the estimation information matrix, as described in Section 7.7.3.5; and
- combined updates (observational and temporal) of generalized Cholesky factors of the covariance matrix of estimation uncertainty, as described in Section 7.7.2.

A modified Givens rotation due to Gentleman [19] is used for the temporal updating of modified Cholesky factors of the covariance matrix.

In these applications, as in most least-squares applications, the orthogonal matrix factor is unimportant. The resulting triangular factor is the intended result, and numerically stable methods have been developed for computing it.

7.4.7.4 Triangularization Algorithms Two of the more stable methods for matrix triangularization are presented in the following subsections. These methods are based on orthogonal transformations (matrices) that, when applied to (multiplied by) general matrices, reduce them to triangular form. Both were published in the same year (1958). Both define the requisite transformation as a product of "elementary" orthogonal transformations:

$$T = T_1T_2T_3\cdots T_m. \tag{7.52}$$

committed the symbols Q and R to play other roles in this book. In this instance of the QR decomposition, it has the transposed form $A^{\mathrm{T}} = T^{\mathrm{T}}C_{k+1(-)}^{\mathrm{T}}$, where T^{T} is the stand-in for the original Q (the orthogonal factor) and $C_{k+1(-)}^{\mathrm{T}}$ is the stand-in for the original R (the triangular factor).

These elementary transformations are either *Givens rotations* or *Householder reflections*. In each case, triangularization is achieved by zeroing of the nonzero elements on one side of the main diagonal. Givens rotations zero these elements one by one. Householder reflections zero entire subrows of elements (i.e., the part of a row left of the triangularization diagonal) on each application. The order in which such transformations may be applied must be constrained so that they do not “unzero” previously zeroed elements.

7.4.7.5 Triangularization by Givens Rotations This method for triangularization, due to Givens [20], uses a *plane rotation matrix* $T_{ij}(\theta)$ of the following form:

$$T_{ij}(\theta) = \begin{array}{c} \\ \\ \\ \\ j \\ \\ \\ \\ i \\ \\ \\ \\ \end{array} \begin{array}{ccccccccccc} & j & & i & & & & & & & \\ \end{array}$$

$$T_{ij}(\theta) = \begin{array}{c|ccccccccccc} & & j & & i & & & & & & & \\ & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ & \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ j & 0 & \cdots & 0 & \cos(\theta) & 0 & \cdots & 0 & \sin(\theta) & 0 & \cdots & 0 \\ & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ & \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots \\ & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 \\ i & 0 & \cdots & 0 & -\sin(\theta) & 0 & \cdots & 0 & \cos(\theta) & 0 & \cdots & 0 \\ & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ & \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots \\ & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 \end{array}. \tag{7.53}$$

It is also called a *Givens rotation matrix* or *Givens transformation matrix*. Except for the ith and jth rows and columns, the plane rotation matrix has the appearance of an identity matrix. When it is multiplied on the right-hand side of another matrix, it affects only the ith and jth columns of the matrix product. It rotates the ith and jth elements of a row or column vector, as shown in Figure 7.3. It can be used to rotate one of the components all the way to zero, which is how it is used in triangularization.

Triangularization of a matrix A by Givens rotations is achieved by successive multiplications of A on one side by Givens rotation matrices, as illustrated by the following example.

Example 7.7 **(2×2 example)** Consider the problem of upper triangularizing the 2×3 symbolic matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \tag{7.54}$$

by multiplying with Givens rotation matrices on the right. The first product

$$\begin{aligned}\acute{A}(\theta) &= AT_{23}(\theta)\\ &= \begin{bmatrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\end{bmatrix}\begin{bmatrix} 1 & 0 & 0\\ 0 & \cos(\theta) & \sin(\theta)\\ 0 & -\sin(\theta) & \cos(\theta)\end{bmatrix}\\ &= \begin{bmatrix} a_{11} & a_{12}\cos(\theta) - a_{13}\sin(\theta) & a_{12}\sin(\theta) + a_{13}\cos(\theta)\\ a_{21} & \boxed{a_{22}\cos(\theta) - a_{23}\sin(\theta)} & a_{22}\sin(\theta) + a_{23}\cos(\theta)\end{bmatrix}.\end{aligned}$$

The framed element in the product will be zero if $a_{22}^2 + a_{23}^2 = 0$, and if $a_{22}^2 + a_{23}^2 > 0$, the values

$$\cos(\theta) = \frac{a_{23}}{\sqrt{a_{22}^2 + a_{23}^2}}, \quad \sin(\theta) = \frac{a_{22}}{\sqrt{a_{22}^2 + a_{23}^2}}$$

will force it to zero. The resulting matrix Á can be multiplied again on the right by the Givens rotation matrix $T_{13}(\acute{\theta})$ to yield yet a second intermediate matrix form

$$\begin{aligned}\breve{A}(\grave{\theta}) &= AT_{23}(\theta)T_{13}(\grave{\theta})\\ &= \begin{bmatrix} a_{11} & \acute{a}_{12} & \acute{a}_{13}\\ a_{21} & 0 & \acute{a}_{23}\end{bmatrix}\begin{bmatrix} \cos(\grave{\theta}) & & \sin(\grave{\theta})\\ 0 & 1 & 0\\ -\sin(\grave{\theta}) & 0 & \cos(\grave{\theta})\end{bmatrix}\\ &= \begin{bmatrix} \grave{a}_{11} & \acute{a}_{12} & \grave{a}_{13}\\ 0 & 0 & \grave{a}_{23}\end{bmatrix}\end{aligned}$$

for $\grave{\theta}$ such that

$$\cos(\grave{\theta}) = \frac{\acute{a}_{23}}{\sqrt{\acute{a}_{21}^2 + \acute{a}_{23}^2}}, \quad \sin(\grave{\theta}) = \frac{\acute{a}_{21}}{\sqrt{\acute{a}_{21}^2 + \acute{a}_{23}^2}}.$$

A third Givens rotation yields the final matrix form

$$\begin{aligned}\breve{A}(\grave{\theta}) &= AT_{23}(\theta)T_{13}(\grave{\theta})T_{12}(\grave{\theta})\\ &= \begin{bmatrix} \grave{a}_{11} & \acute{a}_{12} & \grave{a}_{13}\\ 0 & 0 & \grave{a}_{23}\end{bmatrix}\begin{bmatrix} \cos(\grave{\theta}) & \sin(\grave{\theta}) & 0\\ -\sin(\grave{\theta}) & \cos(\grave{\theta}) & 0\\ 0 & 0 & 1\end{bmatrix}\\ &= \begin{bmatrix} 0 & \breve{a}_{12} & \grave{a}_{13}\\ 0 & 0 & \grave{a}_{23}\end{bmatrix}\end{aligned}$$

for $\grave{\theta}$ such that

$$\cos(\grave{\theta}) = \frac{\acute{a}_{12}}{\sqrt{\grave{a}_{11}^2 + \acute{a}_{12}^2}}, \quad \sin(\grave{\theta}) = \frac{\grave{a}_{11}}{\sqrt{\grave{a}_{11}^2 + \acute{a}_{12}^2}}.$$

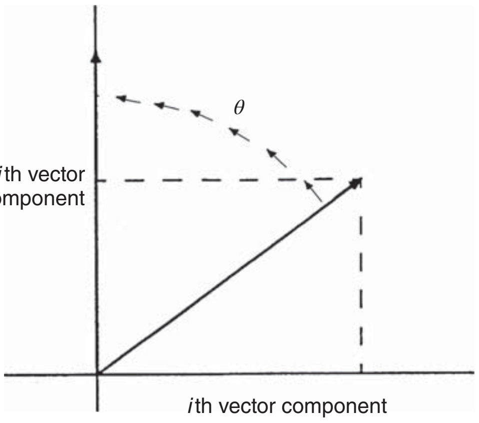

Figure 7.3 Component transformations by plane rotation.

The remaining nonzero part of this final result is an upper triangular submatrix right adjusted within the original array dimensions.

The order in which successive Givens rotation matrices are applied is constrained to avoid “unzeroing” elements of the matrix that have already been zeroed by previous Givens rotations. Figure 7.4 shows the constraints that guarantee such noninterference. If we suppose that the element to be annihilated (designated by x in the figure) is in the ith column and kth row and the corresponding diagonal element of the soon-to-be triangular matrix is in the jth column, then it is sufficient if the elements below the kth rows in those two columns have already been annihilated by Givens rotations. The reason for this is simple: the Givens rotations can only form linear combinations of row elements in those two columns. If those row elements are already zero, then any linear combination of them will also be zero. The result: no effect.

A Givens Triangularization Algorithm The method used in the previous example can be generalized to an algorithm for upper triangularization of an $n \times (n + r)$ matrix, as listed in the following.

```
Input: A, an - by (n + r) matrix

Output: A is overwritten by an upper triangular matrix C,
        right adjusted in the array, such that output value of CC′
        equals input value of AA′.

for i = n : -1:1,
    for j = 1 : r + i,
      rho = sqrt (A (i, r + i)^2+A (i, j)^2);
      s = A (i, j)/ rho;
      c = A (i, r + i)/ rho;
      for k = 1 : i,
                   x = c*A (k, j) - s*A (k, r + i);
        A (k, r + i) = s*A (k, j) + c*A (k, r + i);
           A (k, j) = x;
```

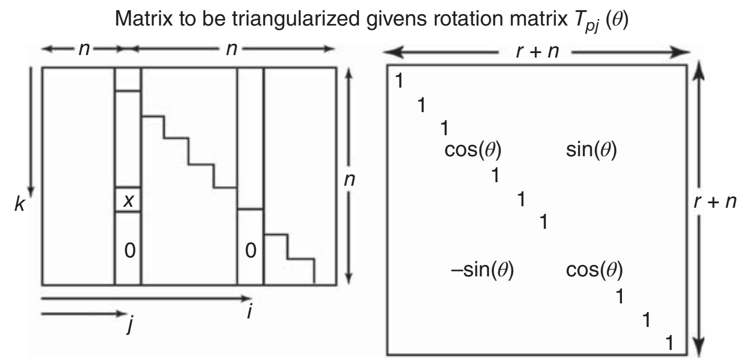
Matrix to be triangularized givens rotation matrix $T_{pj}(\theta)$
n
n
k
x
0
0
n
j
i
r + n
r + n
1
1
1
cos(θ)
sin(θ)
1
1
1
−sin(θ)
cos(θ)
1
1
1

Figure 7.4 Constraints on Givens triangularization order.

```
        end;
    end;
end;
```

In this particular form of the algorithm, the outermost loop (the loop with index i in the listing) zeros elements of A one row at a time. An analogous algorithm can be designed in which the outermost loop is by columns.

7.4.7.6 Triangularization by Householder Reflections This method of triangularization was discovered by Householder [21]. It uses an elementary matrix of the form

$$T(\xi) = I - \frac{2}{\xi^{\mathrm{T}}\xi}\xi\xi^{\mathrm{T}}, \tag{7.55}$$

where ξ is a column vector and I is the identity matrix of the same dimension. This particular form of the elementary matrix is called a *Householder reflection, Householder transformation*, or *Householder matrix.*

Note that Householder transformation matrices are always *symmetric*. They are also *orthogonal*, for

$$T(\xi)T^{\mathrm{T}}(\xi) = \left(I - \frac{2}{\xi^{\mathrm{T}}\xi}\xi\xi^{\mathrm{T}}\right)\left(I - \frac{2}{\xi^{\mathrm{T}}\xi}\xi\xi^{\mathrm{T}}\right) \tag{7.56}$$

$$= I - \frac{4}{\xi^{\mathrm{T}}\xi}\xi\xi^{\mathrm{T}} + \frac{4}{(\xi^{\mathrm{T}}\xi)^2}\xi(\xi^{\mathrm{T}}\xi)\xi^{\mathrm{T}} \tag{7.57}$$

$$= I. \tag{7.58}$$

They are called *reflections* because they transform any matrix x into its "mirror reflection" in the plane (or hyperplane[5]) normal to the vector ξ, as illustrated in Figure 7.5

[5]The dimension of the hyperplane normal to the vector ξ will be one less than that of the space containing ξ. When, as in the illustration, ξ is a three-dimensional vector (i.e., the space containing ξ is three-dimensional), the hyperplane normal to ξ is a two-dimensional plane.

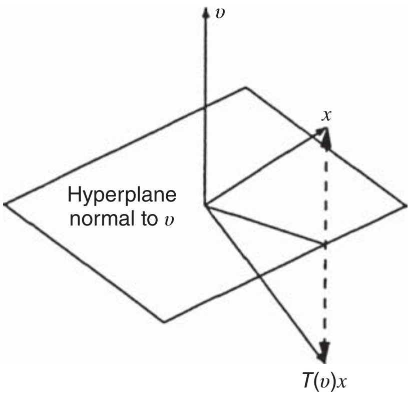

Figure 7.5 Householder reflection of a vector x.

(for three-dimensional ξ and x). By choosing the proper mirror plane, one can place the reflected vector $T(\xi)x$ along any direction whatsoever, including parallel to any coordinate axis.

Example 7.8 (Householder Reflection Along One Coordinate Axis) Let x be any n-dimensional row vector, and let

$$e_k \overset{\text{def}}{=} \begin{array}{ccccccccc} & & & & & k & & & \\ [0 & 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0] \end{array}$$

be the kth row of the $n \times n$ identity matrix. If the vector ξ of the Householder reflection $T(\xi)$ is defined as

$$\xi = x^{\mathrm{T}} + \alpha e_k^{\mathrm{T}},$$

where α is a scalar, then the inner products

$$\xi^{\mathrm{T}}\xi = |x|^2 + 2\alpha x_k + \alpha^2,$$
$$x\xi = |x|^2 + \alpha x_k,$$

where x_k is the kth component of x. The Householder reflection $xT(\xi)$ of x will be

$$\begin{aligned} xT(\xi) &= x\left(I - \frac{2}{\xi^{\mathrm{T}}\xi}\xi\xi^{\mathrm{T}}\right) \\ &= x\left(I - \frac{2}{x^{\mathrm{T}}x + 2\alpha x_k + \alpha^2}\xi\xi^{\mathrm{T}}\right) \\ &= x - \frac{2x\xi}{|x|^2 + 2\alpha x_k + \alpha^2}\xi^{\mathrm{T}} \end{aligned}$$

$$
\begin{aligned}
&= x - \frac{2(|x|^2 + \alpha x_k)}{|x|^2 + 2\alpha x_k + \alpha^2}(x + \alpha e_k) \\
&= \left[1 - \frac{2(|x|^2 + \alpha x_k)}{|x|^2 + 2\alpha x_k + \alpha^2}\right] x - \left[\frac{2\alpha(|x|^2 + \alpha x_k)}{|x|^2 + 2\alpha x_k + \alpha^2}\right] e_k \\
&= \left[\frac{\alpha^2 - |x|^2}{|x|^2 + 2\alpha x_k + \alpha^2}\right] x - \left[\frac{2\alpha(|x|^2 + \alpha x_k)}{|x|^2 + 2\alpha x_k + \alpha^2}\right] e_k.
\end{aligned}
$$

Consequently, if one lets

$$\alpha = \mp |x|, \tag{7.59}$$

then

$$xT(\xi) = \pm |x| e_k. \tag{7.60}$$

That is, $xT(\xi)$ is parallel to the kth coordinate axis.

If, in the above example, x were the last row vector of an $n \times (n + r)$ matrix

$$M = \begin{bmatrix} Z \\ x \end{bmatrix},$$

and letting $k = 1$, then

$$MT(\xi) = \begin{bmatrix} ZT(\xi) \\ xT(\xi) \end{bmatrix} \tag{7.61}$$

$$= \begin{bmatrix} & & & ZT(\xi) & & \\ 0 & 0 & 0 & \dots & 0 & |x| \end{bmatrix}, \tag{7.62}$$

the first step in upper triangularizing a matrix by Householder reflections.

Upper Triangularization by Successive Householder Reflections A single Householder reflection can be used to zero all the elements to the left of the diagonal in an entire row of a matrix, as shown in Figure 7.6. In this case, the vector x to be operated upon by the Householder reflection is a row vector composed of the first k components of a row of a matrix. Consequently, the dimension of the Householder reflection matrix $T(\xi)$ need only be k, which may be strictly less than the number of columns in the matrix to be triangularized. The "undersized" Householder matrix is placed in the upper left corner of the transformation matrix, and the remaining diagonal block is filled with an (appropriately dimensioned) identity matrix I, such that the row dimension of the resulting transformation matrix equals the column dimension of the matrix to be triangularized. The resulting composite matrix will always be orthogonal, so long as $T(\xi)$ is orthogonal.

There are two important features of this construction. The first is that the presence of the identity matrix has the effect of leaving the columns to the right of the kth column undisturbed by the transformation. The second is that the transformation does

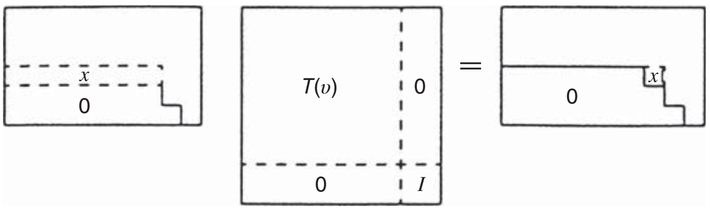

Figure 7.6 Zeroing one subrow with a Householder reflection.

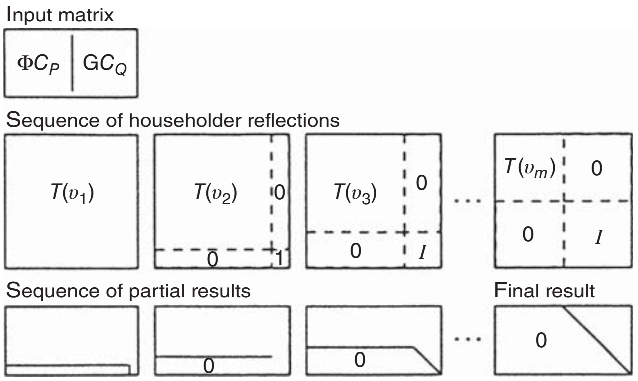
Input matrix
ΦC_P
GC_Q
Sequence of householder reflections
T(v_1)
T(v_2)
T(v_3)
T(v_m)
Sequence of partial results
Final result
0
I

Figure 7.7 Upper triangularization by Householder reflections.

not “unzero” previously zeroed rows below. Together, these features allow the matrix to be triangularized by the sequence of transformations shown in Figure 7.7. Note that the *size* of the Householder transformation shrinks with each step.

Householder Triangularization Algorithm The algorithm listed in the following performs upper triangularization of a rectangular $n \times (n + r)A$ matrix in place using a scratchpad $(n + r)$-vector ξ. The result is an upper triangular $n \times n$ matrix, right adjusted in the array A.

This algorithm includes a rescaling computation (involving the absolute value function abs) for better numerical stability. It is modified after the one given by Golub and Van Loan [17] for Householder triangularization. The modification is required for applying the Householder transformations from the right, rather than from the left, and for the particular form of the input matrix used in the Kalman filter implementation. Further specialization of this algorithm for Kalman filtering is presented later in Table 7.14.

```
for k = n: -1:1,
    sigma = 0;
    for j = 1 : r + k,
       sigma = sigma + A (k, j)^ 2;
    end;
    a = sqrt (sigma);
    sigma = 0;
    for j = 1 : r + k,
       if j == r + k
         v (j) = A (k, j) - a;
       else
         v (j) = A (k, j);
       end;
       sigma = sigma + v (j)^ 2;
    end;
    a = 2/sigma;
    for i = 1 : k,
    sigma = 0;
    for j = 1 : r + k,
       sigma = sigma + A (i, j)*v (j);
    end;
    b = a* sigma;
    for j = 1 : r + k,
       A (i, j) = A (i, j) - b* v (j);
    end;
    end;
  end;
```

7.5 "SQUARE-ROOT" AND *UD* FILTERS

The so-called "square-root" filtering uses a reformulation of the Riccati equations such that the dependent variable is a generalized Cholesky factor (or modified Cholesky factor) of the state estimation uncertainty matrix P. We present here just two of the many forms of square-root Kalman filtering, with other forms presented in the following section. The two selected forms of "square-root" filtering are

1. Carlson–Schmidt "square-root" filtering, which uses Cholesky factors of P, and
2. Bierman–Thornton *UD* filtering, which uses modified Cholesky factors of P.

These are perhaps the more favored implementation forms (after the conventional Kalman filter), because they have been used extensively and successfully on many problems that would be too poorly conditioned for conventional Kalman filter implementation. The *UD* filter, in particular, has been used successfully on problems with thousands of state variables.

This does not necessarily imply that these two methods are to be preferred above all others, however. It may be possible that the Morf–Kailath combined "square-root" filter (Section 7.7.2), for example, performs as well or better, but we are currently not aware of any comparable experience using that method.

7.5.1 Carlson–Schmidt "Square-Root" Filtering

This is a matched pair of algorithms for the observational and temporal update of the covariance matrix P in terms of its Cholesky factors. If the covariance matrices

R (measurement noise) and Q (dynamic process noise) are not diagonal matrices, the implementation also requires *UD* or Cholesky factorization of these matrices.

7.5.1.1 Carlson "Fast Triangular" Update This algorithm is implemented in the MATLAB m-file `carlson.m` on the accompanying diskette. The algorithm is due to Carlson [22]. It generates an upper triangular Cholesky factor W for the Potter factorization and has generally lower computational complexity than the Potter algorithm. It is a specialized and simplified form of an algorithm used by Agee and Turner [23] for Kalman filtering. It is a rank 1 modification algorithm, like the Potter algorithm, but it produces a triangular Cholesky factor. It can be derived from Problem 7.14.

In the case that $m = j$, the summation on the left-hand side of Equation 7.266 has but one term:

$$W_{ij}W_{ij} = \Delta_{ij} - \frac{\mathbf{v}_i\mathbf{v}_j}{R + \sum_{k=1}^{j}\mathbf{v}_k^2}, \tag{7.63}$$

which can be solved in terms of the elements of the upper triangular Cholesky factor W:

$$W_{ij} = \begin{cases} 0, & i > j, \\ \sqrt{\frac{R+\sum_{k=1}^{j-1}\mathbf{v}_k^2}{R+\sum_{k=1}^{j}\mathbf{v}_k^2}}, & i = j, \\ \frac{-\mathbf{v}_i\mathbf{v}_j}{\left(R+\sum_{k=1}^{j-1}\mathbf{v}_k^2\right)\left(R+\sum_{k=1}^{j}\mathbf{v}_k^2\right)}, & i < j. \end{cases} \tag{7.64}$$

Given the above formula for the elements of W, one can derive the formula for the elements of $C_{(+)} = C_{(-)}W$, the upper triangular Cholesky factor of the a posteriori covariance matrix of estimation uncertainty $P_{(+)}$, given $C_{(-)}$, the upper triangular Cholesky factor of the a priori covariance matrix $P_{(-)}$.

Because both C and W are upper triangular, the elements $C_{ik} = 0$ for $k < i$ and the elements $W_{kj} = 0$ for $k > j$. Consequently, for $1 \le i \le j \le n$, the element in the ith row and jth column of the matrix product $C_{(-)}W = C_{(+)}$ will be

$$C_{ij(+)} = \sum_{k=i}^{j} C_{ik(-)}W_{kj} + \text{terms with zero factors} \tag{7.65}$$

$$= C_{ij(-)}W_{jj} + \sum_{k=i}^{j-1} C_{ik(-)}W_{kj} \tag{7.66}$$

$$= C_{ij(-)}\sqrt{\frac{R + \sum_{k=1}^{j-1}\mathbf{v}_k^2}{R + \sum_{k=1}^{j}\mathbf{v}_k^2}} - \sum_{k=i}^{j-1} \frac{C_{ik(-)}\mathbf{v}_k\mathbf{v}_j}{\left(R + \sum_{k=1}^{j-1}\mathbf{v}_k^2\right)\left(R + \sum_{k=1}^{j}\mathbf{v}_k^2\right)} \tag{7.67}$$

$$= \left(R + \sum_{k=1}^{j} \mathbf{v}_k^2 \right)^{-1/2} \times \left[C_{ij(-)} \sqrt{R + \sum_{k=1}^{j-1} \mathbf{v}_k^2} - \frac{\mathbf{v}_j \sum_{k=i}^{j-1} C_{ik(-)} \mathbf{v}_k}{\left(R + \sum_{k=1}^{j-1} \mathbf{v}_k^2 \right)^{1/2}} \right]. \tag{7.68}$$

This is a general formula for the upper triangular a posteriori Cholesky factor of the covariance matrix of estimation uncertainty, in terms of the upper triangular a priori Cholesky factor $C_{(-)}$ and the vector $\mathbf{v} = C^{\mathrm{T}} H^{\mathrm{T}}$, where H is the measurement sensitivity matrix (a row vector). An algorithm implementing the formula is given in Table 7.11. This algorithm performs the complete observational update, including the update of the state estimate, in place. (Note that this algorithm forms the product $\mathbf{v} = C_{(-)}^{\mathrm{T}} H^{\mathrm{T}}$ internally, computing and using the components $\sigma = \mathbf{v}_j$ as needed, without storing the vector $\mathbf{v}$. It does store and use the vector $\mathbf{w} = C_{(-)}\mathbf{v}$, the unscaled Kalman gain, however.)

It is possible—and often desirable—to save array space by storing triangular matrices in singly subscripted arrays. An algorithm (in FORTRAN) for such an implementation of this algorithm is given in Carlson's original paper [22].

7.5.1.2 Schmidt Temporal Update

Nonsquare, Nontriangular Cholesky Factor of $P_{k(-)}$ If C_P is a generalized Cholesky factor of $P_{k-1(+)}$ and C_Q is a generalized Cholesky factor of Q_k, then the partitioned $n \times (n + q)$ matrix

$$A = [G_k C_Q \quad | \quad \Phi_k C_P] \tag{7.69}$$

has the $n \times n$ symmetric matrix product value

$$AA^{\mathrm{T}} = [G_{k-1} C_Q | \Phi_{k-1} C_P][G_{k-1} C_Q | \Phi_{k-1} C_P]^{\mathrm{T}} \tag{7.70}$$

$$= \Phi_{k-1} C_P C_P^{\mathrm{T}} \Phi_{k-1}^{\mathrm{T}} + G_{k-1} C_Q C_Q^{\mathrm{T}} G_{k-1}^{\mathrm{T}} \tag{7.71}$$

$$= \Phi_{k-1} P_{k-1(+)} \Phi_{k-1}^{\mathrm{T}} + G_{k-1} Q_{k-1} G_{k-1}^{\mathrm{T}} \tag{7.72}$$

$$= P_{k(-)}. \tag{7.73}$$

That is, A is a nonsquare, nontriangular generalized Cholesky factor of $P_{k(-)}$. If *only* it were square and triangular, it would be the kind of Cholesky factor we are looking for. It is not, but fortunately there are algorithmic procedures for modifying A to that format.

Programming Note: Computation of GC_Q and ΦC_P *in place. This should be attempted only if memory limitations demand it.* The product ΦC_P can be computed in place by overwriting Φ with the product ΦC_P. This is not desirable if Φ is constant, however. (It is possible to overwrite C_P with the product ΦC_P, but this requires the use of an additional n-vector of storage. This option is left as an exercise for the truly

TABLE 7.11 Carlson’s Fast Triangular Observational Update

Symbol	Definition
z	Value of scalar measurement
R	Variance of scalar measurement uncertainty
H	Scalar measurement sensitivity vector ($1 \times n$ matrix)
C	Cholesky factors of $P_{(-)}$ (input) and $P_{(+)}$ (output)
x	State estimates $x_{(-)}$ (input) and $x_{(+)}$ (output)
$\mathbf{w}$	Unscaled Kalman gain (internal)

```
alpha = R;
delta = z;
for j = 1 : n,
 delta = delta - H (j)*x (j);
 sigma = 0;
 for i = 1 : j,
  sigma = sigma + C (i, j)*H (i);
 end;
 beta = alpha;
 alpha = alpha + sigma^2;
 gamma = sqrt (alpha*beta);
 eta = beta/gamma;
 zeta = sigma/gamma;
 w (j) = 0;
 for i = 1 : j,
  tau = C (i, j);
  C (i, j) = eta*C (i, j) - zeta*w (i);
  w (i) = w (i) + tau*sigma;
 end;
end;
epsilon = delta/alpha;
for i = 1 : n,
 x (i) = x (i) + w (i)*epsilon;
end;
```

Computational complexity: $(2n^2 + 7n + 1)$ flops $+ n\sqrt{\ }$.

needy.) Similarly, the product GC_Q can be computed in place by overwriting G with the product GC_Q if $r \leq n$. Algorithms for doing the easier in-place operations when the Cholesky factors C_Q and C_P are *upper* triangular are shown in Table 7.12. Note that these have roughly half the computational complexities of the general matrix products.

Triangularization Using Givens Rotations The Givens triangularization algorithm is presented in Section 7.4.7. The specialization of this algorithm to use GC_Q and ΦC_P in place, without having to stuff them into a common array, is shown in Table 7.13.

TABLE 7.12 Algorithms Performing Matrix Products in Place

Overwriting Φ by ΦC_P	Overwriting G by GC_Q
`for j = n : -1:1,`	`for j = r : -1:1,`
` for i = 1 : n,`	` for i = 1 : n,`
`  sigma = 0;`	`  sigma = 0;`
`  for k = 1 : j,`	`  for k = 1 : j,`
`   sigma = sigma + Phi (i, k)*CP (k,j);`	`   sigma = sigma + G (i, k)*CQ (kj);`
`  end;`	`  end;`
`  Phi (i, j) = sigma;`	`  G (i, j) = sigma;`
` end;`	` end;`
`end;`	`end;`
Computational complexities	
$n^2(n+1)/2$	$nr(r+1)/2$

The computational complexity of Givens triangularization is greater than that of Householder triangularization, which is covered next. There are two attributes of the Givens method that might recommend it for certain applications, however:

1. The Givens triangularization procedure can exploit sparseness of the matrices to be triangularized. Because it zeros elements one at a time, it can skip over elements that are already zero. (The procedure may "unzero" some elements that are already zero in the process, however.) This will tend to decrease the computational complexity of the application.
2. The Givens method can be "parallelized" to exploit concurrent processing capabilities, if they are available. The parallelized Givens method has no data contention among concurrent processes—they are working with data in different parts of the matrix as it is being triangularized.

Schmidt Temporal Updates Using Householder Reflections The basic Householder triangularization algorithm (see Section 7.4.7.2) operates on a single $n \times (n + r)$ matrix. For the method of Dyer and McReynolds, this matrix is composed of two blocks containing the matrices GC_Q $(n \times r)$ and ΦP $(n \times n)$. The specialization of the Householder algorithm to use the matrices GC_Q and ΦP directly, without having to place them into a common array first, is described and listed in Table 7.14. Algorithms for computing GC_Q and ΦP in place were given in the previous subsection.

7.5.2 Bierman–Thornton *UD* Filtering

This is a pair of algorithms, including the Bierman algorithm for the observational update of the modified Cholesky factors U and D of the covariance matrix

TABLE 7.13 Temporal Update by Givens Rotations

Symbol	Description
A	Input: $G_k C_{Q_k}$, an $n \times r$ matrix. Output: A is overwritten by intermediate results.
B	Input: $\Phi_k C_{P_{k(+)}}$, an $n \times n$ matrix. Output: B is overwritten by the upper triangular matrix $C_{P_{k+1(-)}}$.

```
for i = n : -1:1,
  for j = 1 : r,
    rho = sqrt (B (i, i)^2 + A (i, j)^2);
    s = A (i, j)/rho;
    C = B (i, i)/rho;
    for k = 1 : i,
      tau = c*A (k, j) - s*B (k, i);
      B (k, i) = s*A (k, j) + c*B(k, i);
      A (k, j) = tau;
    end;
  end;
  for j = 1 : i - 1,
    rho = sqrt (B (i, i)^2 + B (i, j)^2);
    s = B (i, j)/rho;
    c = B (i, i)/rho;
    for k = 1 : i,
      tau = c*B (k, j) - s*B (k, i);
      B (k, i) = s*B (k, j) + c*B (k, i);
      B (k, j) = tau;
    end;
  end;
end;
```

Computational complexity:
$\frac{2}{3}n^2(2n + 3r + 6) + 6nr + \frac{8}{3}n$ flops $+ \frac{1}{2}n(n + 2r + 1)\sqrt{\quad}$.

$P = UDU^{\mathrm{T}}$, and the corresponding Thornton algorithm for the temporal update of U and D.

7.5.2.1 Bierman UD Observational Update Bierman's algorithm is one of the more stable implementations of the Kalman filter observational update. It is similar in form and computational complexity to the Carlson algorithm but avoids taking scalar square roots. (It has been called *square-root filtering without square roots*.) The algorithm was developed by the late Gerald J. Bierman (1941–1987), who made many useful contributions to optimal estimation theory, especially in implementation methods.

TABLE 7.14 Schmidt–Householder Temporal Update Algorithm

This modification of the Householder algorithm performs upper triangularization of the partitioned matrix $[\Phi C_{P_{(+)}}, GC_Q]$ by modifying $\Phi C_{P_{(+)}}$ and GC_Q in place using Householder transformations of the (effectively) partitioned matrix.

Input Variables	
Symbol	Description
A	$n \times n$ matrix $\Phi C_{P_{(+)}}$
B	$n \times r$ matrix GC_Q
Output Variables	
A	Array is overwritten by upper triangular result $C_{P_{(-)}}$ such that $C_{P_{(-)}} C^{\mathrm{T}}_{P(-)} = \Phi C_{P_{(+)}} C^{\mathrm{T}}_{P_{(+)}} \Phi^{\mathrm{T}} + GC_Q C_Q^{\mathrm{T}} G^{\mathrm{T}}$.
B	Array is zeroed during processing.
	Intermediate Variables
α, β, σ	Scalars
v	Scratchpad n-vector
w	Scratchpad $(n + r)$-vector

```
for k = n : -1:1,
    sigma = 0;
    for j = 1 : r,
         sigma = sigma + B (k,j)^2;
    end;
    for j = 1 : k,
        sigma = sigma + A (k,j)^2;
    end;
    alpha = sqrt (sigma);
    sigma = 0;
    for j = 1 : r,
        w (j) = B (k,j);
        sigma = sigma + w (j)^2;
    end;
    for j = 1 : k,
        if j == k
             v (j) = A (k,j) - alpha;
        else
             v (j) = A (k,j);
        end;
        sigma = sigma + v (j)^2;
    end;
    alpha = 2/sigma;
    for i = 1 : k,
        sigma = 0;
        for j = 1 : r,
            sigma = sigma + B (i,j)*w (j);
        end;
        for j = 1 : k,
            sigma = sigma + A (i,j)*v (j);
        end;
```

TABLE 7.14 (*Continued*)

```
        beta = alpha*sigma;
        for j = 1 : r,
            B (i,j) = B (i,j) - beta*w (j);
        end;
        for j = 1 : k,
             A (i,j) = A (i,j) - beta*v(j);
        end;
    end;
end;
```

Computational complexity:
$n^3 r + \frac{1}{2}(n+1)^2 r + 5 + \frac{1}{3}(2n+1)$ flops.

Partial UD Factorization of Covariance Equations In a manner similar to the case with Cholesky factors for scalar-valued measurements, the conventional form of the observational update of the covariance matrix

$$P_{(+)} = P_{(-)} - \frac{P_{(-)}H^{\mathrm{T}}HP_{(-)}}{R + HP_{(-)}H^{\mathrm{T}}}$$

can be partially factored in terms of *UD* factors:

$$P_{(-)} \overset{\text{def}}{=} U_{(-)}D_{(-)}U_{(-)}^{\mathrm{T}}, \tag{7.74}$$

$$P_{(+)} \overset{\text{def}}{=} U_{(+)}D_{(+)}U_{(+)}^{\mathrm{T}}, \tag{7.75}$$

$$\begin{aligned} U_{(+)}D_{(+)}U_{(+)}^{\mathrm{T}} &= U_{(-)}D_{(-)}U_{(-)}^{\mathrm{T}} \qquad (7.76)\\ &\quad - \frac{U_{(-)}D_{(-)}U_{(-)}^{\mathrm{T}}H^{\mathrm{T}}HU_{(-)}D_{(-)}}{R + HU_{(-)}D_{(-)}U_{(-)}^{\mathrm{T}}H^{\mathrm{T}}}U_{(-)}^{\mathrm{T}}\\ &= U_{(-)}D_{(-)}U_{(-)}^{\mathrm{T}} - \frac{U_{(-)}D_{(-)}\mathbf{v}\mathbf{v}^{\mathrm{T}}D_{(-)}U_{(-)}^{\mathrm{T}}}{R + \mathbf{v}^{\mathrm{T}}D_{(-)}\mathbf{v}} \qquad (7.77)\\ &= U_{(-)}\left[D_{(-)} - \frac{D_{(-)}\mathbf{v}\mathbf{v}^{\mathrm{T}}D_{(-)}}{R + \mathbf{v}^{\mathrm{T}}D_{(-)}\mathbf{v}}\right]U_{(-)}^{\mathrm{T}}, \qquad (7.78) \end{aligned}$$

where

$$\mathbf{v} = U_{(-)}^{\mathrm{T}}H^{\mathrm{T}} \tag{7.79}$$

is an n-vector and n is the dimension of the state vector.

Equation 7.78 contains the unfactored expression

$$D_{(-)} - D_{(-)}\mathbf{v}[\mathbf{v}^{\mathrm{T}}D_{(-)}\mathbf{v} + R]^{-1}\mathbf{v}^{\mathrm{T}}D_{(-)}.$$

If one were able to factor it with a unit triangular factor B in the form

$$D_{(-)} - D_{(-)}\mathbf{v}[\mathbf{v}^{\mathrm{T}}D_{(-)}\mathbf{v} + R]^{-1}\mathbf{v}^{\mathrm{T}}D_{(-)} = BD_{(+)}B^{\mathrm{T}}, \tag{7.80}$$

then $D_{(+)}$ would be the a posteriori D factor of P because the resulting equation

$$U_{(+)}D_{(+)}U_{(+)}^{\mathrm{T}} = U_{(-)}\{BD_{(+)}B^{\mathrm{T}}\}U_{(-)}^{\mathrm{T}} \tag{7.81}$$

$$= \{U_{(-)}B\}D_{(+)}\{U_{(-)}B\}^{\mathrm{T}} \tag{7.82}$$

can be solved for the a posteriori U factor as

$$U_{(+)} = U_{(-)}B. \tag{7.83}$$

Therefore, for the observational update of the UD factors of the covariance matrix P, it suffices to find a numerically stable and efficient method for the UD factorization of a matrix expression of the form $D - D\mathbf{v}[\mathbf{v}^{\mathrm{T}}D\mathbf{v} + R]^{-1}\mathbf{v}^{\mathrm{T}}D$, where $\mathbf{v} = U^{\mathrm{T}}H^{\mathrm{T}}$ is a column vector. Bierman [24] found such a solution, in terms of a rank 1 modification algorithm for modified Cholesky factors.

Bierman UD Factorization Derivations of the Bierman UD observational update can be found in Reference 24. It is implemented in the accompanying MATLAB m-file `bierman.m`.

An alternative algorithm implementing the Bierman UD observational update in place is given in Table 7.15. One can also store $\mathbf{w}$ over $\mathbf{v}$, to save memory requirements. It is possible to reduce the memory requirements even further by storing D on the diagonal of U, or, better yet, storing just the strictly upper triangular part of U in a singly subscripted array. These programming tricks do little to enhance readability of the algorithms, however. They are best avoided unless one is truly desperate for memory.

7.5.2.2 Thornton UD Temporal Update This UD factorization of the temporal update in the discrete-time Riccati equation is due to Thornton [25]. It is also called *modified weighted Gram–Schmidt* (MWGS) orthogonalization.[6] It uses a factorization algorithm due to Björck [26] that is actually quite different from the conventional Gram–Schmidt orthogonalization algorithm and more robust against roundoff errors. However, the algebraic properties of Gram–Schmidt orthogonalization are useful for deriving the factorization.

[6]Gram–Schmidt *orthonormalization* is a procedure for generating a set of "unit normal" vectors as linear combinations of a set of linearly independent vectors. That is, the resulting vectors are mutually orthogonal and have unit length. The procedure without the unit-length property is called *Gram–Schmidt orthogonalization*. These algorithmic methods were derived by Jorgen Pedersen Gram (1850–1916) and Erhard Schmidt (1876–1959).

TABLE 7.15 Bierman Observational Update

Symbol	Definition
z	Value of scalar measurement
R	Variance of scalar measurement uncertainty
H	Scalar measurement sensitivity vector ($1 \times n$ matrix)
U, D	*UD* factors of $P_{(-)}$ (input) and $P_{(+)}$ (output)
x	State estimates $x_{(-)}$ (input) and $x_{(+)}$ (output)
$\mathbf{v}$	scratchpad n-vector
$\mathbf{w}$	scratchpad n-vector

```
delta = z;
for j = 1 : n,
  delta = delta - H (j)*x (j);
  v (j) = H (j);
  for i = 1 : j - 1,
    v (j) = v (j) + U (i, j)*H (i);
  end;
end;
sigma = R;
for j = 1 : n,
  nu = v (j);
  v (j) = v (j)*D (j, j);
  w (j) = nu;
  for i = 1 : j - 1,
    tau = U (i, j)*nu;
    U (i, j) = U (i, j) - nu*w (i)/sigma;
    w (i) = w (i) + tau;
 end;
 D (j, j) = D (j, j)*sigma;
 sigma = sigma + nu*v (j);
 D (j, j) = D (j, j)*sigma;
end;
epsilon = delta/sigma;
for i = 1 : n,
  x (i) = x (i) + v (i)*epsilon;
end;
```

Computational complexity: $(2n^2 + 7n + 1)$ flops.

Gram–Schmidt orthogonalization is an algorithm for finding a set of n mutually orthogonal m-vectors $b_1, b_2, b_3, \ldots, b_m$ that are linear combinations of a set of n *linearly independent* m-vectors $a_1, a_2, a_3, \ldots, a_m$. That is, the inner products

$$b_i^{\mathrm{T}} b_j = \begin{cases} |b_i|^2 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases} \tag{7.84}$$

The Gram–Schmidt algorithm defines a unit lower[7] triangular $n \times n$ matrix L such that $A = BL$, where

$$A = \begin{bmatrix} a_1 & a_2 & a_3 & \dots & a_n \end{bmatrix} \tag{7.85}$$

$$= BL \tag{7.86}$$

$$= \begin{bmatrix} b_1 & b_2 & b_3 & \dots & b_n \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ \ell_{21} & 1 & 0 & \dots & 0 \\ \ell_{31} & l_{32} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \dots & 1 \end{bmatrix}, \tag{7.87}$$

where the vectors b_i are column vectors of B and the matrix product

$$B^{\mathrm{T}}B = \begin{bmatrix} |b_1|^2 & 0 & 0 & \dots & 0 \\ 0 & |b_2|^2 & 0 & \dots & 0 \\ 0 & 0 & |b_3|^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & |b_n|^2 \end{bmatrix} \tag{7.88}$$

$$= \operatorname{diag}_{1 \le i \le n}\{|b_i|^2\} \tag{7.89}$$

$$= D_\beta, \tag{7.90}$$

a diagonal matrix with positive diagonal values $\beta_i = |b_i|^2$.

Weighted Gram–Schmidt Orthogonalization The m-vectors x and y are said to be orthogonal *with respect to the weights* $w_1, w_2, w_3, \dots, w_m$ if the *weighted* inner product

$$\sum_{i=1}^{m} x_i w_i y_i = x^{\mathrm{T}} D_w y \tag{7.91}$$

$$= 0, \tag{7.92}$$

where the diagonal *weighting matrix*

$$D_w = \operatorname{diag}_{1 \le i \le n}\{w_i\}. \tag{7.93}$$

[7]In its original form, the algorithm produced a unit upper triangular matrix U by processing the a_i in the order $i = 1, 2, 3, \dots, n$. However, if the order is reversed, the resulting coefficient matrix will be lower triangular and the resulting vectors b_i will still be mutually orthogonal.

The Gram–Schmidt orthogonalization procedure can be extended to include orthogonality of the column vectors b_i and b_j with respect to the weighting matrix D_w:

$$b_i^{\mathrm{T}} D_w b_j = \begin{cases} \beta_i > 0 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases} \tag{7.94}$$

The resulting weighted Gram–Schmidt orthogonalization of a set of n linearly independent m-vectors $a_1, a_2, a_3, \dots, a_m$ with respect to a weighting matrix D_w defines a unit lower triangular $n \times n$ matrix L_w such that the product $AL_w = B_w$ and

$$B_w^{\mathrm{T}} D_w B_w = \mathrm{diag}_{1 \le i \le n}\{\beta_i\}, \tag{7.95}$$

where $D_w = I$ for conventional orthogonalization.

Modified Weighted Gram–Schmidt Orthogonalization The standard Gram–Schmidt orthogonalization algorithms are not reliably stable numerical processes when the column vectors of A are close to being linearly dependent or the weighting matrix has a large condition number. An alternative algorithm due to Björck has better overall numerical stability. Although L is not an important result for the orthogonalization problem (B is), its inverse turns out to be more useful for the *UD* filtering problem.

The *Thornton temporal update for UD factors* uses triangularization of the Q matrix (if it is not already diagonal) in the form $Q = GD_QG^{\mathrm{T}}$, where D_Q is a diagonal matrix. If we let the matrices

$$A = \begin{bmatrix} U_{k-1(+)}^{\mathrm{T}} \Phi_k^{\mathrm{T}} \\ G_k^{\mathrm{T}} \end{bmatrix}, \tag{7.96}$$

$$D_w = \begin{bmatrix} D_{k-1(+)} & 0 \\ 0 & D_{Q_k} \end{bmatrix}, \tag{7.97}$$

then the MWGS orthogonalization procedure will produce a unit lower triangular $n \times n$ matrix L^{-1} and a diagonal matrix D_β such that

$$A = BL, \tag{7.98}$$

$$L^{\mathrm{T}} D_\beta L = L^{\mathrm{T}} B^{\mathrm{T}} D_w BL \tag{7.99}$$

$$= (BL)^{\mathrm{T}} D_w BL \tag{7.100}$$

$$= A^{\mathrm{T}} D_w A \tag{7.101}$$

$$= \begin{bmatrix} \Phi_{k-1} U_{k-1(+)} & G_{k-1} \end{bmatrix} \begin{bmatrix} D_{k-1(+)} & 0 \\ 0 & D_{Q_{k-1}} \end{bmatrix} \begin{bmatrix} U_{k-1(+)}^{\mathrm{T}} \Phi_{k-1}^{\mathrm{T}} \\ G_{k-1}^{\mathrm{T}} \end{bmatrix} \tag{7.102}$$

$$= \Phi_{k-1} U_{k-1(+)} D_{k-1(+)} U_{k-1(+)}^{\mathrm{T}} \Phi_{k-1}^{\mathrm{T}} + G_{k-1} D_{Q_{k-1}} G_{k-1}^{\mathrm{T}} \tag{7.103}$$

$$= \Phi_{k-1} P_{k-1(+)} \Phi_{k-1}^{\mathrm{T}} + Q_{k-1} \tag{7.104}$$

$$= P_{k(-)}. \tag{7.105}$$

Consequently, the factors

$$U_{k(-)} = L^{\mathrm{T}}, \tag{7.106}$$

$$D_{k(-)} = D_{\beta} \tag{7.107}$$

are the solutions of the temporal update problem for the UD filter.

Diagonalizing Q It is generally worth the effort to "diagonalize" Q (if it is not already a diagonal matrix), because the net computational complexity is reduced by this procedure. The formula given for total computational complexity of the Thornton algorithm in Table 7.16 includes the computational complexity for the *UD* decomposition of Q as $U_Q\acute{Q}U_Q^{\mathrm{T}}$ for $\acute{Q}$ diagonal $[\frac{1}{6}p(p-1)(p+4)$ flops] plus the computational complexity for multiplying G by the resulting $p \times p$ unit upper triangular factor U_Q $[\frac{1}{2}np(p-1)$ flops] to obtain $\acute{G}$.

The algorithm listed in Table 7.16 operates with the matrix blocks ΦU, $\acute{G}$, D, and $\acute{Q}$ by name and not as submatrices of some larger matrix. It is not necessary to physically relocate these matrices into larger arrays in order to apply the Björck orthogonalization procedure. The algorithm is written to find them in their original arrays. (It makes the listing a little longer, but the computational complexity is the same.)

Multiplying ΦU in Place The complexity formulas in Table 7.16 also include the computations required for forming the product ΦU of the $n \times n$ state-transition matrix Φ and the $n \times n$ unit upper triangular matrix U. This matrix product can be performed in place—overwriting Φ with the product ΦU—by the following algorithm:

```
for i = 1 : n,
  for j = n : -1:1,
    sigma = Phi (i, j);
    for k = 1 : j - i,
      sigma = sigma + Phi (i, k)*U (k, j);
    end; Phi (i, j) = sigma;
  end;
end;
```

The computational complexity of this specialized matrix multiplication algorithm is $n^2(n-1)/2$, which is less than half of the computational complexity of the general $n \times n$ matrix product (n^3). Performing this multiplication in place also frees up the array containing $U_{k(+)}$ to accept the updated value $U_{k+1(-)}$. In some applications, Φ will have a sparse structure that can be exploited to reduce the computational requirements even more.

7.6 *SIGMARHO* FILTERING

Riccati equation implementations using factors of the covariance matrix P not only improve numerical stability of the transformed Riccati equation but also guarantee symmetry and nonnegative definiteness of the resulting covariance matrix.

TABLE 7.16 Thornton *UD* Temporal Update Algorithm[*]

Symbol	Description
Inputs:	
D	The $n \times n$ diagonal matrix. Can be stored as an n-vector.
ΦU	Matrix product of $n \times n$ state-transition matrix Φ and $n \times n$ unit upper triangular matrix U such that $UDU^{\mathrm{T}} = P_{(+)}$, the covariance matrix of a *posteriori* state estimation uncertainty.
$\acute{G}$	$= GU_Q$. The modified $n \times p$ process noise coupling matrix, where $Q = U_Q D_Q U_Q^{\mathrm{T}}$
D_Q	Diagonalized $p \times p$ covariance matrix of process noise. Can be stored as a p-vector.
Outputs	
ΦU is overwritten by intermediate results.	
$\acute{G}$ is overwritten by intermediate results.	
$\acute{D}$	The $n \times n$ diagonal matrix. Can be stored as an n-vector.
$\acute{U}$	The $n \times n$ unit upper triangular matrix such that $\acute{U}\acute{D}\acute{U}^{\mathrm{T}} = \Phi UDU^{\mathrm{T}}\Phi^{\mathrm{T}} + GQG^{\mathrm{T}}$.

```
for i = n : -1:1,
  sigma = 0;
  for j = 1 : n,
    sigma = sigma + Phi U (i, j)^ 2*D (j, j);
  end;
  for j = 1 : p,
    sigma = sigma + G (i, j)^ 2*DQ (j, j);
  end;
  D (i, i) = sigma;
  U (i, i) = 1;
  for j = 1 : i - 1,
    sigma = 0;
    for k = 1 : n,
      sigma = sigma + Phi U (i, k)*D (k, k)*Phi U (j, k);
    end;
    for k = 1 : p,
      sigma = sigma + G (i, k)*DQ (k, k)*G (j, k);
    end;
    U (j, i) = sigma/D (i, i);
    for k = 1 : n,
      Phi U (j, k) = Phi U (j, k) - U (j, i)*Phi U (i, k);
    end;
    for k = 1 : p,
      G (j, k) = G (j, k) - U (j, i)*G (i, k);
    end;
  end;
end;
```

(*continued*)

TABLE 7.16 (*Continued*)

Symbol	Description
Computational Complexity Breakdown (in flops)	
Operation	flops
Matrix product ΦU	$n^2(n-1)/2$
Solve $U_Q D_Q U_Q^{\mathrm{T}} = Q, \acute{G} = G U_Q$	$p(p-1)(3n+p+4)/6$
Thornton algorithm	$3n(n-1)(n+p)/2$
Total	$n(n-1)(4n+3p-1)/2 + p(p-1)(3n+p+4)/6$

*Performs the temporal update of the modified Cholesky factors (*UD* factors) of the covariance matrix of state estimation uncertainty for the Kalman filter.

But these factored approaches have their own drawbacks, the major one being that the factors provide little insight about the nature of the estimation uncertainties. It then becomes necessary to recompute the covariance matrix P from its factors for the purposes of monitoring and assessing estimation uncertainty in familiar terms. Even after P has been computed, it is commonly processed further to calculate the standard deviations σ_i and correlation coefficients ρ_{ij} of state estimation uncertainties to better understand what is going on, and why. These statistical parameters are particularly important during the research, development, testing, and evaluation phases of systems with embedded Kalman filters. They are useful not only for diagnosing filter performance but also for employing corrective adaptation of the filter parameters.

A second problem—from the standpoint of implementation in faster fixed-point arithmetic—is the potentially unconstrained dynamic range of the variables of the factored covariance matrix. Even though the "Bierman rule"[8] suggests a reduction in word-length requirements, the potential dynamic ranges of the variables in the factored Riccati equations are beyond what is needed for straightforward fixed-point implementations.

The "*sigmaRho*" implementation of Grewal and Kain [27] bypasses these issues by propagating the σ_i and ρ_{ij} (hence the name) directly, which facilitates implementation of adaptive measures by directly providing the essential performance diagnostic variables.

Perhaps the most promising attribute of the *sigmaRho* filter is that it is well scaled for implementation in faster fixed-point arithmetic for implementation in specialized high speed digital signal processors. As system-on-chip (SoC) embedded solutions become more mainstream for algorithmic solutions such as the "software radio," high speed Kalman filter implementations are expected to take on greater importance.

The *sigmaRho* implementation equations include three levels of scaling, so they will be derived here in stages:

[8]The claim by the late Gerald J. Bierman (1941–1987) that "square-root" Kalman filtering methods provide "the same precision with half as many bits."

1. Define the basic filter covariance variables, as presented in Chapter 3.
2. Redefine the state variables as the ratios of the standard Kalman filter state variables and their respective standard deviations.
3. Derive the dynamics of these basic state variables, standard deviations, and correlation coefficients in continuous time.
4. Scale the standard deviations for fixed-point representation, and derive the continuous dynamic model for the resulting variables.
5. Derive the equivalent dynamic model in discrete time.
6. Scale the state variables for implementation in fixed-point arithmetic, and derive the appropriate dynamic model for the rescaled variables.
7. Derive the discrete-time measurement update equations for this scaled-state model.

The last two of these stages provide the final implementation equations, using the results of stages 1–5 for their derivations.

7.6.1 Sigma and Rho

Standard deviations (σ_i) and correlation coefficients (ρ_{ij}) are defined in Section 3.3.3.6, where it is shown that a covariance matrix P can be factored as

$$P = D_\sigma C_\rho D_\sigma \tag{7.108}$$

$$= \begin{bmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 & \rho_{13}\sigma_1\sigma_3 & \cdots & \rho_{1n}\sigma_1\sigma_n \\ \rho_{21}\sigma_2\sigma_1 & \sigma_2^2 & \rho_{23}\sigma_2\sigma_3 & \cdots & \rho_{2n}\sigma_2\sigma_n \\ \rho_{31}\sigma_3\sigma_1 & \rho_{32}\sigma_3\sigma_2 & \sigma_3^2 & \cdots & \rho_{3n}\sigma_3\sigma_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho_{n1}\sigma_n\sigma_1 & \rho_{n2}\sigma_n\sigma_2 & \rho_{n3}\sigma_n\sigma_3 & \cdots & \sigma_n^2 \end{bmatrix} \tag{7.109}$$

$$D_\sigma = \operatorname{diag}[\sigma_1,\ \sigma_2,\ \sigma_3,\ \ldots,\ \sigma_n] \tag{7.110}$$

$$= \begin{bmatrix} \sigma_1 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & \cdots & 0 \\ 0 & 0 & \sigma_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \sigma_n \end{bmatrix} \text{(diagonal matrix)} \tag{7.111}$$

$$C_\rho = \begin{bmatrix} 1 & \rho_{12} & \rho_{13} & \cdots & \rho_{1n} \\ \rho_{21} & 1 & \rho_{23} & \cdots & \rho_{2n} \\ \rho_{31} & \rho_{32} & 1 & \cdots & \rho_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho_{n1} & \rho_{n2} & \rho_{n3} & \cdots & 1 \end{bmatrix} \text{(correlation matrix)} \tag{7.112}$$

$$\rho_{ij} = \rho_{ji}, \tag{7.113}$$

where the σ_i are the *standard deviations* and the ρ_{ij} are the *correlation coefficients*. That is,

$$\sigma_i \overset{\text{def}}{=} \sqrt{p_{ii}} \tag{7.114}$$

is the standard deviation of the uncertainty of the ith state variable, ρ_{ij} is the correlation coefficient between the ith and jth state variables, and

$$-1 \leq \rho_{ij} \leq +1. \tag{7.115}$$

The values of the σ_i have the same units as the corresponding state vector components $\hat{x}_i$, and they provide an indication of how well each of the state variables is being estimated. The value of a correlation coefficient ρ_{ij}, if it is near +1 or −1, indicates that the uncertainties in the ith and jth state variables are closely linked statistically, a condition which might be improved by changing the measurements that are being used. For example, a sensor measuring only $x_i + x_j$ would tend to make $\rho_{ij} \to -1$, which can be countered by adding a sensor measuring $x_i - x_j$.

In all implementations except *sigmaRho,* computation of the σ_i and the ρ_{ij} requires calculations beyond those necessary for computing P or its factors.

7.6.2 Basic Dynamic Model in Continuous Time

7.6.2.1 Basic State Variables Basic *sigmaRho* state variables x'_i are normalized by dividing the standard state variables x_i of the Kalman filter model by their respective standard deviations of estimation uncertainty:

$$x'(t) \overset{\text{def}}{=} D^{-1}_{\sigma(t)} x(t), \tag{7.116}$$

$$x'_i(t) = \frac{x_i(t)}{\sigma_i(t)}, 1 \leq i \leq n, \tag{7.117}$$

where both the x_i and their respective standard deviations σ_i will be functions of time.

7.6.2.2 Basic State Dynamics The dynamics of the standard Kalman filter model state variable x are assumed to be defined by a model of the sort

$$\dot{x}(t) = f(x) + w(t), \tag{7.118}$$

where $f(x)$ is continuously differentiable and time invariant and $\{w(t)\}$ is a zero-mean white noise process with known constant covariance Q.

From Equation 7.116, the time derivative of the state variable x' will now involve time derivatives of x and σ. Using Equation 7.118, the time derivatives of the components x'_i can then be derived as

$$\frac{d}{dt}x'_i = \frac{d}{dt}\frac{x_i}{\sigma_i} \tag{7.119}$$

$$= \frac{\dot{x}_i}{\sigma_i} - x_i\frac{\dot{\sigma}_i}{\sigma_i^2} \tag{7.120}$$

$$= \frac{f_i(x)}{\sigma_i} - x'_i\frac{\dot{\sigma}_i}{\sigma_i}, \tag{7.121}$$

$$\frac{d}{dt}x' = D_\sigma^{-1}[f(D_\sigma x') - D_{x'}\dot{\sigma}], \tag{7.122}$$

where f_i is the ith component of the vector-valued function f and $\dot{\sigma}_i$ will be derived in the next subsubsection.

7.6.2.3 Basic Covariance Dynamics The dynamics of P will be replaced by the dynamics of the σ_i and ρ_{ij}, assuming the dynamics of P to be adequately modeled by the linearized Riccati equation:

$$\dot{P} = FP + PF^{\mathrm{T}} + Q \tag{7.123}$$

$$F \overset{\text{def}}{=} \frac{\partial}{\partial x}f(x) \tag{7.124}$$

$$= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \cdots & \frac{\partial f_3}{\partial x_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \frac{\partial f_n}{\partial x_3} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}. \tag{7.125}$$

The dynamics of the σ_i and ρ_{ij} can then be derived from Equation 7.123, using the factorization of Equation 7.108. The variable p_{ij} is defined as the element in the ith row and jth column of P. From Equation 7.108, its time derivative

$$\dot{P}_{ij} = \frac{d}{dt}[\rho_{ij}\sigma_i\sigma_j] \tag{7.126}$$

$$= \dot{\rho}_{ij}\sigma_i\sigma_j + \rho_{ij}\dot{\sigma}_i\sigma_j + \rho_{ij}\sigma_i\dot{\sigma}_j. \tag{7.127}$$

In the case that $i = j$, however,

$$\rho_{ii} = 1 \tag{7.128}$$

$$\dot{\rho}_{ii} = 0 \tag{7.129}$$

$$\dot{P}_{ii} = 2\ \dot{\sigma}_i \sigma_i, \tag{7.130}$$

and, from Equation 7.123,

$$\dot{P}_{ii} = \sum_{j=1}^{n} [f_{ij}\rho_{ji}\sigma_j\sigma_i + f_{ij}\rho_{ij}\sigma_i\sigma_j] + q_{ii} \tag{7.131}$$

$$= 2\ \sum_{j=1}^{n} f_{ij}\rho_{ji}\sigma_j\sigma_i + q_{ii}, \tag{7.132}$$

where f_{ij} is the element in the ith row and jth column of F and q_{ij} is the element in the ith row and jth column of Q.

Next, from Equation 7.130,

$$\dot{\sigma}_i = \sum_{j=1}^{n} f_{ij}\rho_{ji}\sigma_j + \frac{q_{ii}}{2\sigma_i} \tag{7.133}$$

$$= \left\{ FD_\sigma C_\rho + \frac{1}{2} D_\sigma Q \right\}_{ii}, \tag{7.134}$$

which provides formulas for the dynamics of σ_i in terms of F, Q, and the σ_j and ρ_{ij} for $1 \le j \le n$.

7.6.2.4 Auxilliary Variables Formulas for the dynamics of x' and the ρ_{ij} can be tidied up a bit by using the intermediary $n \times n$ matrix M

$$M \overset{\text{def}}{=} D_\sigma^{-1} F D_\sigma C_\rho, \tag{7.135}$$

$$\text{with elements } m_{ij} = \frac{1}{\sigma_i} \sum_{\ell=1}^{n} f_{i\ell}\rho_{\ell j}\sigma_\ell, \tag{7.136}$$

in which case Equation 7.133 becomes

$$\frac{\dot{\sigma}_i}{\sigma_i} = m_{ii} + \frac{q_{ii}}{2\sigma_i^2}, \tag{7.137}$$

$$\text{or } \dot{\sigma}_i = m_{ii}\sigma_i + \frac{q_{ii}}{2\sigma_i}, \tag{7.138}$$

where Equation 7.137 is the form for substitution in Equations 7.121 and 7.138 is the form needed just as a formula for the derivative of σ_i.

TABLE 7.17 Basic *sigmaRho* Dynamics in Continuous Time

State variables	$x' \overset{\text{def}}{=} D_\sigma^{-1} x$
Auxiliary variables	$M = D_\sigma^{-1} F D_\sigma C_\rho$
σ dynamics	$\dot{\sigma}_i = m_{ii}\sigma_i + \frac{q_{ii}}{2\sigma_i}$
ρ dynamics	$\dot{\rho}_{ij} = -\left[\frac{\dot{\sigma}_i}{\sigma_i} + \frac{\dot{\sigma}_j}{\sigma_j}\right] \rho_{ij} + m_{ij} + m_{ji} + \frac{q_{ij}}{\sigma_i \sigma_j}$
State dynamics	$\dot{x}'_i = \frac{1}{\sigma_i} f_i(D_\sigma x') - x'_i \frac{\dot{\sigma}_i}{\sigma_i}$

Notation:
f_{ij} is the ijth element of the matrix $F = \frac{\partial f(x)}{\partial x}$.
f_i is the ith component of the vector $f(\dot{s})$.

When $i \neq j$, Equations 7.123 and 7.127 will yield the identity

$$\dot{\rho}_{ij}\sigma_i\sigma_j + \rho_{ij}\dot{\sigma}_i\sigma_j + \rho_{ij}\sigma_i\dot{\sigma}_j = \sum_{\ell=1}^{n} [f_{i\ell}\rho_{\ell j}\sigma_\ell\sigma_j + f_{j\ell}\rho_{i\ell}\sigma_i\sigma_\ell] + q_{ij}, \tag{7.139}$$

which can now be solved for

$$\dot{\rho}_{ij} = -\left[\frac{\dot{\sigma}_i}{\sigma_i} + \frac{\dot{\sigma}_j}{\sigma_j}\right] \rho_{ij} + m_{ij} + m_{ji} + \frac{q_{ij}}{\sigma_i\sigma_j}, \tag{7.140}$$

where the expressions in square brackets are results of applying Equation 7.137.

The corresponding equation for the derivative of σ_i is Equation 7.138.

This completes the basic *sigmaRho* dynamic model derivations in continuous time, which are summarized in Table 7.17.

7.6.2.5 Integration Issues For some applications, the numerical integration of state and statistical dynamics can be performed by simple trapezoidal integration. This is the case, for example, when sampling frequencies exceed twice the state dynamic bandwidths.

More modern numerical methods are also available to solve nonlinear differential equations in a generalized way regardless of the system dynamics bandwidths. These methods require a computer procedure for computing the derivatives (provided in Tables 7.17 and 7.18), a value of all dynamic terms (state and statistics) at the start of the integration epoch, and a delta-time over which to perform the propagation. The total integration is performed over the input delta-time (between measurements) but this delta-time is broken into subintervals to enforce user-specified criteria for the accuracy of the integration process. An excellent discussion of various integration methods along with software implementing these methods is found in Reference 28. The Richardson extrapolation [29] as implemented by Stoer and Bulirsch (see, e.g., Reference 30 or section 17.3 on p. 921 of Reference 28) is perhaps the most reliable

TABLE 7.18 Normalized *sigmaRho* Dynamics in Continuous Time

State variables	x^*	$\overset{\text{def}}{=}$	$D_{\sigma\text{MAX}}^{-1} D_\sigma^{-1} x$
Standard deviations	σ'	$=$	$D_{\sigma\text{MAX}}^{-1} \sigma$
Auxiliary variables	F'	$\overset{\text{def}}{=}$	$D_{\sigma\text{MAX}}^{-1} F D_{\sigma\text{MAX}}$
	f'_{ij}	$=$	$f_{ij} \sigma_{j\text{MAX}} / \sigma_{i\text{MAX}}$
	Q'	$\overset{\text{def}}{=}$	$D_{\sigma\text{MAX}}^{-1} Q D_{\sigma\text{MAX}}^{-1}$
	q'_{ij}	$=$	$q_{ij} / \sigma_{i\text{MAX}} / \sigma_{j\text{MAX}}$
	M'	$\overset{\text{def}}{=}$	$D_{\sigma'}^{-1} F D_{\sigma'} C_\rho$
	m'_{ij}	$=$	$\frac{1}{\sigma'_i} \sum_{\ell=1}^{n} f'_{i\ell} \rho_{\ell j} \sigma'_\ell$
σ dynamics	$\frac{\dot{\sigma}'_i}{\sigma'_i}$	$=$	$m'_{ii} + \frac{q'_{ii}}{2\sigma'^{\,2}_i}$
ρ dynamics	$\dot{\rho}_{ij}$	$=$	$-\left[\frac{\dot{\sigma}'_i}{\sigma'_i} + \frac{\dot{\sigma}'_j}{\sigma'_j}\right] \rho_{ij} + m'_{ij} + m'_{ji} + \frac{q'_{ij}}{\sigma'_i \sigma'_j}$
State dynamics	$\dot{x}^*_i$	$=$	$f_i(D_\sigma D_{\sigma_{\text{MAX}}} x^*) / \sigma_i / \sigma_{i\text{MAX}} - x^*_i \frac{\dot{\sigma}'_i}{\sigma'_i}$

and efficient method for integrating smooth functions. The fourth-order Runge Kutta method with adaptive step size is considered the most reliable method for dynamic systems that may contain nonsmooth functions [28].

A key detractor for use of such numerical integration methods is that the execution times may not be constant even for constant integration intervals. It is often the case that startup transients result in high rates-of-change early on in the filtering history, so that adaptive step size operations produce more integration subintervals. Such variable and/or unpredictable execution time per measurement is undesirable as the filter designer moves from the conceptual design stage to the operation design stage.

7.6.3 Scaling the σ_i

There are still some numerical scaling issues with the basic *sigmaRho* equations of Table 7.17 for high speed fixed-point implementations.

The numerical values of the variables ρ_{ij} will be between -1 and $+1$, which makes them well conditioned for computation and well suited for implementation in fixed-point arithmetic in low cost high speed signal processors. Computation of the m_{ij} is also well-structured for dot-product implementation in specialized signal processors, although the scaling may need some adjusting for fixed-point implementation.

If the estimation problem is sufficiently observable, then the values of the σ_i will be bounded. It is not uncommon in practice that the maximum expected values are used as the initial standard deviations.

In either case, the numerical values of the dynamic variables (except ρ_{ij}) can be also be rescaled to make them lie between 0 and 1—by dividing by the maximum expected value of the respective σ_i:

$$\sigma_{i\mathrm{MAX}} \overset{\mathrm{def}}{=} \max_t[\sigma_i(t)] \tag{7.141}$$

$$x_i^* \overset{\mathrm{def}}{=} x_i'/\sigma_{i\mathrm{MAX}} \tag{7.142}$$

$$\sigma_i' \overset{\mathrm{def}}{=} \sigma_i/\sigma_{i\mathrm{MAX}} \tag{7.143}$$

$$\tag{7.144}$$

In practice, the $\sigma_{i\mathrm{MAX}}$ would be determined by simulations—with perhaps a little fudging just to make sure that $\sigma_i \leq \sigma_{i\mathrm{MAX}}$ under all expected operating conditions.

This normalization also rescales the state vector to simplify the dynamic model, but that part of the scaling may not be adequate for fixed-point implementation. The resulting normalized *sigmaRho* dynamic equations in continuous time are summarized in Table 7.18

7.6.4 *SigmaRho* Dynamics in Discrete Time

7.6.4.1 Scaling the State Vector for Fixed-point Implementations This first-level normalization is to keep the computed values of the $\sigma_i' \leq 1$, but there is nothing in the resulting model to prevent components of the normalized state vector x^* from exceeding the range $-1 \leq x_i^* \leq +1$ required for fixed-point implementation. However, it is standard practice in fixed-point implementations to redefine the units of the x_i^* to keep them scaled to the arithmetic binary-point limits of the arithmetic processor. This is accomplished by multiplying by a scaling factor λ_i (commonly a power of 2, so that scaling changes only require bit-shifting, not full multiplication) such that the rescaled variable satisfies the scaling constraint $-1 \leq \lambda_i x_i^* \leq +1$.

7.6.4.2 Scaled Discrete-time Model Dynamic variables of the filter in discrete time include the state vector (x) and its covariance matrix of uncertainty (P), which are normally modeled by equations of the sort

$$x_k = \Phi_{k-1}x_{k-1} + w_{k-1} \tag{7.145}$$

$$\mathop{\mathrm{E}}_{w}\left\langle w_i w_j^{\mathrm{T}} \right\rangle = \begin{cases} 0, & i \neq j \\ Q_i, & i = j \end{cases} \tag{7.146}$$

$$P_k = \Phi_{k-1}P_{k-1}\Phi_{k-1}^{\mathrm{T}} + Q_{k-1} \tag{7.147}$$

$$\Phi_{k-1} = \exp\ [(t_k - t_{k-1})F], \tag{7.148}$$

where F is given by Equation 7.124. However, if f is nonlinear, it is also possible (and perhaps desirable) to propagate the estimate forward in time[9] by integration of $\dot{x} = f(x)$.

[9]This approach, which is common for "extended" Kalman filtering, is also extendable to "Unscented" Kalman filtering (see Chapter 8).

7.6.4.3 Discrete-time Dynamics We are sometimes on safer ground starting with familiar dynamic models in continuous time to develop the Kalman filter in discrete time, and Chapter 2 contains formulas for transitioning from models in continuous time to equivalent models in discrete time. However, some of these formulas are not very efficient for applications with time-varying dynamics, because values of the state-transition matrix Φ may need to be calculated in real time. This may involve taking matrix exponentials, which can be risky [31]. If the time steps Δt are sufficiently small, approximations such as

$$\Phi_{k-1} \approx [I + F(t_{k-1})\ \Delta t] \tag{7.149}$$

$$Q_{k-1} \approx Q(t_{k-1})\ \Delta t \tag{7.150}$$

may suffice, or the formula (due to Van Loan [32]):

$$\exp\left(\begin{bmatrix} F\left(t_{k-1}\right) & Q(t_{k-1}) \\ 0 & -F^{\mathrm{T}}(t_{k-1}) \end{bmatrix}\ \Delta t\right) \approx \begin{bmatrix} \Psi & \Phi_{k-1}^{-1} Q_k \\ 0 & \Phi_{k-1}^{\mathrm{T}} \end{bmatrix}. \tag{7.151}$$

In practice, it is always necessary to evaluate the validity of these approximations as part of the development process.

7.6.4.4 SigmaRho State Variables in Discrete Time As in the model for continuous time, the *sigmaRho* state variables are normalized with respect to the standard deviations, component by component:

$$x' \overset{\text{def}}{=} D_{\sigma(+)}^{-1} x \tag{7.152}$$

$$\{x'_k\}_i = \frac{\{x_k\}_i}{\{\sigma_{k-1(+)}\}_i}, \tag{7.153}$$

where the notation $\{x_k\}_i$ denotes the ith component of x_k, the value of the state vector at discrete time t_k, and $\{\sigma_{k-1(+)}\}_i$ denotes the ith component of the standard deviations after the last measurement.

7.6.4.5 Rescaled Covariance Variables With this change of state variable, the rescaled estimation error

$$\delta x' = D_{\sigma(+)}^{-1} \delta x \tag{7.154}$$

and its covariance matrix

$$P' \overset{\text{def}}{=} \mathop{\mathrm{E}}_{\delta x}\langle \delta x' \delta x'^{\mathrm{T}} \rangle \tag{7.155}$$

$$= D_{\sigma(+)}^{-1} \mathop{\mathrm{E}}_{\delta x}\langle \delta x\ \delta x^{\mathrm{T}} \rangle D_{\sigma(+)}^{-1} \tag{7.156}$$

$$= D_{\sigma(+)}^{-1} P D_{\sigma(+)}^{-1} \tag{7.157}$$

$$= C_{\rho(+)}, \tag{7.158}$$

where P is the covariance matrix for the uncertainty of x and $C_{\rho(+)}$ is the corresponding correlation coefficient matrix after the last measurement update.

That is, the covariance matrix for the rescaled state vector is now the a posteriori correlation coefficient matrix.

If the dynamic model function $f(x)$ is time invariant and $\{\sigma_{k-1(+)}\}_i$ remains constant between measurements, the time derivatives

$$\frac{d}{dt}x' = D^{-1}_{\sigma(+)}\frac{d}{dt}x \tag{7.159}$$

$$= D^{-1}_{\sigma(+)}[Fx(t) + w(t)] \tag{7.160}$$

$$\dot{x}'_i = \frac{1}{\sigma_{i(+)}}\left[\sum_{j=1}^{n} f_{ij}x_j + w_i(t)\right] \tag{7.161}$$

$$= \sum_{j=1}^{n} f_{ij}\frac{\sigma_{j(+)}}{\sigma_{i(+)}}x'_j + \frac{w_i(t)}{\sigma_{i(+)}} \tag{7.162}$$

$$= \sum_{j=1}^{n} f'_{ij}x'_j + \frac{w_i(t)}{\sigma_{i(+)}} \tag{7.163}$$

$$f'_{ij} \overset{\text{def}}{=} \frac{\sigma_{j(+)}}{\sigma_{i(+)}}f_{ij} \tag{7.164}$$

$$F' = [f'_{ij}], \text{ an } n \times n \text{ matrix} \tag{7.165}$$

$$= D^{-1}_{\sigma(+)}FD_{\sigma(+)} \tag{7.166}$$

$$F = \frac{\partial f(x)}{\partial x}. \tag{7.167}$$

Now, for this rescaled time-invariant system with dynamic coefficient matrix F', we can define a new state-transition matrix for time step Δt in terms of the state transition for F:

$$\Phi' \overset{\text{def}}{=} \exp\,[F'\ \Delta t] \tag{7.168}$$

$$= \sum_{k=0}^{\infty}\frac{1}{k!}F'^k\Delta t^k \tag{7.169}$$

$$= \sum_{k=0}^{\infty}\frac{1}{k!}[D^{-1}_{\sigma(+)}FD_{\sigma(+)}]^k\Delta t^k \tag{7.170}$$

$$= \sum_{k=0}^{\infty}\frac{1}{k!}D^{-1}_{\sigma(+)}[F]^kD_{\sigma(+)}\Delta t^k \tag{7.171}$$

$$= D_{\sigma(+)}^{-1}\left[\sum_{k=0}^{\infty}\frac{1}{k!}F^k\Delta t^k\right]D_{\sigma(+)} \tag{7.172}$$

$$= D_{\sigma(+)}^{-1}\exp\ [F\Delta t]D_{\sigma(+)} \tag{7.173}$$

$$= D_{\sigma(+)}^{-1}\Phi D_{\sigma(+)} \tag{7.174}$$

$$\Phi \overset{\text{def}}{=} \exp\ [F\ \Delta t]. \tag{7.175}$$

7.6.4.6 Covariance Time Update The a posteriori covariance $P'_{(+)} = C_{\rho(+)}$, and the next a priori covariance

$$P'_{k(-)} = \Phi' C_{\rho(k-1)(+)}\Phi'^{\mathrm{T}} + Q'_{k-1}, \tag{7.176}$$

from which

$$\{P'_{k(-)}\}_{ii} = \left[\frac{\sigma'_{i\ k(-)}}{\sigma'_{i\ (k-1(+)}}\right]^2 \tag{7.177}$$

$$\frac{\sigma'_{k(-)i}}{\sigma'_{i\ (k-1(+)}} = \sqrt{[\Phi' C_{\rho\ k-1(+)}\Phi'^{\mathrm{T}} + Q'_k]_{ii}} \tag{7.178}$$

$$\{P_{k(-)}\}_{ij} = \rho_{k(-)ij}\sigma'_{k(-)i}\sigma'_{k(-)j} \tag{7.179}$$

$$\rho_{k(-)ij} = \left(\frac{\sigma'_{k-1(+)i}}{\sigma'_{k(+)i}}\right)\left(\frac{\sigma'_{k-1(+)j}}{\sigma'_{k(-)j}}\right) \times[\Phi'_{k-1}C_{\rho\ k-1(+)}\Phi'^{\mathrm{T}}_{k-1} + Q'_{k-1}]_{ij}, \tag{7.180}$$

as summarized in Table 7.19.

TABLE 7.19 Normalized *sigmaRho* Dynamics in Discrete Time

Normalized state	x'	$=$	$\lambda D_{\sigma\mathrm{MAX}}^{-1}D_{\sigma'\ k-1(+)}^{-1}x$
Auxilliary variables	λ	$=$	2^p(state scaling)
	Φ'	$=$	$D_{\sigma\mathrm{MAX}}^{-1}D_{\sigma}^{-1}\Phi D_{\sigma}D_{\sigma\mathrm{MAX}}$
	Q'	$=$	$D_{\sigma\mathrm{MAX}}^{-1}D_{\sigma}^{-1}QD_{\sigma}^{-1}D_{\sigma\mathrm{MAX}}^{-1}$
State dynamics	$x'_{k+1(-)}$	$=$	$\left(\frac{\sigma_{k(+)i}}{\sigma_{k(-)i}}\right)\{\Phi'_k x'_{k(+)}\}_i$
Standard deviations	$\frac{\sigma'_{i\ k+1(-)}}{\sigma'_{i\ (k-1(+)}}$	$=$	$\sqrt{[\Phi' C_{\rho\ k-1(+)}\Phi'^{\mathrm{T}} + Q'_k]_{ii}}$
Correlation coefficient	$\rho_{k(-)ij}$	$=$	$\left(\frac{\sigma'_{k-1(+)i}}{\sigma'_{k(+)i}}\right)\left(\frac{\sigma'_{k-1(+)j}}{\sigma'_{k(-)j}}\right) \times [\Phi'_{k-1}C_{\rho\ k-1(+)}\Phi'^{\mathrm{T}}_{k-1} + Q'_{k-1}]_{ij}$

7.6.5 *SigmaRho* Measurement Updates

7.6.5.1 Standard Equations In conventional Kalman filtering, the measurement update at discrete time t_k uses a measurement $z_k = H_k x_{k(-)} + \upsilon_k$ to improve the a priori estimate $\hat{x}_{k(-)}$ and obtain a better a posteriori estimate $\hat{x}_{k(+)}$ and its associated a posteriori covariance of estimation uncertainty $P_{k(+)}$, using the Kalman gain and update formulas

$$\overline{K}_k = P_{k(-)} H_k^{\mathrm{T}} \underbrace{[H_k P_{k(-)} H_k^{\mathrm{T}} + R_k]}_{P_{\nu\nu,k}}^{-1} \tag{7.181}$$

$$P_{k(+)} = [I - \overline{K}_k H_k] P_{k(-)}, \tag{7.182}$$

$$x_{k(+)} = x_{k(-)} + \overline{K}_k \underbrace{[z_k - H_k x_{k(-)}]}_{\nu_k}, \tag{7.183}$$

where the term in square brackets in Equation 7.183 is the *measurement innovation*[10] ν_k (also called *measurement residual*), representing the information in the matrix that could not be predicted from the estimated state. The term inside square brackets in Equation 7.181 is its theoretical covariance $P_{\nu\nu,k}$, also called the *covariance of innovations* or *covariance of measurement residuals*.

7.6.5.2 Scalar Measurements As we showed for the "square-root" implementations of the Kalman filter, one can always assume that the measurements are scalars, because—if not–they can be easily transformed into independent scalar measurements.[11]

In that case, the measurement sensitivity matrices H_k will be $1 \times n$ row vectors, the Kalman gain $\overline{K}_k$ will be a $n \times 1$ column vector, and the term in square brackets in Equation 7.181 will be a scalar,

$$\sigma^2_{\nu k} = P_{\nu\nu,k}, \tag{7.184}$$

which transforms a matrix divide problem into a scalar divide problem and greatly simplifies the derivation. Also, the scalar

$$R_k = \sigma^2_{\upsilon,k}, \tag{7.185}$$

the variance of measurement noise υ_k.

[10]Innovations are often represented by the Greek letter ν (nu) because they represent "what is new" in the measurement. However, the typeset ν is quite similar to the typeset υ (measurement noise).
[11]Kaminski et al. [33] have shown that the same ploy speeds up the conventional Kalman filter implementation, as well.

7.6.5.3 Kalman Gain When z_k is a scalar, the Kalman gain formula of Equation 7.181 can be composed from vector expressions, using the decomposition

$$P_{k(-)} = D_{\sigma\ k(-)} C_{\rho\ k(-)} D_{\sigma\ k(-)}, \tag{7.186}$$

so that the vector–matrix products

$$\begin{aligned}
H_k P_{k(-)} &= \underbrace{H_k D_{\sigma\ k(-)} C_{\rho\ k(-)}}_{d_k} D_{\sigma\ k(-)} && (7.187)\\
&= d_k D_{\sigma\ k(-)} && (7.188)\\
d_k &\overset{\text{def}}{=} H_k D_{\sigma\ k(-)} C_{\rho\ k(-)} && (7.189)\\
H_k P_{k(-)} H_k^{\mathrm{T}} &= d_k D_{\sigma\ k(-)} H_k^{\mathrm{T}} && (7.190)\\
\sigma^2_{\nu\ k} &= H_k P_{k(-)} H_k^{\mathrm{T}} + R_k && (7.191)\\
&= d_k D_{\sigma\ k(-)} H_k^{\mathrm{T}} + R_k && (7.192)\\
K_k &= \frac{1}{\sigma^2_{\nu\ k}} D_{\sigma\ k(-)} d_k^{\mathrm{T}}, && (7.193)
\end{aligned}$$

a column vector.

7.6.5.4 Covariance Update Using the decomposition of Equation 7.186, Equation 7.182 becomes

$$D_{\sigma\ k(+)} C_{\rho\ k(+)} D_{\sigma\ k(+)} = D_{\sigma\ k(-)} C_{\rho\ k(-)} D_{\sigma\ k(-)} - K_k H_k P_{k(-)}, \tag{7.194}$$

which—using Equations 7.188 and 7.193—becomes

$$\begin{aligned}
D_{\sigma\ k(+)} C_{\rho\ k(+)} D_{\sigma\ k(+)} &= D_{\sigma\ k(-)} C_{\rho\ k(-)} D_{\sigma\ k(-)} - K_k d_k D_{\sigma\ k(-)} && (7.195)\\
&= D_{\sigma\ k(-)} C_{\rho\ k(-)} D_{\sigma\ k(-)} - \frac{1}{\sigma^2_{\nu\ k}} D_{\sigma\ k(-)} d_k^{\mathrm{T}} d_k D_{\sigma\ k(-)} && (7.196)\\
&= D_{\sigma\ k(-)} \left[C_{\rho\ k(-)} - \frac{1}{\sigma^2_{\nu\ k}} d_k^{\mathrm{T}} d_k \right] D_{\sigma\ k(-)} && (7.197)\\
C_{\rho\ k(+)} &= D^{-1}_{\sigma\ k(+)} D_{\sigma\ k(-)} \left[C_{\rho\ k(-)} - \frac{1}{\sigma^2_{\nu\ k}} d_k^{\mathrm{T}} d_k \right] D_{\sigma\ k(-)} D^{-1}_{\sigma\ k(+)}. && (7.198)
\end{aligned}$$

Because the diagonal elements $\rho_{ii} = 1$, the diagonal elements of Equation 7.197 become

$$\sigma^2_{k(+),i} = \sigma^2_{k(+),i} \left[1 - \frac{d^2_{k,i}}{\sigma^2_{\nu\ k}} \right] \tag{7.199}$$

$$\sigma_{k(+),i} = \sigma_{k(+),i} \sqrt{1 - \frac{d^2_{k,i}}{\sigma^2_{\nu\ k}}}. \tag{7.200}$$

The ratios

$$\frac{\sigma_{k(+),i}}{\sigma_{k(+),i}} = \sqrt{1 - \frac{d^2_{k,i}}{\sigma^2_{\nu\ k}}} \tag{7.201}$$

$$= \frac{1}{\sigma_{\nu\ k}} \sqrt{\sigma^2_{\nu\ k} - d^2_{k,i}}, \tag{7.202}$$

are the *standard deviation improvement ratios* from the measurement update.

7.6.5.5 Estimate Update The standard update formula,

$$\hat{x}_{k(+)} = \hat{x}_{k(+)} + K_k[z_k - H_k \hat{x}_{k(-)}], \tag{7.203}$$

7.6.5.6 Normalized Implementation With the scaling of Section 7.6.4.3, the normalized parameters and variables become

$$D_{\sigma' k} = D^{-1}_{\sigma \mathrm{MAX}} D_{\sigma\ k} \tag{7.204}$$

$$H'_k = D_{\sigma' k(-)} H_k \tag{7.205}$$

$$\hat{x}'_k = \lambda D_{\sigma' k} \hat{x}_k, \tag{7.206}$$

and the normalized measurement update implementation equations become

$$d'_k = H'_k D_{\sigma', k(-)} C_{\rho\ k(-)}. \tag{7.207}$$

7.6.6 Efficacy

The *sigmaRho* Kalman filter implementation is still quite new and would benefit from further development and applications experience. Besides its natural heuristic attributes for safe and reliable development of Kalman filter applications, the implementation lends itself to high speed applications where fixed-point arithmetic is required, and potentially to applications where process and measurement noise covariances might be varied for adaptive filtering.

TABLE 7.20 Normalized *SigmaRho* Measurement Update

State variable	x^*	$=$	$\lambda D^{-1}_{\sigma\text{MAX}} D^{-1}_{\sigma} x$
Auxiliary variables	λ	$\overset{\text{def}}{=}$	2^p(scaling factor)
	H'_k	$=$	$D_{\sigma' k(-)} H_k$
	d'_k	$=$	$H'_k D_{\sigma' k(-)} C_{\rho\ k(-)}$
Innovations covariance	$\sigma^2_{\nu\ k}$	$=$	$d_k D_{\sigma\ k(-)} H^{\text{T}}_k + R_k$
Standard deviation improvement ratio	$\dfrac{\sigma_{k(+),i}}{\sigma_{k(+),i}}$	$=$	$\sqrt{1 - \dfrac{d^2_{k,i}}{\sigma^2_{\nu\ k}}}$
Correlation coefficients	$\rho_{k(+)ij}$	$=$	$\left[\dfrac{\sigma'^{-}_i}{\sigma'^{+}_i}\right] \left[\dfrac{\sigma'^{-}_j}{\sigma'^{+}_j}\right] \left[\rho_{k(-)ij} - \dfrac{D_i D_j}{\Omega^2}\right]$
State update	$x'_{i(+)}$	$=$	$\left[\dfrac{\sigma_{k(-)i}}{\sigma^{+}_i}\right] \left\{ x'_i(-) + \dfrac{D_i}{\Omega} \left[\dfrac{\lambda z_k - H'_k \sigma'_k x'_k}{\Omega}\right] \right\}$

The *sigmaRho* filter has been implemented for a sampling rate of 74 MHz for signal tracking of binary phase-shift keying (BPSK) in a software receiver [27]. This particular nonlinear application has closed-form solutions for the state-transition matrix and process noise covariance for a four-state *sigmaRho* implementation. The result was well scaled for fast fixed-point implementation. Performance was very good, which is encouraging for Kalman filter implementations at those speeds. Hopefully, this will spark more interest in further development and evaluation of the *sigmaRho* Kalman filter for embedded Kalman filter applications with speed requirements that might otherwise be considered excessive.

7.7 OTHER IMPLEMENTATION METHODS

The main thrust of this chapter is on the "square-root" filtering methods presented in the previous section. Although factorization methods are probably the most numerically stable implementation methods currently available, there are other alternative methods that may perform adequately on some applications, and there are some earlier alternatives that bear mentioning for their historical significance and for comparisons with the factorization methods.

7.7.1 Earlier Implementation Methods

7.7.1.1 Swerling Inverse Formulation This is *not* recommended as an implementation method for the Kalman filter, because its computational complexity and numerical stability place it at a disadvantage relative to the other methods. Its computational complexity is derived here for the purpose of demonstrating this.

Recursive Least Mean Squares This form of the covariance update for recursive least-mean-squares estimation was published by Swerling [34]. Swerling's estimator

TABLE 7.21 Operation Summary for Swerling Inverse Formulation

Operation	Flops
R^{-1}	$\ell^3 + \frac{1}{2}\ell^2 + \frac{1}{2}\ell$ if $\ell > 1$, 1 if $\ell = 1$
$(R^{-1})H$	$n\ell^2$
$H^{\mathrm{T}}(R^{-1}H)$	$\frac{1}{2}n^2\ell + \frac{1}{2}n\ell$
$P^{-1(-)} + (H^{\mathrm{T}}R^{-1}H)$	$n^3 + \frac{1}{2}n^2 + \frac{1}{2}n$
$[P^{-1(-)} + H^{\mathrm{T}}R^{-1}H]^{-1}$	$n^3 + \frac{1}{2}n^2 + \frac{1}{2}n$
Total	$2n^3 + n^2 + n + \frac{1}{2}n^2\ell + n\ell^2 + \frac{1}{2}n\ell + \ell^3 + \frac{1}{2}\ell^2 + \ell$

was essentially the Kalman filter but with the observational update equation for the covariance matrix in the form

$$P_{(+)} = [P_{(-)}^{-1} + H^{\mathrm{T}}R^{-1}H]^{-1}.$$

This implementation requires three matrix inversions and two matrix products. If the observation is scalar valued ($m = 1$), the matrix inversion R^{-1} requires only one divide operation. One can also take advantage of diagonal and symmetric matrix forms to make the implementation more efficient.

Computational Complexity of Swerling Inverse Formulation[12] For the case that the state dimension $n = 1$ and the measurement dimension $\ell = 1$, it can be done with 4 flops. In the case that $n > 1$, the dependence of the computational complexity on ℓ and n is shown in Table 7.21.[13] This is the most computationally intensive method of all. The number of arithmetic operations increases as n^3. The numbers of operations for the other methods increase as $n^2\ell + \ell^3$, but usually $n > \ell$ in Kalman filtering applications.

7.7.1.2 Kalman Formulation

Data Flows A data flow diagram of the implementation equations defined by Kalman [36] is shown in Figure 7.8. This shows the points at which the measurements (z) and model parameters (H, R, Φ, and Q) are introduced in the matrix operations. There is some freedom to choose exactly how these operations shown will be implemented, however, and the computational requirements can be reduced substantially by reuse of repeated subexpressions.

[12]See Section 7.4.4.7 for an explanation of how computational complexity is determined.

[13]There is an alternative matrix inversion method (due to Strassen [35]) that reduces the number of multiplies in an $n \times n$ matrix inversion from n^3 to $n^{\log_2 7}$ but increases the number of additions significantly.

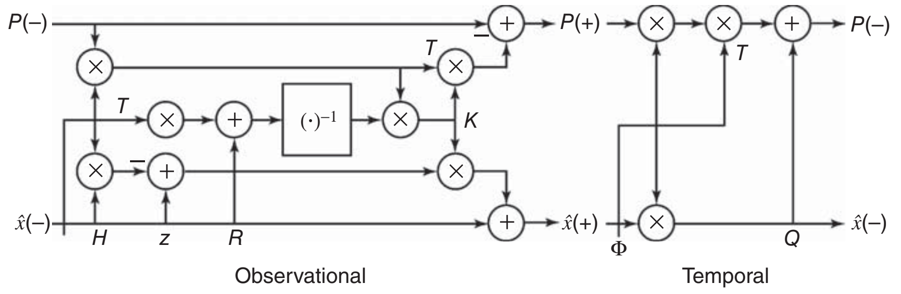

Figure 7.8 Data flows of Kalman update implementations.

Reuse of Partial Results In this "conventional" form of the observational update equations, the factor $\{HP\}$ occurs several times in the computation of $\overline{K}$ and P:

$$\overline{K} = P_{(-)}H^{\mathrm{T}}[HP_{(-)}H^{\mathrm{T}} + R]^{-1} \tag{7.208}$$

$$= \{HP_{(-)}\}^{\mathrm{T}}[\{HP_{(-)}\}H^{\mathrm{T}} + R]^{-1}, \tag{7.209}$$

$$P_{(+)} = P_{(-)} - \overline{K}\{HP_{(-)}\}. \tag{7.210}$$

The factored form shows the reusable partial results $[HP_{(-)}]$ and $\overline{K}$ (the Kalman gain). Using these partial results where indicated, the implementation of the factored form requires four matrix multiplies and one matrix inversion. As in the case of the Swerling formulation, the matrix inversion can be accomplished by a divide if the dimension of the observation (ℓ) is 1, and the efficiency of all operations can be improved by computing only the unique elements of symmetric matrices. The number of floating-point arithmetic operations required for the observational update of the Kalman filter, using these methods, is summarized in Table 7.22. Note that the total number of operations required does not increase as n^3, as was the case with the Swerling formulation.

7.7.1.3 Potter "Square-Root" Filter The inventor of "square-root" filtering is James E. Potter. Soon after the introduction of the Kalman filter, Potter introduced the idea of factoring the covariance matrix and provided the first of the square-root methods for the observational update (see Battin [37, pp. 338–340] and Potter and Stern [38]).

Potter defined the generalized Cholesky factor of the covariance matrix P as

$$P_{(-)} \overset{\text{def}}{=} C_{(-)}C_{(-)}^{\mathrm{T}}, \tag{7.211}$$

$$P_{(+)} \overset{\text{def}}{=} C_{(+)}C_{(+)}^{\mathrm{T}}, \tag{7.212}$$

so that the observational update equation

$$P_{(+)} = P_{(-)} - P_{(-)}H^{\mathrm{T}}[HP_{(-)}H^{\mathrm{T}} + R]^{-1}HP_{(-)} \tag{7.213}$$

could be partially factored as

$$\begin{aligned} C_{(+)}C_{(+)}^{\mathrm{T}} &= C_{(-)}C_{(-)}^{\mathrm{T}} \\ &\quad - C_{(-)}C_{(-)}^{\mathrm{T}}H^{\mathrm{T}}[HC_{(-)}C_{(-)}^{\mathrm{T}}H^{\mathrm{T}} + R]^{-1}HC_{(-)}C_{(-)}^{\mathrm{T}} \qquad (7.214) \\ &= C_{(-)}C_{(-)}^{\mathrm{T}} - C_{(-)}V[V^{\mathrm{T}}V + R]^{-1}V^{\mathrm{T}}C_{(-)}^{\mathrm{T}} \qquad (7.215) \\ &= C_{(-)}\{I_n - V[V^{\mathrm{T}}V + R]^{-1}V^{\mathrm{T}}\}C_{(-)}^{\mathrm{T}}, \qquad (7.216) \end{aligned}$$

where

$I_n = \times n$ identity matrix,

$V = C_{(-)}^{\mathrm{T}}H^{\mathrm{T}}$ is an $n \times \ell$ general matrix,

$n =$ dimension of state vector, and

$\ell =$ dimension of measurement vector.

Equation 7.216 contains the unfactored expression $\{I_n - V[V^{\mathrm{T}}V + R]^{-1}V^{\mathrm{T}}\}$. For the case that the measurement is a scalar ($\ell = 1$), Potter was able to factor it in the form

$$I_n - V[V^{\mathrm{T}}V + R]^{-1}V^{\mathrm{T}} = WW^{\mathrm{T}}, \qquad (7.217)$$

so that the resulting equation

$$\begin{aligned} C_{(+)}C_{(+)}^{\mathrm{T}} &= C_{(-)}\{WW^{\mathrm{T}}\}C_{(-)}^{\mathrm{T}} \qquad (7.218) \\ &= \{C_{(-)}W\}\{C_{(-)}W\}^{\mathrm{T}} \qquad (7.219) \end{aligned}$$

could be solved for the a posteriori generalized Cholesky factor of $P_{(+)}$ as

$$C_{(+)} = C_{(-)}W. \qquad (7.220)$$

When the measurement is a scalar, the expression to be factored is a symmetric *elementary matrix* of the form[14]

$$I_n - \frac{\mathbf{v}\mathbf{v}^{\mathrm{T}}}{R + |\mathbf{v}|^2}, \qquad (7.221)$$

where R is a positive scalar and $\mathbf{v} = C_{(-)}^{\mathrm{T}}H^{\mathrm{T}}$ is a column n-vector.

[14]This expression—or something very close to it—is used in many of the "square-root" filtering methods for observational updates. The Potter "square-root" filtering algorithm finds a symmetric factor W, which does not preserve triangularity of the product $C_{(-)}W$. The Carlson observational update algorithm (in Section 7.5.1.1) finds a triangular factor W, which preserves triangularity of $C_{(+)} = C_{(-)}W$ if both factors $C_{(+)}$ and W are of the same triangularity (i.e., if both $C_{(-)}$ and W are upper triangular or both lower triangular). The Bierman observational update algorithm uses a related *UD* factorization. Because the rank of the matrix $\mathbf{v}\mathbf{v}^{\mathrm{T}}$ is 1, these methods are referred to as *rank 1 modification methods*.

TABLE 7.22 Operation Summary for Conventional Kalman Filter

Operation	Flops
$H \times P_{(-)}$	$n^2\ell$
$H \times [HP_{(-)}]^{\mathrm{T}} + R$	$\frac{1}{2}n\ell^2 + \frac{1}{2}n\ell$
$\{H[HP_{(-)}]^{\mathrm{T}} + R\}^{-1}$	$\ell^3 + \frac{1}{2}\ell^2 + \frac{1}{2}\ell$
$\{H[HP_{(-)}]^{\mathrm{T}} + R\}^{-1} \times [HP_{(-)}]$	$n\ell^2$
$P_{(-)} - [HP_{(-)}] \times \{H[HP_{(-)}]^{\mathrm{T}} + R\}^{-1}[HP_{(-)}]$	$\frac{1}{2}n^2\ell + \frac{1}{2}n\ell$
Total	$\frac{3}{2}n^2\ell + \frac{3}{2}n\ell^2 + n\ell + \ell^3 + \frac{1}{2}\ell^2 + \frac{1}{2}\ell$

The formula for the symmetric square root of a symmetric elementary matrix is given in Equation 7.44. For the elementary matrix format in Equation 7.221, the scalar s of Equation 7.44 has the value

$$s = \frac{1}{R + |\mathbf{v}|^2}, \tag{7.222}$$

so that the radicand

$$1 - s|\mathbf{v}|^2 = 1 - \frac{|\mathbf{v}|^2}{R + |\mathbf{v}|^2} \tag{7.223}$$

$$= \frac{R}{R + |\mathbf{v}|^2} \tag{7.224}$$

$$\geq 0 \tag{7.225}$$

because the variance $R \geq 0$. Consequently, the matrix expression 7.221 will always have a real matrix square root.

Potter Formula for Observational Updates Because the matrix square roots of symmetric elementary matrices are also symmetric matrices, they are also generalized Cholesky factors. That is,

$$(I - s\mathbf{v}\mathbf{v}^{\mathrm{T}}) = (I - \sigma\mathbf{v}\mathbf{v}^{\mathrm{T}})(I - \sigma\mathbf{v}\mathbf{v}^{\mathrm{T}}) \tag{7.226}$$

$$= (I - \sigma\mathbf{v}\mathbf{v}^{\mathrm{T}})(I - \sigma\mathbf{v}\mathbf{v}^{\mathrm{T}})^{\mathrm{T}}. \tag{7.227}$$

Following the approach leading to Equation 7.220, the solution for the a posteriori generalized Cholesky factor $C_{(+)}$ of the covariance matrix P can be expressed as the product

$$C_{(+)}C_{(+)}^{\mathrm{T}} = P_{(+)} \tag{7.228}$$

$$= C_{(-)}(I - s\mathbf{v}\mathbf{v}^{\mathrm{T}})C_{(-)}^{\mathrm{T}} \tag{7.229}$$

$$= C_{(-)}(I - \sigma\mathbf{v}\mathbf{v}^{\mathrm{T}})(I - \sigma\mathbf{v}\mathbf{v}^{\mathrm{T}})^{\mathrm{T}}C_{(-)}^{\mathrm{T}}, \tag{7.230}$$

which can be factored as[15]

$$C_{(+)} = C_{(-)}(I - \sigma \mathbf{v}\mathbf{v}^{\mathrm{T}}) \tag{7.231}$$

with

$$\sigma = \frac{1 + \sqrt{1 - s|\mathbf{v}|^2}}{|\mathbf{v}|^2} \tag{7.232}$$

$$= \frac{1 + \sqrt{R/(R + |\mathbf{v}|^2)}}{|\mathbf{v}|^2}. \tag{7.233}$$

Equations 7.231 and 7.233 define the Potter "square-root" observational update formula, which is implemented in the accompanying MATLAB m-file `potter.m`. The Potter formula can be implemented *in place* (i.e., by overwriting C).

This algorithm updates the state estimate x and a generalized Cholesky factor C of P in place. This generalized Cholesky factor is a general $n \times n$ matrix. That is, it is not maintained in any particular form by the Potter update algorithm. The other "square-root" algorithms maintain C in triangular form.

7.7.1.4 Joseph-Stabilized Implementation This variant of the Kalman filter is due to Joseph [39], who demonstrated improved numerical stability by rearranging the standard formulas for the observational update (given here for scalar measurements) into the formats

$$\acute{z} = R^{-1/2} z, \tag{7.234}$$

$$\acute{H} = \acute{z} H, \tag{7.235}$$

$$\overline{K} = (\acute{H} P_{(-)} \acute{H}^{\mathrm{T}} + 1)^{-1} P_{(-)} \acute{H}^{\mathrm{T}}, \tag{7.236}$$

$$P_{(+)} = (I - \overline{K}\acute{H}) P_{(-)} (I - \overline{K}\acute{H})^{\mathrm{T}} + \overline{K}\,\overline{K}^{\mathrm{T}}, \tag{7.237}$$

taking advantage of partial results and the redundancy due to symmetry. The mathematical equivalence of Equation 7.237 to the conventional update formula for the covariance matrix was shown as Equation 5.18. This formula, by itself, does not uniquely define the Joseph implementation, however. As shown, it has $\sim n^3$ computational complexity.

Bierman Implementation This is a slight alteration due to Bierman [24] that reduces computational complexity by measurement decorrelation (if necessary) and the parsimonious use of partial results. The data flow diagram shown in Figure 7.9 is for a scalar measurement update, with data flow from top (inputs) to bottom (outputs) and showing all intermediate results. Calculations at the same level in this diagram

[15]Note that as $R \to \infty$ (no measurement), $\sigma \to 2/|v|^2$ and $I - \sigma v v^{\mathrm{T}}$ becomes a Householder matrix.

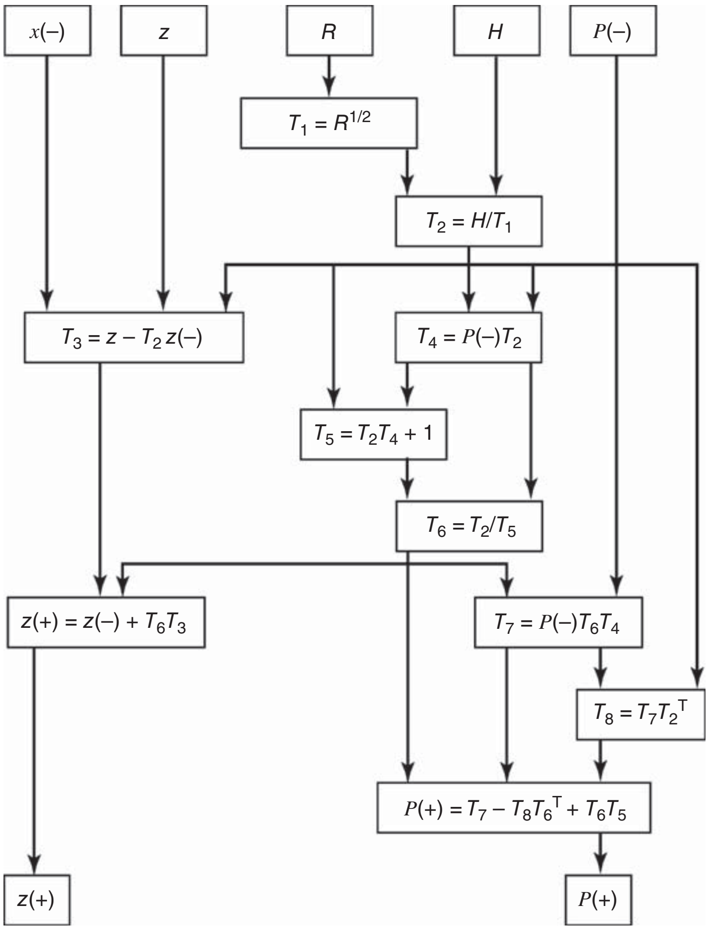

Figure 7.9 Data flow of Bierman–Joseph implementation.

may be implemented in parallel. Intermediate (temporary) results are labeled as $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_8$, where $\mathcal{T}_6 = \overline{K}$, the Kalman gain. If the result (left-hand side) of an $m \times m$ array is symmetric, then only the $\frac{1}{2}m(m+1)$ unique elements need be computed. Bierman [24] has made the implementation more memory efficient by the reuse of memory locations for these intermediate results. Bierman's implementation does not eliminate the redundant memory from symmetric arrays, however.

The computational complexity of this implementation grows as $3\ell n(3n+5)/2$ flops, where n is the number of components in the state vector and ℓ is the number of components in the measurement vector [24]. However, this formulation does require that R be a diagonal matrix. Otherwise, an additional computational complexity of $(4\ell^3 + \ell^2 - 10\ell + 3\ell^2 n - 3\ell n)/6$ flops for measurement decorrelation is incurred.

TABLE 7.23 De Vries–Joseph Implementation of Covariance Update

Operation	Complexity
	Without Using Decorrelation
$\mathcal{F}_1 = P_{(-)}H^{\mathrm{T}}$	ℓn^2
$\mathcal{F}_2 = H\mathcal{F}_1 + R$	$n\ell(\ell+1)/2$
$\mathcal{UDU}^{\mathrm{T}} = \mathcal{F}_2$	$\frac{1}{6}\ell(\ell+1)(\ell+2)$ (*UD* factorization)
$\mathcal{UDU}^{\mathrm{T}}K^{\mathrm{T}} = \mathcal{F}_1^{\mathrm{T}}$	$\ell^2 n$ [to solve for K]
$\mathcal{F}_3 = \frac{1}{2}K\mathcal{F}_2 - \mathcal{F}_1$	$\ell^2(n+1)$
$\mathcal{F}_4 = \mathcal{F}_3 K^{\mathrm{T}}$	ℓn^2
$P_{(+)} = P_{(-)} + \mathcal{F}_4 + \mathcal{F}_4^{\mathrm{T}}$	(included above)
Total	$\frac{1}{6}\ell^3 + \frac{3}{2}\ell^2 + \frac{1}{3}\ell + \frac{1}{2}\ell n + \frac{5}{2}\ell^2 n + 2\ell n^2$
	Using Decorrelation
Decorrelation ℓ repeats:	$\frac{2}{3}\ell^3 + \ell^2 - \frac{5}{3}\ell - \frac{1}{2}\ell n + \frac{1}{2}\ell^2 n$
	$\ell \times \{$
$\mathcal{F}_1 = P_{(-)}H^{\mathrm{T}}$	n^2
$\mathcal{F}_2 = H\mathcal{F}_1 + R$	n
$K = \mathcal{F}_1/\mathcal{F}_2$	n
$\mathcal{F}_3 = \frac{1}{2}K\mathcal{F}_2 - \mathcal{F}_1$	$n+1$
$\mathcal{F}_4 = \mathcal{F}_3 K^{\mathrm{T}}$	n^2
$P_{(+)} = P_{(-)} + \mathcal{F}_4 + \mathcal{F}_4^{\mathrm{T}}$	(included above) $\}$
Total	$\frac{2}{3}\ell^3 + \ell^2 - \frac{2}{3}\ell + \frac{5}{3}\ell n + \frac{1}{2}\ell^2 n + 2\ell n^2$

De Vries Implementation This implementation, which was shown to the authors by Thomas W. De Vries at Rockwell International, is designed to reduce the computational complexity of the Joseph formulation by judicious rearrangement of the matrix expressions and reuse of intermediate results. The fundamental operations—and their computational complexities—are summarized in Table 7.23.

Negative Evaluations of Joseph-Stabilized Implementation In comparative evaluations of several Kalman filter implementations on orbit estimation problems, Thornton and Bierman [14] found that the Joseph-stabilized implementation failed on some ill-conditioned problems for which "square-root" methods performed well.

7.7.2 Morf–Kailath Combined Observational/Temporal Update

The lion's share of the computational effort in Kalman filtering is spent in solving the Riccati equation. This effort is necessary for computing the Kalman gains. However, only the a priori value of the covariance matrix is needed for this purpose. Its a posteriori value is only used as an intermediate result on the way to computing the next a priori value.

Actually, it is not necessary to compute the a posteriori values of the covariance matrix explicitly. It is possible to compute the a priori values from one temporal epoch to the next temporal epoch, without going through the intermediate a posteriori values. This concept, and the methods for doing it, were introduced by Martin Morf and Thomas Kailath [40].

7.7.2.1 Combined Updates of Cholesky Factors The direct computation of $C_{P(k+1)(-)}$, the triangular Cholesky factor of $P_{k+1(-)}$, from $C_{P(k)(-)}$, the triangular Cholesky factor of $P_{k(-)}$, can be implemented by triangularization of the $(n+m)\times(p+n+m)$ partitioned matrix

$$A_k = \begin{bmatrix} GC_{Q(k)} & \Phi_k C_{P(k)} & 0 \\ 0 & H_k C_{P(k)} & C_{R(k)} \end{bmatrix}, \tag{7.238}$$

where $C_{R(k)}$ is a generalized Cholesky factor of R_k and $C_{Q(k)}$ is a generalized Cholesky factor of Q_k. Note that the $(n+m)\times(n+m)$ symmetric product

$$A_k A_k^{\mathrm{T}} = \begin{bmatrix} \Phi_k P_{k(-)} \Phi_k^{\mathrm{T}} + G_k Q_k G_k^{\mathrm{T}} & \Phi_k P_{k(-)} H_k^{\mathrm{T}} \\ H_k P_{k(-)} \Phi_k^{\mathrm{T}} & H_k P_{k(-)} H_k^{\mathrm{T}} \end{bmatrix}. \tag{7.239}$$

Consequently, if A_k is upper triangularized in the form

$$A_k T = C_k \tag{7.240}$$

$$= \begin{bmatrix} 0 & C_{P(k+1)} & \Psi_k \\ 0 & 0 & C_{E(k)} \end{bmatrix} \tag{7.241}$$

by an orthogonal transformation T, then the matrix equation

$$C_k C_k^{\mathrm{T}} = A_k A_k^{\mathrm{T}}$$

implies that the newly created block submatrices $C_{E(k)}$, Ψ_k, and $C_{P(k+1)}$ satisfy the equations

$$C_{E(k)} C_{E(k)}^{\mathrm{T}} = H_k P_{k(-)} H_k^{\mathrm{T}} + R_k \tag{7.242}$$

$$= E_k, \tag{7.243}$$

$$\Psi_k \Psi_k^{\mathrm{T}} = \Phi_k P_{k(-)} H_k^{\mathrm{T}} E_k^{-1} H_k P_{k(-)} \Phi_k, \tag{7.244}$$

$$\Psi_k = \Phi_k P_{k(-)} H_k^{\mathrm{T}} C_{E(k)}^{-1}, \tag{7.245}$$

$$C_{P(k+1)} C_{P(k+1)}^{\mathrm{T}} = \Phi_k P_{k(-)} \Phi_k^{\mathrm{T}} + G_k Q_k G_k^{\mathrm{T}} - \Psi_k \Psi_k^{\mathrm{T}} \tag{7.246}$$

$$= P_{k+1(-)}, \tag{7.247}$$

and the Kalman gain can be computed as

$$\overline{K}_k = \Psi_k C_{E(k)}^{-1}. \tag{7.248}$$

The computation of C_k from A_k can be done by Householder or Givens triangularization.

7.7.2.2 Combined Updates of UD Factors This implementation uses the *UD* factors of the covariance matrices *P, R*, and *Q*,

$$P_k = U_{P(k)} D_{P(k)} U_{P(k)}^{\mathrm{T}}, \tag{7.249}$$

$$R_k = U_{R(k)} D_{R(k)} U_{R(k)}^{\mathrm{T}}, \tag{7.250}$$

$$Q_k = U_{Q(k)} D_{Q(k)} U_{Q(k)}^{\mathrm{T}}, \tag{7.251}$$

in the partitioned matrices

$$B_k = \begin{bmatrix} GU_{Q(k)} & \Phi_k U_{P(k)} & 0 \\ 0 & H_k U_{P(k)} & U_{R(k)} \end{bmatrix}, \tag{7.252}$$

$$D_k = \begin{bmatrix} D_{Q(k)} & 0 & 0 \\ 0 & D_{P(k)} & 0 \\ 0 & 0 & D_{R(k)} \end{bmatrix}, \tag{7.253}$$

which satisfy the equation

$$B_k D_k B_k^{\mathrm{T}} = \begin{bmatrix} \Phi_k P_{k(-)} \Phi_k^{\mathrm{T}} + G_k Q_k G_k^{\mathrm{T}} & \Phi_k P_{k(-)} H_k^{\mathrm{T}} \\ H_k P_{k(-)} \Phi_k^{\mathrm{T}} & H_k P_{k(-)} H_k^{\mathrm{T}} \end{bmatrix}. \tag{7.254}$$

The MWGS orthogonalization of the rows of B_k with respect to the weighting matrix D_k will yield the matrices

$$\acute{B}_k = \begin{bmatrix} U_{P(k+1)} & U_{\Psi(k)} \\ 0 & U_{E(k)} \end{bmatrix}, \tag{7.255}$$

$$\acute{D}_k = \begin{bmatrix} D_{P(k+1)} & 0 \\ 0 & D_{E(k)} \end{bmatrix}, \tag{7.256}$$

where $U_{P(k+1)}$ and $D_{P(k+1)}$ are the *UD* factors of $P_{k+1(-)}$, and

$$U_{\Psi(k)} = \Phi_k P_{k(-)} H_k^{\mathrm{T}} U_{E(k)}^{-\mathrm{T}} D_{E(k)}^{-1} \tag{7.257}$$

$$= \overline{K}_k U_{E(k)}. \tag{7.258}$$

Consequently, the Kalman gain

$$\overline{K}_k = U_{\Psi(k)} U_{E(k)}^{-1} \tag{7.259}$$

can be computed as a by-product of the MWGS procedure, as well.

7.7.3 Information Filtering

7.7.3.1 Information Matrix of an Estimate The inverse of the covariance matrix of estimation uncertainty is called the *information matrix*:[16]

$$Y \overset{\text{def}}{=} P^{-1}. \tag{7.260}$$

Implementations using Y (or its generalized Cholesky factors) rather than P (or its generalized Cholesky factors) are called *information filters*. (Implementations using P are also called *covariance filters*.

7.7.3.2 Uses of Information Filtering

Problems without Prior Information Using the information matrix, one can express the idea that an estimation process may start with no a priori information whatsoever, expressed by

$$Y_0 = 0, \tag{7.261}$$

a matrix of zeros. An information filter starting from this condition will have absolutely no bias toward the a priori estimate. Covariance filters cannot do this.

One can also represent a priori estimates with no information in specified subspaces of state space by using information matrices with characteristic values equal to zero. In that case, the information matrix will have an eigenvalue–eigenvector decomposition of the form

$$Y_0 = \sum_i \lambda_i e_i e_i^{\mathrm{T}}, \tag{7.262}$$

where some of the eigenvalues $\lambda_i = 0$ and the corresponding eigenvectors e_i represent directions in state space with zero a priori information. Subsequent estimates will have no bias toward these components of the a priori estimate.

Information filtering cannot be used if P is singular, just as covariance filtering cannot be used if Y is singular. However, one may switch representations if both conditions do not occur simultaneously. For example, an estimation problem with zero initial information can be started with an information filter and then switched to a covariance implementation when Y becomes nonsingular. Conversely, a filtering problem with zero initial uncertainty may be started with a covariance filter, then switched to an information filter when P becomes nonsingular.

[16]This is also called the *Fisher information matrix*, named after the English statistician Ronald Aylmer Fisher (1890–1962). More generally, for distributions with differentiable probability density functions, the information matrix is defined as the matrix of second-order derivatives of the logarithm of the probability density with respect to the variates. For Gaussian distributions, this equals the inverse of the covariance matrix.

Robust Observational Updates The observational update of the *uncertainty matrix* is less robust against roundoff errors than the temporal update. It is more likely to cause the uncertainty matrix to become indefinite, which tends to destabilize the estimation feedback loop.

The observational update of the information matrix is more robust against roundoff errors. This condition is the result of a certain duality between information filtering and covariance filtering, by which the algorithmic structures of the temporal and observational updates are switched between the two approaches. The downside of this duality is that the temporal update of the information matrix is less robust than the observational update against roundoff errors and is a more likely cause of degradation. Therefore, information filtering may not be a panacea for all conditioning problems, but in those cases for which the observational update of the uncertainty matrix is the culprit, information filtering offers a possible solution to the roundoff problem.

Disadvantages of Information Filtering The greatest objection to information filtering is the loss of "transparency" of the representation. Although information is a more practical concept than uncertainty for some problems, it can be more difficult to interpret its physical significance and to use it in our thinking. With a little practice, it is relatively easy to visualize how σ (the square root of variance) is related to probabilities and to express uncertainties as "3σ" values. One must invert the information matrix before one can interpret its values in this way.

Perhaps the greatest impediment to widespread acceptance of information filtering is the loss of physical significance of the associated state vector components. These are linear combinations of the original state vector components, but the coefficients of these linear combinations change with the state of information/uncertainty in the estimates.

7.7.3.3 Information States Information filters do not use the same state vector representations as covariance filters. Those that use the information matrix in the filter implementation use the *information state*

$$d \overset{\text{def}}{=} Yx, \tag{7.263}$$

and those that use its generalized Cholesky factors C_Y such that

$$C_Y C_Y^{\mathrm{T}} = Y \tag{7.264}$$

use the "square-root" information state

$$s \overset{\text{def}}{=} C_Y x. \tag{7.265}$$

TABLE 7.24 Information Filter Equations

Observational update

$$\hat{d}_{k(+)} = \hat{d}_{k(-)} + H_k^{\mathrm{T}} R_k^{-1} z_k$$

$$Y_{k(+)} = Y_{k(-)} + H_k^{\mathrm{T}} R_k^{-1} H_k$$

Temporal update

$$A_k \overset{\mathrm{def}}{=} \Phi_k^{-\mathrm{T}} Y_{k(+)} \Phi_k^{-1}$$

$$Y_{k+1(-)} = \{I - A_k G_k [G_k^{\mathrm{T}} A_k G_k + Q_k^{-1}]^{-1} G_k^{\mathrm{T}}\} A_k$$

$$\hat{d}_{k+1(-)} = \{I - A_k G_k [G_k^{\mathrm{T}} A_k G_k + Q_k^{-1}]^{-1} G_k^{\mathrm{T}}\} \Phi_k^{-\mathrm{T}} \hat{d}_{k(+)}$$

7.7.3.4 Information Filter Implementation The implementation equations for the “straight” information filter (i.e., using Y, rather than its generalized Cholesky factors) are shown in Table 7.24. These can be derived from the Kalman filter equations and the definitions of the information matrix and information state. Note the similarities in form between these equations and the Kalman filter equations, with respective observational and temporal equations switched.

7.7.3.5 “Square-Root” Information Filtering The *“square-root” information filter* is usually abbreviated as SRIF. (The conventional “square-root” filter is often abbreviated as SRCF, which stands for *square-root covariance filter*.) Like the SRCF, the SRIF is more robust against roundoff errors than the “straight” form of the filter.

Historical Note: A complete formulation (i.e., including both updates) of the SRIF was developed by Dyer and McReynolds [11], using the “square-root” least-squares methods (triangularization) developed by Golub [41] and applied to sequential least-squares estimation by Lawson and Hanson [42]. The form developed by Dyer and McReynolds is shown in Table 7.25.

7.8 SUMMARY

Although Kalman filtering has been called “ideally suited to digital computer implementation” [43], the digital computer is not ideally suited to the task. The conventional implementation of the Kalman filter—in terms of covariance matrices—is particularly sensitive to roundoff errors.

TABLE 7.25 Square-Root Information Filter Using Triangularization

Observational update $\begin{bmatrix} C_{Y_{k(-)}} & H_k^{\mathrm{T}} C_{R_k^{-1}} \\ \hat{s}_{k(-)}^{\mathrm{T}} & z_k^{\mathrm{T}} C_{R_k^{-1}} \end{bmatrix} \mathrm{T}_{\mathrm{obs}} = \begin{bmatrix} C_{Y_{k(+)}} & 0 \\ \hat{s}_{k(+)}^{\mathrm{T}} & \epsilon \end{bmatrix}$

Temporal update

$$\begin{bmatrix} C_{Q_k^{-1}} & -G_k \Phi_k^{-\mathrm{T}} C_{Y_{k(+)}} \\ 0 & \Phi_k^{-\mathrm{T}} C_{Y_{k(+)}} \\ 0 & \hat{s}_{k(+)}^{\mathrm{T}} \end{bmatrix} \mathrm{T}_{\mathrm{temp}} = \begin{bmatrix} \Theta & 0 \\ \Gamma & C_{Y_{k+1(-)}} \\ \tau^{\mathrm{T}} & \hat{s}_{k+1(-)}^{\mathrm{T}} \end{bmatrix}$$

Note: T_{obs} and T_{temp} are orthogonal matrices (composed of Householder or Givens transformations), which lower triangularize the left-hand-side matrices. The submatrices other than s and C_Y on the right-hand sides are extraneous.

Many methods have been developed for decreasing the sensitivity of the Kalman filter to roundoff errors. The most successful approaches use alternative representations for the covariance matrix of estimation uncertainty, in terms of symmetric products of triangular factors. These fall into three general classes.

1. *"Square-root" covariance filters*, which use a decomposition of the covariance matrix of estimation uncertainty as a symmetric product of triangular Cholesky factors:
$$P = CC^{\mathrm{T}}.$$
2. *UD covariance filters*, which use a modified (square-root-free) Cholesky decomposition of the covariance matrix:
$$P = UDU^{\mathrm{T}}.$$
3. *"Square-root" information filters*, which use a symmetric product factorization of the information matrix, P^{-1}.

The alternative Kalman filter implementations use these factors of the covariance matrix (or it inverse) in three types of filter operations:

1. *temporal updates*,
2. *observational updates*, and
3. *combined updates* (temporal and observational).

The basic algorithmic methods used in these alternative Kalman filter implementations fall into four general categories. The first three of these categories of methods are concerned with decomposing matrices into triangular factors and maintaining the triangular form of the factors through all the Kalman filtering operations:

1. *Cholesky decomposition methods*, by which a symmetric positive-definite matrix M can be represented as symmetric products of a triangular matrix C:

$$M = CC^{\mathrm{T}} \text{ or } M = UDU^{\mathrm{T}}.$$

   The Cholesky decomposition algorithms compute C (or U and D), given M.

2. *Triangularization methods*, by which a symmetric product of a general matrix A can be represented as a symmetric product of a triangular matrix C:

$$AA^{\mathrm{T}} = CC^{\mathrm{T}} \text{ or } A\acute{D}A^{\mathrm{T}} = UDU^{\mathrm{T}}.$$

   These methods compute C (or U and D), given A (or A and $\acute{D}$).

3. *Rank 1 modification methods*, by which the sum of a symmetric product of a triangular matrix $\acute{C}$ and scaled symmetric product of a vector (rank 1 matrix) $\mathbf{v}$ can be represented by a symmetric product of a new triangular matrix C:

$$\acute{C}\acute{C}^{\mathrm{T}} + s\mathbf{v}\mathbf{v}^{\mathrm{T}} = CC^{\mathrm{T}} \text{ or } \acute{U}\acute{D}\acute{U}^{\mathrm{T}} + s\mathbf{v}\mathbf{v}^{\mathrm{T}} = UDU^{\mathrm{T}}.$$

These methods compute C (or U and D), given $\acute{C}$ (or $\acute{U}$ and $\acute{D}$), s, and $\mathbf{v}$.

The fourth category of methods includes standard matrix operations (multiplications, inversions, etc.) that have been specialized for triangular matrices.

These implementation methods have succeeded where the conventional Kalman filter implementation has failed.

It would be difficult to overemphasize the importance of good numerical methods in Kalman filtering. Limited to finite precision, computers will always make approximation errors. They are not infallible. One must always take this into account in problem analysis. The effects of roundoff may be thought to be minor, but overlooking them could be a major blunder.

PROBLEMS

7.1 An $n \times n$ Moler matrix M has elements

$$M_{ij} = \begin{cases} i & \text{if } i = j, \\ \min(i,j) & \text{if } i \neq j. \end{cases}$$

Calculate the 3×3 Moler matrix and its lower triangular Cholesky factor.

7.2 Write a MATLAB script to compute and print out the $n \times n$ Moler matrices and their lower triangular Cholesky factors for $2 \leq n \leq 20$.

7.3 Show that the condition number of a Cholesky factor C of $P = CC^{\mathrm{T}}$ is the square root of the condition number of P.

7.4 Show that if A and B are $n \times n$ upper triangular matrices, then their product AB is also upper triangular.

7.5 Show that a square triangular matrix is singular if and only if one of its diagonal terms is zero. (Hint: What is the determinant of a triangular matrix?)

7.6 Show that the inverse of an upper (lower) triangular matrix is also an upper (lower) triangular matrix.

7.7 Show that if the upper triangular Cholesky decomposition algorithm is applied to the matrix product

$$\begin{bmatrix} H & z \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} H & z \end{bmatrix} = \begin{bmatrix} H^{\mathrm{T}}H & H^{\mathrm{T}}z \\ z^{\mathrm{T}}H & z^{\mathrm{T}}z \end{bmatrix}$$

and the upper triangular result is similarly partitioned as $\begin{bmatrix} U & y \\ 0 & \varepsilon \end{bmatrix}$, then the solution $\hat{x}$ to the equation $U\hat{x} = y$ (which can be computed by back substitution) solves the least-squares problem $Hx \approx z$ with root-summed square residual $\|H\hat{x} - z\| = \varepsilon$ (Cholesky's method of least squares).

7.8 The *singular-value decomposition* of a symmetric, nonnegative-definite matrix P is a factorization $P = EDE^{\mathrm{T}}$ such that E is an orthogonal matrix and $D = \mathrm{diag}(d_1, d_2, d_3, \ldots, d_n)$ is a diagonal matrix with nonnegative elements $d_i \geq 0, 1 \leq i \leq n$. For $D^{1/2} = \mathrm{diag}(d_1^{1/2}, d_2^{1/2}, d_3^{1/2}, \ldots, d_n^{1/2})$, show that the symmetric matrix $C = ED^{1/2}E^{\mathrm{T}}$ is both a generalized Cholesky factor of P and a square root of P.

7.9 Show that the column vectors of the orthogonal matrix E in the singular-value decomposition of P (in the above exercise) are the characteristic vectors (eigenvectors) of P, and the corresponding diagonal elements of D are the respective characteristic values (eigenvalues). That is, for $1 \leq i \leq n$, if e_i is the ith column of E, show that $Pe_i = d_i e_i$.

7.10 Show that if $P = EDE^{\mathrm{T}}$ is a singular-value decomposition of P (defined above), then $P = \sum_{i=1}^{n} d_i e_i e_i^{\mathrm{T}}$, where e_i is the ith column vector of E.

7.11 Show that if C is an $n \times n$ generalized Cholesky factor of P, then, for any orthogonal matrix T, CT is also a generalized Cholesky factor of P.

7.12 Show that $(I - vv^{\mathrm{T}})^2 = (I - vv^{\mathrm{T}})$ if $|v|^2 = 1$ and that $(I - vv^{\mathrm{T}})^2 = I$ if $|v|^2 = 2$.

7.13 Show that the following formula generalizes the Potter observational update to include vector-valued measurements:

$$C_{(+)} = C_{(-)}[I - VM^{-\mathrm{T}}(M + F)^{-1}V^{\mathrm{T}}],$$

where $V = C_{(-)}^{\mathrm{T}}H^{\mathrm{T}}$ and F and M are generalized Cholesky factors of R and $R + V^{\mathrm{T}}V$, respectively.

7.14 Prove the following lemma: If W is an upper triangular $n \times n$ matrix such that

$$WW^{\mathrm{T}} = I - \frac{\mathbf{v}\mathbf{v}^{\mathrm{T}}}{R + |\mathbf{v}|^2},$$

then[17]

$$\sum_{k=m}^{j} W_{ik} W_{mk} = \Delta_{im} - \frac{\mathbf{v}_i \mathbf{v}_m}{R + \sum_{k=1}^{j} \mathbf{v}_k^2} \tag{7.266}$$

for all i, m, j such that $1 \le i \le m \le j \le n$.

7.15 Prove that the Björck "modified" Gram–Schmidt algorithm results in a set of mutually orthogonal vectors.

7.16 Suppose that

$$V = \begin{bmatrix} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{bmatrix},$$

where ϵ is so small that $1 + \epsilon^2$ (but not $1 + \epsilon$) rounds to 1 in machine precision. Compute the rounded result of Gram–Schmidt orthogonalization by the conventional and modified methods. Which result is closer to the theoretical value?

7.17 Show that if A and B are orthogonal matrices, then

$$\begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}$$

is an orthogonal matrix.

7.18 What is the inverse of the Householder reflection matrix $I - 2vv^{\mathrm{T}}/v^{\mathrm{T}}v$?

7.19 How many Householder transformations are necessary for triangularization of an $n \times q$ matrix when $n < q$? Does this change when $n = q$?

7.20 (Continuous temporal update of generalized Cholesky factors.) Show that all differentiable generalized Cholesky factors $C(t)$ of the solution $P(t)$ to the linear dynamic equation $\dot{P} = F(t)P(t) + P(t)F^{\mathrm{T}}(t) + G(t)Q(t)G^{\mathrm{T}}(t)$, where Q is symmetric, are solutions of a nonlinear dynamic equation $\dot{C}(t) = F(t)C(t) + \frac{1}{2}[G(T)Q(t)G^{\mathrm{T}}(t) + A(t)]C^{-\mathrm{T}}(t)$, where $A(t)$ is a skew-symmetric matrix [44].

7.21 Prove that the condition number of the information matrix is equal to the condition number of the corresponding covariance matrix in the case that neither of them is singular. (The condition number is the ratio of the largest characteristic value to the smallest characteristic value.)

[17]Kronecker's delta (Δ_{ij}) is defined to equal 1 only if its subscripts are equal ($i = j$) and to equal zero otherwise.

7.22 Prove the correctness of the triangularization equation for the observational update of the SRIF. (*Hint*: Multiply the partitioned matrices on the right by their respective transposes.)

7.23 Prove the correctness of the triangularization equation for the temporal update of the SRIF.

7.24 Prove to yourself that the conventional Kalman filter Riccati equation

$$P_{(+)} = P_{(-)} - P_{(-)}H^{\mathrm{T}}[HP_{(-)}H^{\mathrm{T}} + R]^{-1}HP_{(-)}$$

for the observational update is equivalent to the information form

$$P_{(+)}^{-1} = P_{(-)}^{-1} + H^{\mathrm{T}}R^{-1}H$$

of Peter Swerling. (*Hint:* Try multiplying the form for $P_{(+)}$ by the form for $P^{-1(+)}$ and see if it equals I, the identity matrix.)

7.25 Show that, if C is a generalized Cholesky factor of P (i.e., $P = CC^{\mathrm{T}}$), then $C^{-\mathrm{T}} = (C^{-1})^{\mathrm{T}}$ is a generalized Cholesky factor of $Y = P^{-1}$, provided that the inverse of C exists. Conversely, the *transposed* inverse of any generalized Cholesky factor of the information matrix Y is a generalized Cholesky factor of the covariance matrix P, provided that the inverse exists.

7.26 Write a MATLAB script to implement Example 5.8 using the Bierman–Thornton UD filter, plotting as a function of time the resulting root-mean-square (RMS) estimation uncertainty values of $P_{(+)}$ and $P_{(-)}$ and the components of $\overline{K}$. (You can use the scripts `bierman.m` and `thornton.m`, but you will have to compute UDU^{T} and take the square roots of its diagonal values to obtain RMS uncertainties.)

7.27 Write a MATLAB script to implement Example 5.8 using the Potter “square-root” filter and plotting the same values as in the problem above.

REFERENCES

[1] L. Strachey, *Eminent Victorians*, Penguin Books, London, 1988.

[2] J. S. Meditch, “A survey of data smoothing for linear and nonlinear dynamic systems,” *Automatica*, Vol. 9, pp. 151–162, 1973.

[3] S. R. McReynolds, “Fixed interval smoothing: revisited,” *AIAA Journal of Guidance, Control, and Dynamics*, Vol. 13, pp. 913–921, 1990.

[4] G. J. Bierman, “A new computationally efficient fixed-interval discrete time smoother,” *Automatica*, Vol. 19, pp. 503–561, 1983.

[5] K. Watanabe and S. G. Tzafestas, “New computationally efficient formula for backward-pass fixed interval smoother and its UD factorization algorithm,” *IEE Proceedings*, Vol. 136D, pp. 73–78, 1989.

[6] J. M. Jover and T. Kailath, "A parallel architecture for Kalman filter measurement update and parameter update," *Automatica*, Vol. 22, pp. 783–786, 1986.

[7] ANSI/IEEE Std. 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic*, Institute of Electrical and Electronics Engineers, New York, 1985.

[8] L. A. McGee and S. F. Schmidt, *Discovery of the Kalman Filter as a Practical Tool for Aerospace and Industry*, Technical Memorandum 86847, National Aeronautics and Space Administration, Mountain Veiw, CA, 1985.

[9] S. F. Schmidt, "The Kalman filter: its recognition and development for aerospace applications," *AIAA Journal of Guidance and Control*, Vol. 4, No. 1, pp. 4–8, 1981.

[10] R. H. Battin, "Space guidance evolution—a personal narrative," *AIAA Journal of Guidance and Control*, Vol. 5, pp. 97–110, 1982.

[11] P. Dyer and S. McReynolds, "Extension of square-root filtering to include process noise," *Journal of Optimization Theory and Applications*, Vol. 3, pp. 444–458, 1969.

[12] M. Verhaegen and P. Van Dooren, "Numerical aspects of different Kalman filter implementations," *IEEE Transactions on Automatic Control*, Vol. AC-31, pp. 907–917, 1986.

[13] J. E. Potter, "Matrix quadratic solutions," *SIAM Journal of Applied Mathematics*, Vol. 14, pp. 496–501, 1966.

[14] C. L. Thornton and G. J. Bierman, *A Numerical Comparison of Discrete Kalman Filtering Algorithms: An Orbit Determination Case Study*, JPL Technical Memorandum 33-771, NASA/JPL, Pasadena, CA, 1976.

[15] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, *LINPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.

[16] G. E. Forsythe, M. A. Malcolm, and C. B. Moler, *Computer Methods for Mathematical Computations*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[17] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed., Johns Hopkins University Press, Baltimore, MD, 2013.

[18] P. G. Kaminski, Square root filtering and smoothing for discrete processes, PhD thesis, Stanford University, 1971.

[19] W. M. Gentleman, "Least squares computations by Givens transformations without square roots," *Journal of the Institute for Mathematical Applications*, Vol. 12, pp. 329–336, 1973.

[20] W. Givens, "Computation of plane unitary rotations transforming a general matrix to triangular form," *Journal of the Society for Industrial and Applied Mathematics*, Vol. 6, pp. 26–50, 1958.

[21] A. S. Householder, "Unitary triangularization of a nonsymmetric matrix," *Journal of the Association for Computing Machinery*, Vol. 5, pp. 339–342, 1958.

[22] N. A. Carlson, "Fast triangular formulation of the square root filter," *AIAA Journal*, Vol. 11, No. 9, pp. 1259–1265, 1973.

[23] W. S. Agee and R. H. Turner, *Triangular Decomposition of a Positive Definite Matrix Plus a Symmetric Dyad, with Applications to Kalman Filtering*, Technical Report No. 38, White Sands Missile Range Oct. 1972.

[24] G. J. Bierman, *Factorization Methods for Discrete Sequential Estimation*, Academic Press, New York, 1977.

[25] C. L. Thornton, Triangular covariance factorizations for Kalman filtering, PhD thesis, University of California at Los Angeles, 1976.

[26] Å. Björck, “Solving least squares problems by orthogonalization,” *BIT*, Vol. 7, pp. 1–21, 1967.

[27] M. S. Grewal and J. Kain, “Kalman filter implementation with improved numerical properties,” *IEEE Transactions on Automatic Control,* Vol. 55, No. 9, pp. 2058–2068, 2010.

[28] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vettering, *Numerical Recipes in C++: The Art of Scientific Computing*, Cambridge University Press, Cambridge, 2007.

[29] L. F. Richardson, “The approximate arithmetical solution by finite differences of physical problems including differential equations, with an application to the stresses in a masonry dam,” *Philosophical Transactions of the Royal Society A*, Vol. 210, No. 459–470, pp. 307–357, 1911.

[30] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, 3rd ed., Springer-Verlag, New York, 2002.

[31] C. Moler and C. Van Loan, “Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later,” *SIAM Review*, Vol. 45, No. 1, pp. 3–49, 2003.

[32] C. F. Van Loan, “Computing integrals involving the matrix exponential,” *IEEE Transactions on Automatic Control*, Vol. AC-23, pp. 395–404, 1978.

[33] P. G. Kaminski, A. E. Bryson Jr., and S. F. Schmidt, “Discrete square root filtering: a survey of current techniques,” *IEEE Transactions on Automatic Control*, Vol. AC-16, pp. 727–736, 1971.

[34] P. Swerling, “First order error propagation in a stagewise differential smoothing procedure for satellite observations,” *Journal of Astronautical Sciences*, Vol. 6, pp. 46–52, 1959.

[35] V. Strassen, “Gaussian elimination is not optimal,” *Numerische Matematik*, Vol. 13, p. 354, 1969.

[36] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *ASME Journal of Basic Engineering*, Vol. 82, pp. 34–45, 1960.

[37] R. H. Battin, *Astronautical Guidance*, McGraw-Hill, New York, 1964.

[38] J. E. Potter and R. G. Stern, “Statistical filtering of space navigation measurements,” in *Proceedings of the 1963 AIAA Guidance and Control Conference*, AIAA, New York, pp. 333-1–333-13, 1963.

[39] R. S. Bucy and P. D. Joseph, *Filtering for Stochastic Processes with Applications to Guidance*, American Mathematical Society, Chelsea Publishing, Providence, RI, 2005.

[40] M. Morf and T. Kailath, “Square root algorithms for least squares estimation,” *IEEE Transactions on Automatic Control*, Vol. AC-20, pp. 487–497, 1975.

[41] G. H. Golub, “Numerical methods for solving linear least squares problems,” *Numerische Mathematik*, Vol. 7, pp. 206–216, 1965.

[42] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[43] A. Gelb, J. F. Kasper Jr., R. A. Nash Jr., C. F. Price, and A. A. Sutherland Jr., *Applied Optimal Estimation*, MIT Press, Cambridge, MA, 1974.

[44] A. Andrews, “A square root formulation of the Kalman covariance equations,” *AIAA Journal*, Vol. 6, pp. 1165–1166, 1968.