Module 29.2: nag_tsa_kalman
Kalman Filtering

`nag_tsa_kalman` contains procedures for Kalman filters.

Contents

Introduction

1 Theoretical Background

Kalman filtering can be used for estimating or filtering a multi-dimensional stochastic process X_i on which observations Y_i are made (see, for example, Anderson and Moore [1] and Wei [4]).

The technique assumes that X_i and Y_i obey the linear system

$$X_{i+1} = A_i X_i + B_i W_i$$

$$Y_i = C_i X_i + V_i$$

where X_i is the state vector to be estimated, Y_i is the measurement vector, W_i is the state noise, V_i is the measurement noise, A_i is the state transition matrix, B_i is the noise coefficient matrix and C_i is the measurement coefficient matrix (where the subscript i refers to the appropriate quantity at time i). The state noise and the measurement noise are assumed to be uncorrelated and have zero mean. This implies that

$$E\{W_i\} = 0, \quad E\{V_i\} = 0 \text{ and } E\{W_i V_i^T\} = 0$$

and the covariance matrices are

$$E\{W_i W_i^T\} = Q_i \text{ and } E\{V_i V_i^T\} = R_i,$$

where E denotes the expectation operator, Q_i and R_i are symmetric positive definite matrices.

If the system matrices A_i, B_i, C_i and the covariance matrices Q_i, R_i are known then Kalman filtering can be used to compute the minimum variance estimate of the stochastic variable X_i.

The estimate of X_i given observations Y_1 to Y_{i-1} is denoted by $\hat{X}_{i|i-1}$ with state covariance matrix $P_{i|i-1}$ while the estimate of X_i given observations Y_1 to Y_i is denoted by $\hat{X}_{i|i}$ with covariance matrix $P_{i|i}$. The update of the estimate, $\hat{X}_{i+1|i}$, from time i to time $i+1$, is computed in two stages.

First, the *update equations* are:

$$\hat{X}_{i|i} = \hat{X}_{i|i-1} + K_i r_i, \qquad P_{i|i} = [I - K_i C_i] P_{i|i-1},$$

where the residual $r_i = Y_i - C_i \hat{X}_{i|i-1}$ has an associated covariance matrix $H_i = C_i P_{i|i-1} C_i^T + R_i$, and the Kalman gain matrix $K_i = P_{i|i-1} C_i^T H_i^{-1}$.

The second stage is the one-step-ahead *prediction equations* given by:

$$\hat{X}_{i+1|i} = A_i \hat{X}_{i|i}, \qquad P_{i+1|i} = A_i P_{i|i} A_i^T + B_i Q_i B_i^T.$$

These two stages can be combined to give the one step-ahead *update-prediction* equations:

$$\hat{X}_{i+1|i} = A_i \hat{X}_{i|i-1} + A_i K_i r_i.$$

The above equations thus provide a method for recursively calculating the estimates of the state vectors $\hat{X}_{i|i}$ and $\hat{X}_{i+1|i}$ and their covariance matrices $P_{i|i}$ and $P_{i+1|i}$ from their previous values. The initial values $\hat{X}_{1|0}$ and $P_{1|0}$ are required to start the recursion. For stationary systems $P_{1|0}$ can be computed from the equation

$$P_{1|0} = A_1 P_{1|0} A_1^T + B_1 Q_1 B_1^T.$$

For $\hat{X}_{1|0}$ the value $E\{X\}$ can be used if it is available.

2 Computational Background

To improve the stability of the computations the square root algorithm is used. One recursion of the square root covariance filter algorithm can be summarized as follows (see Vanbegin *et al.* [2]):

$$\begin{pmatrix} R_i^{1/2} & C_iS_i & 0 \\ \\ 0 & A_iS_i & B_iQ_i^{1/2} \end{pmatrix} U = \begin{pmatrix} H_i^{1/2} & 0 & 0 \\ \\ G_i & S_{i+1} & 0 \end{pmatrix},$$

where U is an orthogonal transformation triangularizing the left-hand pre-array to produce the right-hand post-array, S_i is the lower triangular Cholesky factor of the state covariance matrix $P_{i+1|i}$, $Q_i^{1/2}$ and $R_i^{1/2}$ are the lower triangular Cholesky factors of the covariance matrices Q and R, and $H^{1/2}$ is the lower triangular Cholesky factor of the covariance matrix of the residuals.

The relationship between the Kalman gain matrix, K_i, and G_i is given by

$$A_iK_i = G_i \left(H_i^{1/2}\right)^{-1}.$$

To improve the efficiency of the computations when the matrices A_i, B_i and C_i do not vary with time the system can be transformed to give a simpler structure; the transformed state vector is U^*X where U^* is the transformation that reduces the matrix pair (A, C) to lower observer Hessenberg form. That is, the matrix U^* is computed such that the compound matrix

$$\begin{bmatrix} CU^{*T} \\ U^*AU^{*T} \end{bmatrix}$$

is a lower trapezoidal matrix. Further, the transformed matrix U^*B is used in place of the untransformed matrix B. These transformations need only be computed once at the start of a series. Note that the covariance matrices Q_i and R_i can be time varying.

3 Model Estimation Using the Kalman Filter

If the state space model contains unknown parameters, θ, these can be estimated using maximum likelihood. For normal variates the log-likelihood given observations $Y_i, i = 1, 2, \ldots, t$, is

$$L(Y;\theta) = \kappa - \frac{1}{2}\sum_{i=1}^{t} \ln(\det(H_i)) - \frac{1}{2}\sum_{i=1}^{t} r_i^T H_i^{-1} r_i$$

where κ is a constant, $Y = \{Y_1, \ldots, Y_t\}$ and the other symbols have been defined earlier.

Optimal estimates for the unknown model parameters θ can then be obtained by using a suitable optimizer procedure to maximise the likelihood function.

4 Forecasting

Once a state space model has been constructed and the values $\hat{X}_{T|T-1}$ and $P_{T|T-1}$ at time $t = T$ have been obtained, then the L-step-ahead forecasts can be computed by missing out the Kalman filter update equations and setting $\hat{X}_{t|t} = \hat{X}_{t|t-1}$ and $P_{t|t} = P_{t|t-1}$. This yields the following forecasts for the state vector:

$$\begin{aligned} \hat{X}_{T+1|T} &= A_T\hat{X}_{T|T-1} \\ \hat{X}_{T+2|T} &= A_{T+1}A_T\hat{X}_{T|T-1} \\ &\vdots \\ \hat{X}_{T+L|T} &= \left[\prod_{j=0}^{L-1} A_{T+j}\right]\hat{X}_{T|T-1} \end{aligned}$$

and the forecasts, $\hat{Y}_{T+L|T}$, can be obtained from

$$\begin{aligned}\hat{Y}_{T+L|T} &= C_{T+L}\hat{X}_{T+L|T}\\ \hat{Y}_{T+L|T} &= C_{T+L}\left[\prod_{j=0}^{L-1} A_{T+j}\right]\hat{X}_{T|T-1}.\end{aligned}$$

For a time invariant system these forecasts take the simplified form:

$$\begin{aligned}\hat{X}_{T+L|T} &= A^L\hat{X}_{T|T-1}\\ \hat{Y}_{T+L|T} &= CA^L\hat{X}_{T|T-1}.\end{aligned}$$

The forecast state covariance vector is

$$\hat{P}_{T+L|T} = A^L\hat{P}_{T|T-1}\left(A^L\right)^T + \sum_{j=0}^{L-1} A^jBQB^T\left(A^j\right)^T.$$

Missing data, Y_i, can be dealt with in a similar way to that used for forecasting.

5 Kalman Filter and Time Series Models

Many commonly used time series models can be written as a state space model. Thus the Kalman filter can be used in computing the likelihood when fitting the model, computing residuals for model checking and finally for producing forecasts.

The auto-regressive moving average (ARMA) model is described in the Chapter Introduction. A univariate ARMA(p,q) model can be cast into the following state space form:

$$\begin{aligned}x_t &= Ax_{t-1} + B\epsilon_t\\ w_t &= Cx_t\end{aligned}$$

where

$$A = \begin{pmatrix} \phi_1 & 1 & & & \\ \phi_2 & & 1 & & \\ \vdots & & & \ddots & \\ \phi_{r-1} & & & & 1 \\ \phi_r & 0 & 0 & \cdots & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 1 \\ -\theta_1 \\ -\theta_2 \\ \vdots \\ -\theta_{r-1} \end{pmatrix}, \quad C^T = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

where $r = \max(p, q+1)$.

The representation for a k-variate ARMA(p,q) series (VARMA) is very similar to that given above, except that now the state vector is of length kr and the ϕ and θ are now $k \times k$ matrices and the 1 in A, B and C are now identity matrices of order k. If $p < r$ or $q+1 < r$ then the appropriate ϕ or θ matrices are set to zero.

Since the compound matrix

$$\begin{bmatrix} C \\ A \end{bmatrix}$$

is already in lower observer Hessenberg form (i.e., it is lower trapezoidal with zeros in the top right-hand triangle) the invariant Kalman filter algorithm can be used directly without the need to generate a transformation matrix U^*.

The state space equations for the basic structural model as described in the Chapter Introduction are:

$$\begin{aligned}x_t &= Ax_{t-1} + w_t\\ y_t &= Cx_t + \epsilon_t.\end{aligned}$$

For $s = 4$ (quarterly data) the dummy variable form has

$$A = \begin{pmatrix} 1 & 1 & \vdots & & & \\ 0 & 1 & \vdots & & \mathbf{0} & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ & & \vdots & -1 & -1 & -1 \\ & \mathbf{0} & \vdots & 1 & 0 & 0 \\ & & \vdots & 0 & 1 & 0 \end{pmatrix}, \quad w_t = \begin{pmatrix} \eta_t \\ \zeta_t \\ \cdots \\ \omega_t \\ 0 \\ 0 \end{pmatrix}, \quad x_t = \begin{pmatrix} \mu_t \\ \beta_t \\ \cdots \\ \gamma_t \\ \gamma_{t-1} \\ \gamma_{t-2} \end{pmatrix}, \quad C^T = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix},$$

where the third element, γ_t, in the state vector x_t represents the current seasonal effect.

In the equivalent trigonometric seasonal model the parameters are

$$A = \begin{pmatrix} 1 & 1 & \vdots & & & \vdots & \\ 0 & 1 & \vdots & & \mathbf{0} & \vdots & \mathbf{0} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ & & \vdots & 0 & 1 & \vdots & -1 \\ & \mathbf{0} & \vdots & -1 & 0 & \vdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ & \mathbf{0} & \vdots & & \mathbf{0} & \vdots & -1 \end{pmatrix}, \quad w_t = \begin{pmatrix} \eta_t \\ \zeta_t \\ \cdots \\ \omega_{1,t} \\ \omega^*_{1,t} \\ \cdots \\ \omega_{2,t} \end{pmatrix}, \quad x_t = \begin{pmatrix} \mu_t \\ \beta_t \\ \cdots \\ \gamma_{1,t} \\ \gamma^*_{1,t} \\ \cdots \\ \gamma_{2,t} \end{pmatrix}, \quad C^T = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}.$$

Procedure: nag_kalman_init

1 Description

For the state space model

$$X_{i+1} = A_i X_i + B_i W_i$$

where X_i is the state vector of length n, W_i is the state noise vector of length l, A_i is the n by n state transition matrix and B_i is the n by l noise coefficient matrix, this procedure provides an initial estimate of the state covariance matrix $P_{i|i-1} = \mathrm{var}(\hat{X}_{i|i-1}), P_{1|0}$ by solving the Lyapunov equation $P_{1|0} = A_1 P_{1|0} A_1^T + B_1 Q_1 B_1^T$ where Q_i is the covariance matrix of the state noise.

2 Usage

```
USE nag_tsa_kalman
CALL nag_kalman_init(a, b, s  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$n > 0$ — the dimension of the state vector

$l > 0$ — the dimension of the state noise vector

3.1 Mandatory Arguments

a(n, n) — real(kind=*wp*), intent(in)

Input: the initial state transition matrix, A_1.

b(n, l) — real(kind=*wp*), intent(in)

Input: the initial noise coefficient matrix, B_1.

s(n, n) — real(kind=*wp*), intent(out)

Output: the lower triangular Cholesky factor of the initial estimate for the state covariance matrix, $P_{1|0}$.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

q(l, l) — real(kind=*wp*), intent(inout), optional

Input: the initial state noise covariance matrix, Q_1.

Output: the lower triangular Cholesky factor of Q_1.

Default: if q is not present then it is assumed to be the identity matrix.

Constraints: q must be positive definite.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	Cannot compute an initial estimate. The system is not stationary since the absolute value of each eigenvalue of the transition matrix, `a`, is not less than 1.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

Procedure: nag_kalman_predict

1 Description

For the state space model

$$X_{i+1} = A_i X_i + B_i W_i$$

$$Y_i = C_i X_i + V_i$$

where X_i is the state vector, Y_i is the measurement vector, W_i is the state noise (with covariance matrix Q_i), V_i is the measurement noise, A_i is the state transition matrix, B_i is the noise coefficient matrix and C_i is the measurement coefficient matrix, this procedure calculates a prediction step using a square root covariance Kalman filter. The predicted state vector for time i given observations up to time $i-1$ is denoted by $\hat{X}_{i|i-1}$ with associated state covariance matrix $P_{i|i-1} = \mathrm{var}(\hat{X}_{i|i-1})$. The procedure computes S_i from S_{i-1}, where S_i is the lower triangular Cholesky factor of $P_{i|i-1}$, and optimally computes $\hat{X}_{i+1|i}$ and $\hat{Y}_{i+1}$ from $\hat{X}_{i|i-1}$.

2 Usage

```
USE nag_tsa_kalman
CALL nag_kalman_predict(s, a, b  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$n > 0$ — the dimension of the state vector
$m > 0$ — the dimension of the observation vector
$l > 0$ — the dimension of the state noise vector

3.1 Mandatory Arguments

s(n, n) — real(kind=*wp*), intent(inout)

Input: the lower triangular Cholesky factor, S_i, of the state covariance matrix.

Output: the prediction of the lower triangular Cholesky factor, S_{i+1}, of the state covariance matrix.

Constraints: the diagonal elements of s must be non-negative.

a(n, n) — real(kind=*wp*), intent(in)

Input: the state transition matrix, A_i.

b(n, l) — real(kind=*wp*), intent(in)

Input: the noise coefficient matrix, B_i.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

q(l, l) — real(kind=*wp*), intent(in), optional

Input: the lower triangular Cholesky factor of the state noise covariance matrix, Q_i.

Default: if `q` is not present then it is assumed to be the identity matrix.

Constraints: the diagonal elements of `q` must be non-negative.

x(n) — real(kind=*wp*), intent(inout), optional

Input: the estimated state vector, $\hat{X}_{i|i-1}$.

Output: the estimated state vector, $\hat{X}_{i+1|i}$.

c(m, n) — real(kind=*wp*), intent(in), optional

Input: the measurement coefficient matrix, C_i.

Constraints: if `c` is present then arguments `x` and `y` must also be present.

y(m) — real(kind=*wp*), intent(out), optional

Output: the forecast observation vector, $\hat{Y}_i$.

Constraints: if `y` is present then arguments `x` and `c` must also be present.

p(n, n) — real(kind=*wp*), intent(out), optional

Output: the covariance matrix, $P_{i+1|i}$, associated with the computed state vector, $\hat{X}_{i+1|i}$.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Mathematical Background

The one step-ahead prediction equations are:

$$\begin{array}{lcl} \hat{X}_{i+1|i} & = & A_i \hat{X}_{i|i} \\ \hat{Y}_{i+1} & = & C_{i+1} X_{i+1|i} \\ P_{i+1|i} & = & A_i P_{i|i} A_i^T + B_i Q_i B_i^T. \end{array}$$

The procedure can be used to provide estimates for missing data, Y_i, and can also be used to obtain an initial value $S_{1|0}$, via the steady-state solution of the equation $P_{i+1|i} = A_1 P_{i|i-1} A_1^T + B_1 Q_i B_1^T$, where at steady-state $S_{i+1|i} S_{i+1|i}^T = P_{i+1|i} = P_{i|i-1} = P_{1|0}$.

6.2 Algorithmic Detail

The procedure performs one recursion of the square root covariance filter algorithm, summarized as follows:

$$\left(\begin{array}{cc} A_i S_i & B_i Q_i^{1/2} \end{array} \right) U = \left(\begin{array}{cc} S_{i+1} & 0 \end{array} \right),$$

where U is an orthogonal transformation triangularizing the left-hand pre-array to produce the right-hand post-array.

6.3 Accuracy

The use of the square root algorithm improves the stability of the computations as compared with the direct coding of the Kalman filter. The accuracy will depend on the model.

Procedure: nag_kalman_sqrt_cov_var

1 Description

For the state space model

$$X_{i+1} = A_i X_i + B_i W_i$$

$$Y_i = C_i X_i + V_i$$

where X_i is the state vector, Y_i is the measurement vector, W_i is the state noise (with covariance matrix Q_i), V_i is the measurement noise (with covariance matrix R_i), A_i is the state transition matrix, B_i is the noise coefficient matrix and C_i is the measurement coefficient matrix, this procedure calculates a combined update-prediction step using a time-varying square root covariance Kalman filter.

2 Usage

```
USE nag_tsa_kalman
CALL nag_kalman_sqrt_cov_var(s, a, b, c, r  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathtt{x}(n)$' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$n > 0$ — the dimension of the state vector

$m > 0$ — the dimension of the observation vector

$l > 0$ — the dimension of the state noise vector

3.1 Mandatory Arguments

s(n, n) — real(kind=*wp*), intent(inout)

Input: the lower triangular Cholesky factor, S_i, of the state covariance matrix at time i.

Output: the lower triangular Cholesky factor, S_{i+1}, of the state covariance matrix at time $i + 1$.

Constraints: the diagonal elements of s must be non-negative.

a(n, n) — real(kind=*wp*), intent(in)

Input: the state transition matrix, A_i.

b(n, l) — real(kind=*wp*), intent(in)

Input: the noise coefficient matrix, B_i.

c(m, n) — real(kind=*wp*), intent(in)

Input: the measurement coefficient matrix, C_i.

r(m, m) — real(kind=*wp*), intent(in)

Input: the lower triangular Cholesky factor of the measurement noise covariance matrix, R_i.

Constraints: the diagonal elements of r must be non-negative.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

q(l, l) — real(kind=*wp*), intent(in), optional

Input: the lower triangular Cholesky factor of the state noise covariance matrix, Q_i.

Default: if `q` is not present then it is assumed to be the identity matrix.

Constraints: the diagonal elements of `q` must be non-negative.

h(m, m) — real(kind=*wp*), intent(out), optional

Output: the lower triangular Cholesky factor of the covariance matrix for the computed residuals, H_i.

p(n, n) — real(kind=*wp*), intent(out), optional

Output: the covariance matrix, $P_{i+1|i}$, associated with the computed state vector, $\hat{X}_{i+1|i}$.

k(n, m) — real(kind=*wp*), intent(out), optional

Output: the Kalman gain matrix, K_i, premultiplied by the state transition matrix, A_i; i.e., A_iK_i.

tol — real(kind=*wp*), intent(in), optional

Input: the tolerance used to test for singularity of the matrix H_i. The inverse of the condition number of the lower triangular matrix `h` is estimated. If this estimate is less than `tol`, then H_i is assumed to be singular.

Note: if $0.0 \leq$ `tol` $\leq m^2 \times$ `EPSILON(1.0_`*wp*`)` then the default value is used.

Default: `tol` $= m^2 \times$ `EPSILON(1.0_`*wp*`)`.

x(n) — real(kind=*wp*), intent(inout), optional

Input: the estimated state vector, $\hat{X}_{i|i-1}$.

Output: the estimated state vector, $\hat{X}_{i+1|i}$.

Constraints: if `x` is present then the argument `y` must also be present.

y(m) — real(kind=*wp*), intent(in), optional

Input: the observation vector, Y_i.

Constraints: if `y` is present then argument `x` must also be present.

resid(m) — real(kind=*wp*), intent(out), optional

Output: the calculated residuals r_i, $r_i = Y_i - C_i\hat{X}_{i|i-1}$.

Constraints: if `resid` is present then the arguments `x` and `y` must also be present.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.

305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

Warnings (error%level = 1):

error%code	**Description**
201	The Cholesky factor of matrix H_i is singular. The singularity of the Cholesky factor of matrix H_i means that the procedure is not able to return values for the either Kalman gain matrix K_i, or the predicted state vector $\hat{X}_{i+1\|i}$.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Mathematical Background

For models with time-varying A, B and C where (A, C) is already in Hessenberg form, the procedure `nag_kalman_sqrt_cov_invar` should be used with `call_type` = `'N'` or `'n'`.

The Cholesky factors of the covariance matrices can be computed using `nag_sym_lin_fac`.

Note that the model

$$\begin{array}{lll} X_{i+1} & = & A_iX_i + W_i, \quad \mathrm{var}(W_i) = Q_i \\ Y_i & = & C_iX_i + V_i, \quad \mathrm{var}(V_i) = R_i \end{array}$$

can be specified either with the argument `q` $= Q^{1/2}$ and B set to the identity matrix or with $B = Q^{1/2}$ and the argument `q` not present.

If W_i and V_i are independent multivariate Normal variates then the log-likelihood for observations $i = 1, 2, \ldots, t$ is given by

$$L(\theta) = \kappa - \frac{1}{2}\sum_{i=1}^{t} \ln(\det(H_i)) - \frac{1}{2}\sum_{i=1}^{t} r_i^T H_i^{-1} r_i,$$

where κ is a constant and H_i is the covariance matrix for the residuals, $r_i = Y_i - C_i\hat{X}_{i|i-1}$.

6.2 Algorithmic Detail

The procedure performs one recursion of the square root covariance filter algorithm, summarized as follows:

$$\left(\begin{array}{ccc} R_i^{1/2} & C_iS_i & 0 \\ & & \\ 0 & A_iS_i & B_iQ_i^{1/2} \end{array}\right) U = \left(\begin{array}{ccc} H_i^{1/2} & 0 & 0 \\ & & \\ G_i & S_{i+1} & 0 \end{array}\right),$$

where U is an orthogonal transformation triangularizing the left-hand pre-array to produce the right-hand post-array, S_i is the lower triangular Cholesky factor of the state covariance matrix $P_{i+1|i}$, $Q_i^{1/2}$ and $R_i^{1/2}$ are the lower triangular Cholesky factors of the covariance matrices Q and R, and $H^{1/2}$ is the lower triangular Cholesky factor of the covariance matrix of the residuals (see Vanbegin *et al.* [2]).

The relationship between the Kalman gain matrix, K_i, and G_i is given by

$$A_iK_i = G_i\left(H_i^{1/2}\right)^{-1}.$$

The algorithm requires $\frac{7}{6}n^3 + n^2\left(\frac{5}{2}m + l\right) + n\left(\frac{1}{2}l^2 + m^2\right)$ operations and is backward stable (Verhaegen and Van Dooren [3]).

6.3 Accuracy

The use of the square root algorithm improves the stability of the computations as compared with the direct coding of the Kalman filter. The accuracy will depend on the model.

Procedure: nag_kalman_sqrt_cov_invar

1 Description

For the state space model

$$X_{i+1} = AX_i + BW_i$$

$$Y_i = CX_i + V_i$$

where X_i is the state vector, Y_i is the measurement vector, W_i is the state noise (with covariance matrix Q_i), V_i is the measurement noise (with covariance matrix R_i), A is the state transition matrix, B is the noise coefficient matrix and C is the measurement coefficient matrix, this procedure calculates a combined update-prediction step using a time-invariant square root covariance Kalman filter.

2 Usage

```
USE nag_tsa_kalman
CALL nag_kalman_sqrt_cov_invar(st, at, bt, ct, r  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '`x`(n)' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$n > 0$ — the dimension of the state vector

$m > 0$ — the dimension of the observation vector

$l > 0$ — the dimension of the state noise vector

3.1 Mandatory Arguments

st(n, n) — real(kind=*wp*), intent(inout)

Input:

If `call_type` = `'N'`, `'n'`, `'F'` or `'f'`, the lower triangular Cholesky factor, S_i, of the state covariance matrix;

if `call_type` = `'S'` or `'s'`, the lower triangular Cholesky factor, S_i^*, of the transformed state covariance matrix.

Output:

If `call_type` = `'N'` or `'n'`, the lower triangular Cholesky factor, S_{i+1}, of the state covariance matrix;

if `call_type` = `'F'`, `'f'`, `'S'` or `'s'`, the lower triangular Cholesky factor, S_{i+1}^*, of the transformed state covariance matrix.

Constraints: the diagonal elements of `st` must be non-negative.

at(n, n) — real(kind=*wp*), intent(inout)

Input:

If `call_type` = `'N'`, `'n'`, `'F'` or `'f'`, the state transition matrix, A;

if `call_type` = `'S'` or `'s'`, the transformed matrix, U^*AU^{*T}.

Output:

If `call_type` = `'F'` or `'f'`, the transformed matrix, U^*AU^{*T};

if `call_type` = `'N'`, `'n'`, `'S'` or `'s'`, unchanged on exit.

bt(n, l) — real(kind=*wp*), intent(inout)

Input:

If `call_type` = `'N'`, `'n'`, `'F'` or `'f'`, the noise coefficient matrix, B;
if `call_type` = `'S'` or `'s'`, the transformed matrix, U^*B.

Output:

If `call_type` = `'F'` or `'f'`, the transformed matrix, U^*B;
if `call_type` = `'N'`, `'n'`, `'S'` or `'s'`, unchanged on exit.

ct(m, n) — real(kind=*wp*), intent(inout)

Input:

If `call_type` = `'N'`, `'n'`, `'F'` or `'f'`, the measurement coefficient matrix, C;
if `call_type` = `'S'` or `'s'`, the transformed matrix, CU^{*T}.

Output:

If `call_type` = `'F'` or `'f'`, the transformed matrix, CU^{*T};
if `call_type` = `'N'`, `'n'`, `'S'` or `'s'`, unchanged on exit.

r(m, m) — real(kind=*wp*), intent(in)

Input: the lower triangular Cholesky factor of the measurement noise covariance matrix, R_i.

Constraints: the diagonal elements of `r` must be non-negative.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

q(l, l) — real(kind=*wp*), intent(in), optional

Input: the lower triangular Cholesky factor of the state noise covariance matrix, Q_i.

Note: if `q` is not present then it is assumed to be the identity matrix.

Constraints: the diagonal elements of `q` must be non-negative.

h(m, m) — real(kind=*wp*), intent(out), optional

Output: the lower triangular Cholesky factor of the covariance matrix for the computed residuals, H_i.

kt(n, m) — real(kind=*wp*), intent(out), optional

Output: the Kalman gain matrix, K_i, premultiplied by the state transformed transition matrix, U^*AK_i.

u(n, n) — real(kind=*wp*), intent(out), optional

Output: the transformation matrix, U^*.

Constraints: if `call_type` = `'N'` or `'n'` then `u` must not be present.

u_s(n, n) — real(kind=*wp*), intent(in), optional

Input: the user-specified transformation matrix, U^*.

Constraints: if `call_type` = `'N'` or `'n'` then `u_s` must not be present, and if `call_type` = `'S'` or `'s'` then `u_s` must be present.

p(n, n) — real(kind=*wp*), intent(out), optional

Output: the covariance matrix, $P_{i+1|i}$, associated with the computed state vector, $\hat{X}_{i+1|i}$.

tol — real(kind=*wp*), intent(in), optional

Input: the tolerance used to test for singularity of the matrix, H_i. The inverse of the condition number of the lower triangular matrix `h` is estimated. If this estimate is less than `tol`, then H_i is assumed to be singular.

Note: if $0.0 \leq$ `tol` $\leq m^2 \times$ `EPSILON(1.0_`*wp*`)` then the default value is used.

Default: `tol` $= m^2 \times$ `EPSILON(1.0_`*wp*`)`.

call_type — character(len=1), intent(in), optional

Input: specifies how to transform the input matrices.

If `call_type` = `'F'` or `'f'` then this indicates the 'first call' with a given matrix pair (A, C). When the argument `u_s` is not present the transformation matrix U^* is computed internally, otherwise U^* is set to `u_s`.

If `call_type` = `'S'` or `'s'` then this indicates a 'subsequent call' with the matrix pair (A, C). It is assumed that the input matrices are in Hessenberg form and the transformation matrix U^* is supplied in the optional argument `u_s`.

If `call_type` = `'N'` or `'n'` then the original system is already in Hessenberg form ($U^* = I$) and the optional argument `u_s` must not be present.

Note: when using `call_type` = `'N'` or `'n'` it must be used for both the first call and also subsequent calls.

Default: `call_type` = `'N'`.

xt(n) — real(kind=*wp*), intent(inout), optional

Input:

If `call_type` = `'N'`, `'n'`, `'F'` or `'f'`, the estimate of the state vector, $\hat{X}_{i|i-1}$;

if `call_type` = `'S'` or `'s'`, the estimated transformed state vector, $U^*\hat{X}_{i|i-1}$.

Output:

If `call_type` = `'N'` or `'n'`, the estimate of the state vector, $\hat{X}_{i+1|i}$;

if `call_type` = `'F'`, `'f'`, `'S'` or `'s'`, the estimated transformed state vector, $U^*\hat{X}_{i+1|i}$.

Constraints: if `xt` is present then the argument `y` must also be present.

x(n) — real(kind=*wp*), intent(out), optional

Output: the estimated untransformed state vector, $\hat{X}_{i+1|i}$; this is obtained by premultiplying the vector `xt` by U^{*T}.

Constraints: if `x` is present then the argument `xt` must also be present.

Note: if `call_type` = `'N'` or `'n'` then `x` is set to `xt`.

y(m) — real(kind=*wp*), intent(in), optional

Input: the observation vector, Y_i.

Constraints: if `y` is present then argument `xt` must also be present.

resid(m) — real(kind=*wp*), intent(out), optional

Output: the calculated residuals r_i, $r_i = Y_i - C\hat{X}_{i|i-1}$.

Note: the residual is independent of the reference frame.

Constraints: if `resid` is present then the arguments `xt` and `y` must also be present.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
304	Invalid presence of an optional argument.
305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

Warnings (error%level = 1):

error%code	Description
201	The Cholesky factor of matrix H_i is singular. The singularity of the Cholesky factor of matrix H_i means that the procedure is not able to return values for either the Kalman gain matrix K_i or the predicted state vector $\hat{X}_{i+1\|i}$.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

6 Further Comments

6.1 Mathematical Background

For models with time-varying A, B and C where (A, C) is already in Hessenberg form, this procedure should be used with `call_type` = `'N'` or `'n'`. For more general time-varying models `nag_kalman_sqrt_cov_var` can be used.

The Cholesky factors of the covariance matrices can be computed using `nag_sym_lin_fac`.

Note that the model

$$\begin{array}{lcll} X_{i+1} & = & AX_i + W_i, & \mathrm{var}(W_i) = Q_i \\ Y_i & = & CX_i + V_i, & \mathrm{var}(V_i) = R_i \end{array}$$

can be specified either with the argument $\mathtt{q} = Q^{1/2}$ and B set to the identity matrix or with $B = Q^{1/2}$ and the argument `q` not present.

If W_i and V_i are independent multivariate Normal variates then the log-likelihood for observations $i = 1, 2, \ldots, t$ is given by

$$L(\theta) = \kappa - \frac{1}{2}\sum_{i=1}^{t} \ln(\det(H_i)) - \frac{1}{2}\sum_{i=1}^{t} r_i^T H_i^{-1} r_i,$$

where κ is a constant and H_i is the covariance matrix for the residuals, $r_i = Y_i - C\hat{X}_{i|i-1}$.
(Note: Since $Y_i - C\hat{X}_{i|i-1} = Y_i - (CU^{*T})\hat{X}^*_{i|i-1}$, the residuals r_i are the same for both the transformed and original model.)

The initial estimate of the transformed state vector can be computed from the estimate of the original state vector $\hat{X}_{1|0}$, say, by premultiplying it by the matrix U^* as returned by calling this procedure with `call_type = 'F'`; that is, $\hat{X}^*_{1|0} = U^*\hat{X}_{1|0}$.

The estimate of the transformed state vector $\hat{X}^*_{i+1|i}$ can be computed from the previous value $\hat{X}^*_{i|i-1}$ by

$$\hat{X}^*_{i+1|i} = (U^*AU^{*T})\hat{X}^*_{i|i-1} + (U^*AK_i)r_i.$$

6.2 Algorithmic Detail

The procedure performs one recursion of the square root covariance filter algorithm, summarized as follows:

$$\begin{pmatrix} R_i^{1/2} & 0 & CS_i \\ & & \\ 0 & BQ_i^{1/2} & AS_i \end{pmatrix} U = \begin{pmatrix} H_i^{1/2} & 0 & 0 \\ & & \\ G_i & S_{i+1} & 0 \end{pmatrix},$$

where U is an orthogonal transformation triangularizing the left-hand pre-array to produce the right-hand post-array and S_i is the lower triangular Cholesky factor of the state covariance matrix $P_{i+1|i}$, $Q_i^{1/2}$ and $R_i^{1/2}$ are the lower triangular Cholesky factors of the covariance matrices Q and R and $H^{1/2}$ is the lower triangular Cholesky factor of the covariance matrix of the residuals (see Vanbegin *et al.* [2]). The triangularization is carried out via Householder transformations exploiting the zero pattern of the pre-array. The relationship between the Kalman gain matrix K_i and G_i is given by

$$AK_i = G_i\left(H_i^{1/2}\right)^{-1}.$$

In order to exploit the invariant parts of the model to simplify the computation of U the results for the transformed state space U^*X are computed, where U^* is the transformation that reduces the matrix pair (A, C) to lower observer Hessenberg form. That is, the matrix U^* is computed such that the compound matrix

$$\begin{bmatrix} CU^{*T} \\ U^*AU^{*T} \end{bmatrix}$$

is a lower trapezoidal matrix. Further, the transformed matrix U^*B is used in place of the untransformed matrix B. These transformations need only be computed once at the start of a series, and the procedure will, optionally, compute them. This procedure returns the product of the matrices U^*AU^{*T} and U^*K_i, U^*AK_i, the Cholesky factor of the updated transformed state covariance matrix S^*_{i+1} (where $U^*P_{i+1|i}U^{*T} = S^*_{i+1}S^{*T}_{i+1}$) and the matrix $H_i^{1/2}$, valid for both transformed and original models, which is used in the computation of the likelihood for the model. Note that the covariance matrices Q_i and R_i can be time-varying.

The algorithm requires $\frac{1}{6}n^3 + n^2\left(\frac{3}{2}m + l\right) + 2nm^2 + \frac{2}{3}m^3$ operations and is backward stable (see Verhaegen and Van Dooren [3]). The transformation to lower observer Hessenberg form requires $O((n+m)n^2)$ operations.

6.3 Accuracy

The use of the square root algorithm improves the stability of the computations as compared with the direct coding of the Kalman filter. The accuracy will depend on the model.

Example 1: Time-varying square root covariance filter example

The example program first inputs the number of updates to be computed and the problem sizes. The initial state vector and state covariance matrix are input followed by the model matrices A_i, B_i, C_i, R_i and optionally Q_i. The Cholesky factors of the covariance matrices can be computed if required. The model matrices can be input at each update or only once at the first step. At each update the observed values are input and the residuals are computed and printed and the estimate of the state vector, $\hat{X}_{i|i-1}$, and the deviance are updated. The deviance is $-2 \times$ log-likelihood ignoring the constant. After the final update the state covariance matrix is computed from S and printed along with final estimate of the state vector and the value of the deviance.

The data is for a two-dimensional time series to which a VARMA(1,1) has been fitted. For the specification of a VARMA model as a state space model see the Chapter Introduction. The estimate for $S_{1|0}$ is obtained by using `nag_kalman_predict` to obtain the steady-state solution of the prediction equation.

The mean of each series is input before the first update and subtracted from the observations before the measurement update is computed.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_tsa_kalman_ex01

  ! Example Program Text nag_tsa_kalman
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_tsa_kalman, ONLY : nag_kalman_sqrt_cov_var, nag_kalman_predict
  USE nag_tri_lin_sys, ONLY : nag_tri_lin_sol
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_sym_lin_sys, ONLY : nag_key_pos, nag_sym_lin_fac
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC ABS, DOT_PRODUCT, EPSILON, KIND, LOG, MAXVAL, SQRT
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  REAL (wp), PARAMETER :: zero = 0.0_wp
  ! .. Local Scalars ..
  INTEGER :: i, j, l, m, n, ncall, step
  REAL (wp) :: dev
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:,:), b(:,:), c(:,:), h(:,:), p(:,:), &
   q(:,:), r(:,:), resid(:), s(:,:), s_last(:,:), x(:), y(:), ymean(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_tsa_kalman_ex01'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) ncall, n, m, l

  ALLOCATE (a(n,n),b(n,l),c(m,n),h(m,m),p(n,n),q(l,l),r(m,m),s(n,n),x(n), &
   y(m),resid(m),ymean(m),s_last(n,n)) ! Allocate storage

  READ (nag_std_in,*) x
  READ (nag_std_in,*) ymean
  READ (nag_std_in,*) (a(i,:),i=1,n)
```

```
    READ (nag_std_in,*) (b(i,:),i=1,n)
    READ (nag_std_in,*) (c(i,:),i=1,m)
    READ (nag_std_in,*) (r(i,:),i=1,m)
    READ (nag_std_in,*) (q(i,:),i=1,l)

    CALL nag_sym_lin_fac(nag_key_pos,'l',q)

    ! Obtain initial estimate for S via steady solution
    ! of the prediction equation.

    s = zero
    j = 0

    DO
      s_last = s
      j = j + 1

      CALL nag_kalman_predict(s,a,b,q=q)

      IF (MAXVAL(ABS(s-s_last))<0.1_wp*SQRT(EPSILON(1.0_wp))) EXIT
      IF (j==50) THEN
        WRITE (nag_std_out,*) 'nag_kalman_predict failed to obtain &
         &initial estimate for s via steady'
        WRITE (nag_std_out,*) &
         'state solution of the prediction equation in 50 iterations'

        DEALLOCATE (a,b,c,h,p,q,r,s,s_last,x,y,resid, &
         ymean)                  ! Deallocate storage

        STOP
      END IF
    END DO

    dev = zero
    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) ' step        Residuals'

    ! Loop through data

    DO step = 1, ncall
      READ (nag_std_in,*) y
      y = y - ymean

      ! Perform time and measurement update

      IF (step==ncall) THEN

        CALL nag_kalman_sqrt_cov_var(s,a,b,c,r,q=q,h=h,resid=resid,y=y,x=x, &
         p=p)

      ELSE

        CALL nag_kalman_sqrt_cov_var(s,a,b,c,r,q=q,h=h,resid=resid,y=y,x=x)

      END IF
      WRITE (nag_std_out,'(i4,4f12.4)') step, resid

      ! Update loglikelihood

      CALL nag_tri_lin_sol('l',h,resid)

      dev = dev + DOT_PRODUCT(resid,resid)
```

```
        DO i = 1, m
          dev = dev + 2.0_wp*LOG(h(i,i))
        END DO
      END DO

      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) ' Final X(i+1|i) '
      WRITE (nag_std_out,'(4f12.4)') x
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) ' Final Value of P'

      DO i = 1, n
        WRITE (nag_std_out,'(4f12.4)') p(i,1:i)
      END DO
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,'(A,e13.4)') ' Deviance = ', dev

      DEALLOCATE (a,b,c,h,p,q,r,s,s_last,x,y,resid,ymean) ! Deallocate storage

    END PROGRAM nag_tsa_kalman_ex01
```

2 Program Data

```
Example Program Data for nag_tsa_kalman_ex01
48 4 2 2                     : ncall, n, m, l

 0.000  0.000  0.000  0.000   : x(1:n)
 4.404  7.991                 : ymean(1:m)
 0.607 -0.033  1.000  0.000
 0.000  0.543  0.000  1.000
 0.000  0.000  0.000  0.000
 0.000  0.000  0.000  0.000   : a(1:n,1:n)
 1.000  0.000
 0.000  1.000
 0.543  0.125
 0.134  0.026                 : b(1:n,1:l)
 1.000  0.000  0.000  0.000
 0.000  1.000  0.000  0.000   : c(1:m,1:n)
 0.000  0.000
 0.000  0.000                 : r(1:m,1:m)
 2.598  0.560
 0.560  5.330                 : q(1:l,1:l)

-1.490  7.340                 : first y(1:m)
-1.620  6.350
 5.200  6.960
 6.230  8.540
 6.210  6.620
 5.860  4.970
 4.090  4.550
 3.180  4.810
 2.620  4.750
 1.490  4.760
 1.170 10.880
 0.850 10.010
-0.350 11.620
 0.240 10.360
 2.440  6.400
 2.580  6.240
 2.040  7.930
 0.400  4.040
 2.260  3.730
```

```
3.340  5.600
5.090  5.350
5.000  6.810
4.780  8.270
4.110  7.680
3.450  6.650
1.650  6.080
1.290 10.250
4.090  9.140
6.320 17.750
7.500 13.300
3.890  9.630
1.580  6.800
5.210  4.080
5.250  5.060
4.930  4.940
7.380  6.650
5.870  7.940
5.810 10.760
9.680 11.890
9.070  5.850
7.290  9.010
7.840  7.500
7.550 10.020
7.320 10.380
7.970  8.150
7.760  8.370
7.000 10.730
8.350 12.140                 : last (ncall) y(1:m)
```

3 Program Results

```
Example Program Results for nag_tsa_kalman_ex01

 step        Residuals
   1     -5.8940     -0.6510
   2     -1.4710     -1.0407
   3      5.1658      0.0447
   4     -1.3280      0.4580
   5      1.3652     -1.5066
   6     -0.2337     -2.4192
   7     -0.8685     -1.7065
   8     -0.4624     -1.1519
   9     -0.7510     -1.4218
  10     -1.3526     -1.3335
  11     -0.6707      4.8593
  12     -1.7389      0.4138
  13     -1.6376      2.7549
  14     -0.6137      0.5463
  15      0.9067     -2.8093
  16     -0.8255     -0.9355
  17     -0.7494      1.0247
  18     -2.2922     -3.8441
  19      1.8812     -1.7085
  20     -0.7112     -0.2849
  21      1.6747     -1.2400
  22     -0.6619      0.0609
  23      0.3271      1.0074
  24     -0.8165     -0.5325
  25     -0.2759     -1.0489
  26     -1.9383     -1.1186
  27     -0.3131      3.5855
 28      1.3726     -0.1289
 29      1.4153      8.9545
 30      0.3672     -0.4126
 31     -2.3659     -1.2823
 32     -1.0130     -1.7306
 33      3.2472     -3.0836
 34     -1.1501     -1.1623
 35      0.6855     -1.2751
 36      2.3432      0.2570
 37     -1.6892      0.3565
 38      1.3871      3.0138
 39      3.3840      2.1312
 40     -0.5118     -4.7670
 41      0.8569      2.3741
 42      0.9558     -1.2209
 43      0.6778      2.1993
 44      0.4304      1.1393
 45      1.4987     -1.2255
 46      0.5361      0.1237
 47      0.2649      2.4582
 48      2.0095      2.5623

 Final X(i+1|i)
     3.6698      2.5888      0.0000      0.0000

 Final Value of P
     2.5980
     0.5600      5.3300
     1.4807      0.9703      0.9253
     0.3627      0.2136      0.2236      0.0542

Deviance =    0.2229E+03
```

Example 2: Time-invariant square root covariance filter example

The example program first inputs the number of updates to be computed and the problem sizes. The initial state vector and the Cholesky factor of the state covariance matrix are input followed by the model matrices $A, B, C, R^{1/2}$ and optionally $Q^{1/2}$ (the Cholesky factors of the covariance matrices being input). Since the matrix pair (A, C) is already in condensed form $U^* = I$ and the default value of `call_type` can be used. At each update the observed values are input and the residuals are computed and printed and the estimate of the state vector, $\hat{X}_{i|i-1}$, and the deviance are updated. The deviance is $-2\times$log-likelihood ignoring the constant. After the final update the estimate of the state vector is computed and the state covariance matrix is computed from S and these are printed along with the value of the deviance.

The data is for a two-dimensional time series to which a VARMA(1,1) has been fitted. For the specification of a VARMA model as a state space model see the Chapter Introduction. The means of the two series are included as additional states that do not change over time. The estimate for $S_{1|0}$ is obtained by using `nag_kalman_init`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_tsa_kalman_ex02

  ! Example Program Text nag_tsa_kalman
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_tsa_kalman, ONLY : nag_kalman_sqrt_cov_invar, nag_kalman_init
  USE nag_tri_lin_sys, ONLY : nag_tri_lin_sol
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC DOT_PRODUCT, KIND, LOG
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  REAL (wp), PARAMETER :: zero = 0.0_wp
  ! .. Local Scalars ..
  INTEGER :: i, l, m, n, ncall, step
  REAL (wp) :: dev
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:,:), b(:,:), c(:,:), h(:,:), p(:,:), &
   q(:,:), r(:,:), resid(:), s(:,:), x(:), xt(:), y(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_tsa_kalman_ex02'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) ncall, n, m, l

  ALLOCATE (a(n,n),b(n,l),c(m,n),h(m,m),p(n,n),q(l,l),r(m,m),s(n,n),x(n), &
   xt(n),y(m),resid(m))        ! Allocate storage

  READ (nag_std_in,*) xt
  READ (nag_std_in,*) (a(i,:),i=1,n)
  READ (nag_std_in,*) (b(i,:),i=1,n)
  READ (nag_std_in,*) (c(i,:),i=1,m)
  READ (nag_std_in,*) (r(i,:),i=1,m)
  READ (nag_std_in,*) (q(i,:),i=1,l)
```

```
      s = zero

      CALL nag_kalman_init(a(1:n-m,1:n-m),b(1:n-m,1:l),s(1:n-m,1:n-m),q=q)

      dev = zero
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) ' step        Residuals'

      ! Loop through data

      DO step = 1, ncall
        READ (nag_std_in,*) y

        ! Perform time and measurement update

        IF (step==1) THEN

          CALL nag_kalman_sqrt_cov_invar(s,a,b,c,r,q=q,h=h,resid=resid,y=y, &
           xt=xt)

        ELSE

          CALL nag_kalman_sqrt_cov_invar(s,a,b,c,r,q=q,h=h,resid=resid,y=y, &
           x=x,xt=xt,p=p)

        END IF
        WRITE (nag_std_out,'(i4,4f10.4)') step, resid

        ! Update loglikelihood

        CALL nag_tri_lin_sol('l',h,resid)

        dev = dev + DOT_PRODUCT(resid,resid)
        DO i = 1, m
          dev = dev + 2.0_wp*LOG(h(i,i))
        END DO
      END DO

      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) ' Final X(i+1|i) '
      WRITE (nag_std_out,'(10f10.4)') x
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) ' Final Value of P'

      DO i = 1, n
        WRITE (nag_std_out,'(10f10.4)') p(i,1:i)
      END DO
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,'(A,e10.4)') ' Deviance = ', dev

      DEALLOCATE (a,b,c,h,p,q,r,s,x,xt,y,resid) ! Deallocate storage

    END PROGRAM nag_tsa_kalman_ex02
```

2 Program Data

```
Example Program Data for nag_tsa_kalman_ex02
48 6 2 2                                 : ncall, n, m, l

 0.000  0.000  0.000  0.000  4.404  7.991 : xt(1:n)
 0.607 -0.033  1.000  0.000  0.000  0.000
 0.000  0.543  0.000  1.000  0.000  0.000
 0.000  0.000  0.000  0.000  0.000  0.000
 0.000  0.000  0.000  0.000  0.000  0.000
 0.000  0.000  0.000  0.000  1.000  0.000
 0.000  0.000  0.000  0.000  0.000  1.000 : a(1:n,1:n)
 1.000  0.000
 0.000  1.000
 0.543  0.125
 0.134  0.026
 0.000  0.000
 0.000  0.000                             : b(1:n,1:l)
 1.000  0.000  0.000  0.000  1.000  0.000
 0.000  1.000  0.000  0.000  0.000  1.000 : c(1:m,1:n)
 0.000  0.000
 0.000  0.000                             : r(1:m,1:m)
 2.598  0.560
 0.560  5.330                             : q(1:l,1:l)

-1.490  7.340                             : first y(1:m)
-1.620  6.350
 5.200  6.960
 6.230  8.540
 6.210  6.620
 5.860  4.970
 4.090  4.550
 3.180  4.810
 2.620  4.750
 1.490  4.760
 1.170 10.880
 0.850 10.010
-0.350 11.620
 0.240 10.360
 2.440  6.400
 2.580  6.240
 2.040  7.930
 0.400  4.040
 2.260  3.730
 3.340  5.600
 5.090  5.350
 5.000  6.810
 4.780  8.270
 4.110  7.680
 3.450  6.650
 1.650  6.080
 1.290 10.250
 4.090  9.140
 6.320 17.750
 7.500 13.300
 3.890  9.630
 1.580  6.800
 5.210  4.080
 5.250  5.060
 4.930  4.940
 7.380  6.650
 5.870  7.940
 5.810 10.760
 9.680 11.890
 9.070  5.850
 7.290  9.010
 7.840  7.500
 7.550 10.020
 7.320 10.380
 7.970  8.150
```

```
7.760  8.370
7.000 10.730
8.350 12.140                            : last (ncall) y(1:m)
```

3 Program Results

```
Example Program Results for nag_tsa_kalman_ex02

 step        Residuals
  1   -5.8940   -0.6510
  2   -1.4710   -1.0407
  3    5.1658    0.0447
  4   -1.3280    0.4580
  5    1.3652   -1.5066
  6   -0.2337   -2.4192
  7   -0.8685   -1.7065
  8   -0.4624   -1.1519
  9   -0.7510   -1.4218
 10   -1.3526   -1.3335
 11   -0.6707    4.8593
 12   -1.7389    0.4138
 13   -1.6376    2.7549
 14   -0.6137    0.5463
 15    0.9067   -2.8093
 16   -0.8255   -0.9355
 17   -0.7494    1.0247
 18   -2.2922   -3.8441
 19    1.8812   -1.7085
 20   -0.7112   -0.2849
 21    1.6747   -1.2400
 22   -0.6619    0.0609
 23    0.3271    1.0074
 24   -0.8165   -0.5325
 25   -0.2759   -1.0489
 26   -1.9383   -1.1186
 27   -0.3131    3.5855
 28    1.3726   -0.1289
 29    1.4153    8.9545
 30    0.3672   -0.4126
 31   -2.3659   -1.2823
 32   -1.0130   -1.7306
 33    3.2472   -3.0836
 34   -1.1501   -1.1623
 35    0.6855   -1.2751
 36    2.3432    0.2570
 37   -1.6892    0.3565
 38    1.3871    3.0138
 39    3.3840    2.1312
 40   -0.5118   -4.7670
 41    0.8569    2.3741
 42    0.9558   -1.2209
 43    0.6778    2.1993
 44    0.4304    1.1393
 45    1.4987   -1.2255
 46    0.5361    0.1237
 47    0.2649    2.4582
 48    2.0095    2.5623

 Final X(i+1|i)
   3.6698    2.5888    0.0000    0.0000    4.4040    7.9910

 Final Value of P
```

```
    2.5980
    0.5600    5.3300
    1.4807    0.9703    0.9253
    0.3627    0.2136    0.2236    0.0542
    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000

 Deviance = 0.2229E+03
```

References

[1] Anderson B D O and Moore J B (1979) *Optimal Filtering* Prentice Hall, Englewood Cliffs, New Jersey

[2] Vanbegin M, Van Dooren P and Verhaegen M H G (1989) Algorithm 675: FORTRAN subroutines for computing the square root covariance filter and square root information filter in dense or Hessenberg forms. *ACM Trans. Math. Software* **15** 243–256.

[3] Verhaegen M H G and Van Dooren P (1986) Numerical aspects of different Kalman filter implementations *IEEE Trans. Auto. Contr.* **AC-31** 907–917

[4] Wei W W S (1990) *Time Series Analysis: Univariate and Multivariate Methods* Addison-Wesley