nag_kalman_unscented_state_revcom (g13ejc)

1 Purpose

nag_kalman_unscented_state_revcom (g13ejc) applies the Unscented Kalman Filter to a nonlinear state space model, with additive noise.

nag_kalman_unscented_state_revcom (g13ejc) uses reverse communication for evaluating the nonlinear functionals of the state space model.

2 Specification

```c
#include <nag.h>
#include <nagg13.h>

void nag_kalman_unscented_state_revcom (Integer *irevcm, Integer mx,
    Integer my, const double y[], const double lx[], Integer pdlx,
    const double ly[], Integer pdly, double x[], double st[], Integer pdst,
    Integer *n, double xt[], Integer pdxt, double fxt[], Integer pdfxt,
    const double ropt[], Integer lropt, Integer icomm[], Integer licomm,
    double rcomm[], Integer lrcomm, NagError *fail)
```

3 Description

nag_kalman_unscented_state_revcom (g13ejc) applies the Unscented Kalman Filter (UKF), as described in Julier and Uhlmann (1997b) to a nonlinear state space model, with additive noise, which, at time $t$, can be described by:

$$
x_{t+1} = F(x_t) + v_t \\
y_t = H(x_t) + u_t
$$

where $x_t$ represents the unobserved state vector of length $m_x$, and $y_t$ the observed measurement vector of length $m_y$. The process noise is denoted $v_t$, which is assumed to have mean zero and covariance structure $\Sigma_x$, and the measurement noise by $u_t$, which is assumed to have mean zero and covariance structure $\Sigma_y$.

3.1 Unscented Kalman Filter Algorithm

Given $\hat{x}_0$, an initial estimate of the state and $P_0$ and initial estimate of the state covariance matrix, the UKF can be described as follows:

(a) Generate a set of sigma points (see section Section 3.2):

$$\mathcal{X}_t = \begin{bmatrix} \hat{x}_{t-1} & \hat{x}_{t-1} + \gamma \sqrt{P_{t-1}} & \hat{x}_{t-1} - \gamma \sqrt{P_{t-1}} \end{bmatrix}$$

(b) Evaluate the known model function $F$:

$$\mathcal{F}_i = F(\mathcal{X}_i)$$

The function $F$ is assumed to accept the $m_x \times n$ matrix, $\mathcal{X}_i$ and return an $m_x \times n$ matrix, $\mathcal{F}_i$. The columns of both $\mathcal{X}_i$ and $\mathcal{F}_i$ correspond to different possible states. The notation $\mathcal{F}_{i,j}$ is used to denote the $i$th column of $\mathcal{F}_i$, hence the result of applying $F$ to the $i$th possible state.
(c) Time Update:
\[ \dot{x}_t = \sum_{i=1}^{n} W_i^m \mathcal{F}_{t,i} \]  
\[ P_t = \sum_{i=1}^{n} W_i^c (\mathcal{F}_{t,i} - \dot{x}_t)(\mathcal{F}_{t,i} - \dot{x}_t)^T + \Sigma_x \]  
(3)  
(4)

(d) Redraw another set of sigma points (see section Section 3.2):
\[ y_t = [\hat{x}_t \ \hat{x}_t + \gamma \sqrt{P_t} \ \hat{x}_t - \gamma \sqrt{P_t}] \]  
(5)

(e) Evaluate the known model function \( H \):
\[ \mathcal{H}_t = H(y_t) \]  
(6)

The function \( H \) is assumed to accept the \( m_x \times n \) matrix, \( y_t \) and return an \( m_y \times n \) matrix, \( \mathcal{H}_t \). The columns of both \( y_t \) and \( \mathcal{H}_t \) correspond to different possible states. As above \( \mathcal{H}_{t,i} \) is used to denote the \( i \)th column of \( \mathcal{H}_t \).

(f) Measurement Update:
\[ \hat{y}_t = \sum_{i=1}^{n} W_i^m \mathcal{H}_{t,i} \]  
\[ P_{yy} = \sum_{i=1}^{n} W_i^c (\mathcal{H}_{t,i} - \hat{y}_t)(\mathcal{H}_{t,i} - \hat{y}_t)^T + \Sigma_y \]  
\[ P_{yx} = \sum_{i=1}^{n} W_i^c (\mathcal{F}_{t,i} - \hat{x}_t)(\mathcal{H}_{t,i} - \hat{y}_t)^T \]  
\[ K_t = P_{yx} P_{yy}^{-1} \]  
\[ \hat{x}_t = \hat{x}_t + K_t (y_t - \hat{y}_t) \]  
\[ P_t = P_t - K_t P_{yy} K_t^T \]  
(7)  
(8)  
(9)  
(10)  
(11)  
(12)

Here \( K_t \) is the Kalman gain matrix, \( \hat{x}_t \) is the estimated state vector at time \( t \) and \( P_t \) the corresponding covariance matrix. Rather than implementing the standard UKF as stated above nag_kalman_unscented_state_revcom (g13ejc) uses the square-root form described in the Haykin (2001).

### 3.2 Sigma Points

A nonlinear state space model involves propagating a vector of random variables through a nonlinear system and we are interested in what happens to the mean and covariance matrix of those variables. Rather than trying to directly propagate the mean and covariance matrix, the UKF uses a set of carefully chosen sample points, referred to as sigma points, and propagates these through the system of interest. An estimate of the propagated mean and covariance matrix is then obtained via the weighted sample mean and covariance matrix.

For a vector of \( m \) random variables, \( x \), with mean \( \mu \) and covariance matrix \( \Sigma \), the sigma points are usually constructed as:
\[ \chi_t = \begin{bmatrix} \mu & \mu + \gamma \sqrt{\Sigma} & \mu - \gamma \sqrt{\Sigma} \end{bmatrix} \]

When calculating the weighted sample mean and covariance matrix two sets of weights are required, one used when calculating the weighted sample mean, denoted \( W^m \) and one used when calculated the weighted sample covariance matrix, denoted \( W^c \). The weights and multiplier, \( \gamma \), are constructed as follows:
\[
\lambda = \alpha^2 (L + \kappa) - L \\
\gamma = \sqrt{L + \lambda} \\
W^n_i = \begin{cases} 
\frac{\lambda}{2(L+\lambda)} & i = 1 \\
\frac{1}{2(L+\lambda)} & i = 2, 3, \ldots, 2L + 1 
\end{cases} \\
W^c_i = \begin{cases} 
\frac{\lambda}{2(L+\lambda)} + 1 - \alpha^2 + \beta & i = 1 \\
\frac{1}{2(L+\lambda)} & i = 2, 3, \ldots, 2L + 1 
\end{cases}
\]

where, usually \( L = m \) and \( \alpha, \beta \) and \( \kappa \) are constants. The total number of sigma points, \( n \), is given by \( 2L + 1 \). The constant \( \alpha \) is usually set to somewhere in the range \( 10^{-4} \leq \alpha \leq 1 \) and for a Gaussian distribution, the optimal values of \( \kappa \) and \( \beta \) are \( 3 - L \) and 2 respectively.

Rather than redrawing another set of sigma points in (d) of the UKF an alternative method can be used where the sigma points used in (a) are augmented to take into account the process noise. This involves replacing equation (5) with:

\[
Y_i = \left[ X_{t_i}^t, \mathcal{X}_{t+1}^t + \gamma \sqrt{\Sigma_x}, \mathcal{Y}_{t+1} - \gamma \sqrt{\Sigma_x} \right] 
\]

Augmenting the sigma points in this manner requires setting \( L \) to \( 2L \) (and hence \( n \) to \( 2n - 1 \)) and recalculating the weights. These new values are then used for the rest of the algorithm. The advantage of augmenting the sigma points is that it keeps any odd-moments information captured by the original propagated sigma points, at the cost of using a larger number of points.

4 References

Haykin S (2001) *Kalman Filtering and Neural Networks* John Wiley and Sons


5 Arguments

**Note:** this function uses reverse communication. Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the argument \texttt{irevcm}. Between intermediate exits and re-entries, all arguments other than \texttt{fxt} must remain unchanged.

1: \texttt{irevcm} – Integer *

\textit{Input/Output}

\textit{On initial entry:} must be set to 0 or 3.

If \texttt{irevcm} = 0, it is assumed that \( t = 0 \), otherwise it is assumed that \( t \neq 0 \) and that \texttt{nag_kalman_unscented_state_revcom (g13ejc)} has been called at least once before at an earlier time step.

\textit{On intermediate exit:} \texttt{irevcm} = 1 or 2. The value of \texttt{irevcm} specifies what intermediate values are returned by this function and what values the calling program must assign to arguments of \texttt{nag_kalman_unscented_state_revcom (g13ejc)} before re-entering the routine. Details of the output and required input are given in the individual argument descriptions.

\textit{On intermediate re-entry:} \texttt{irevcm} must remain unchanged.

\textit{On final exit:} \texttt{irevcm} = 3

\textit{Constraint:} \texttt{irevcm} = 0, 1, 2 or 3.
2: \( \text{mx} \) – Integer

\textit{Input}

\textit{On entry:} \( m_x \), the number of state variables.

\textit{Constraint:} \( \text{mx} \geq 1 \).

3: \( \text{my} \) – Integer

\textit{Input}

\textit{On entry:} \( m_y \), the number of observed variables.

\textit{Constraint:} \( \text{my} \geq 1 \).

4: \( y[\text{my}] \) – const double

\textit{Input}

\textit{On entry:} \( y_t \), the observed data at the current time point.

5: \( \text{lx} \[\text{dim}] \) – const double

\textit{Input}

\textit{Note:} the dimension, \( \text{dim} \), of the array \( \text{lx} \) must be at least \( \text{pdlx} \times \text{mx} \).

The \((i, j)\)th element of the matrix is stored in \( \text{lx}[(j - 1) \times \text{pdlx} + i - 1] \).

\textit{On entry:} \( L_x \), such that \( L_x L_x^T = \Sigma_x \), i.e., the lower triangular part of a Cholesky decomposition of the process noise covariance structure. Only the lower triangular part of the matrix stored in \( \text{lx} \) is referenced.

If \( \text{pdlx} = 0 \), there is no process noise \( (v_t = 0 \text{ for all } t) \) and \( \text{lx} \) is not referenced and may be NULL.

If \( \Sigma_x \) is time dependent, then the value supplied should be for time \( t \).

6: \( \text{pdlx} \) – Integer

\textit{Input}

\textit{On entry:} the stride separating matrix row elements in the array \( \text{lx} \).

\textit{Constraint:} \( \text{pdlx} = 0 \) or \( \text{pdlx} \geq \text{mx} \).

7: \( \text{ly} \[\text{dim}] \) – const double

\textit{Input}

\textit{Note:} the dimension, \( \text{dim} \), of the array \( \text{ly} \) must be at least \( \text{pdly} \times \text{my} \).

The \((i, j)\)th element of the matrix is stored in \( \text{ly}[(j - 1) \times \text{pdly} + i - 1] \).

\textit{On entry:} \( L_y \), such that \( L_y L_y^T = \Sigma_y \), i.e., the lower triangular part of a Cholesky decomposition of the observation noise covariance structure. Only the lower triangular part of the matrix stored in \( \text{ly} \) is referenced.

If \( \Sigma_y \) is time dependent, then the value supplied should be for time \( t \).

8: \( \text{pdly} \) – Integer

\textit{Input}

\textit{On entry:} the stride separating matrix row elements in the array \( \text{ly} \).

\textit{Constraint:} \( \text{pdly} \geq \text{my} \).

9: \( x[\text{mx}] \) – double

\textit{Input/Output}

\textit{On initial entry:} \( \hat{x}_{t-1} \) the state vector for the previous time point.

\textit{On intermediate exit:} when

\( \text{irevcm} = 1 \) 

\( x \) is unchanged.

\( \text{irevcm} = 2 \) 

\( \hat{x}_t \).

\textit{On intermediate re-entry:} \( x \) must remain unchanged.

\textit{On final exit:} \( \hat{x}_t \) the updated state vector.
10: st[dim] – double

**Note:** the dimension, dim, of the array st must be at least pdst × mx.

The (i, j)th element of the matrix is stored in st[(j - 1) × pdst + i - 1].

On initial entry: $S_t$, such that $S_{t-1}^{T} = P_{t-1}$, i.e., the lower triangular part of a Cholesky decomposition of the state covariance matrix at the previous time point. Only the lower triangular part of the matrix stored in st is referenced.

On intermediate exit: when

irevcm = 1
  st is unchanged.

irevcm = 2
  $S_t$, the lower triangular part of a Cholesky factorization of $P_t$.

On intermediate re-entry: st must remain unchanged.

On final exit: $S_t$, the lower triangular part of a Cholesky factorization of the updated state covariance matrix.

11: pdst – Integer

**Input**

On entry: the stride separating matrix row elements in the array st.

**Constraint:** pdst ≥ mx.

12: n – Integer *

**Input/Output**

On initial entry: the value used in the sizing of the fxt and xt arrays. The value of n supplied must be at least as big as the maximum number of sigma points that the algorithm will use. nag_kalman_unscented_state_revc (g13ejc) allows sigma points to be calculated in two ways during the measurement update; they can be redrawn or augmented. Which is used is controlled by ropt.

If redrawn sigma points are used, then the maximum number of sigma points will be $2m_x + 1$, otherwise the maximum number of sigma points will be $4m_x + 1$.

On intermediate exit: the number of sigma points actually being used.

On intermediate re-entry: n must remain unchanged.

On final exit: reset to its value on initial entry.

**Constraints:** if irevcm = 0 or 3,
  - if redrawn sigma points are used, $n \geq 2 \times m_x + 1$;
  - otherwise $n \geq 4 \times m_x + 1$.

13: xt[dim] – double

**Input/Output**

**Note:** the dimension, dim, of the array xt must be at least pdxt × max(my, n).

On initial entry: need not be set.

On intermediate exit: $X_t$, when irevcm = 1, otherwise $Y_t$.

For the $j$th sigma point, the value for the $i$th parameter is held in xt[(j - 1) × pdxt + i - 1], for $i = 1, 2, \ldots, m_x$ and $j = 1, 2, \ldots, n$.

On intermediate re-entry: xt must remain unchanged.

On final exit: the contents of xt are undefined.

14: pdxt – Integer

**Input**

On entry: the stride separating row elements in the two-dimensional data stored in the array xt.

**Constraint:** pdxt ≥ mx.
15:  fxt[dim] – double

   Input/Output

   Note: the dimension, dim, of the array fxt must be at least pdfxt × (n + max(mx, my)).

   On initial entry: need not be set.

   On intermediate exit: the contents of fxt are undefined.

   On intermediate re-entry: F(X_t) when irevcm = 1, otherwise H(Y_t) for the values of X_t and Y_t
   held in xt.

   For the jth sigma point the value for the ith parameter should be held in fxt[(j - 1) × pdfxt + i - 1], for
   j = 1, 2, ..., n. When irevcm = 1, i = 1, 2, ..., mx and when
   irevcm = 2, i = 1, 2, ..., my.

   On final exit: the contents of fxt are undefined.

16:  pdfxt – Integer

   Input

   On entry: the stride separating row elements in the two-dimensional data stored in the array fxt.

   Constraint: pdfxt \geq max(mx, my).

17:  ropt[lropt] – const double

   Input

   On entry: optional arguments. The default value will be used for ropt[i] if lropt < i. Setting
   lropt = 0 will use the default values for all optional arguments and ropt need not be set and may
   be NULL.

   ropt[0]
   If set to 1 then the second set of sigma points are redrawn, as given by equation (5). If set
   to 2 then the second set of sigma points are generated via augmentation, as given by
   equation (13).

   Default is for the sigma points to be redrawn (i.e., ropt[0] = 1)

   ropt[1]
   \kappa_x, value of \kappa used when constructing the first set of sigma points, X_t.

   Defaults to 3 – mx.

   ropt[2]
   \alpha_x, value of \alpha used when constructing the first set of sigma points, X_t.

   Defaults to 1.

   ropt[3]
   \beta_x, value of \beta used when constructing the first set of sigma points, X_t.

   Defaults to 2.

   ropt[4]
   Value of \kappa used when constructing the second set of sigma points, Y_t.

   Defaults to 3 – 2 × mx when pdlx ≠ 0 and the second set of sigma points are augmented and \kappa_x
   otherwise.

   ropt[5]
   Value of \alpha used when constructing the second set of sigma points, Y_t.

   Defaults to \alpha_x.

   ropt[6]
   Value of \beta used when constructing the second set of sigma points, Y_t.

   Defaults to \beta_x.

   Constraints:
   ropt[0] = 1 or 2;
   ropt[1] > -mx;
\( \text{ropt}[4] > -2 \times \text{mx} \) when pdly \( \neq 0 \) and the second set of sigma points are augmented, otherwise \( \text{ropt}[4] > -\text{mx} \);
\( \text{ropt}[i - 1] > 0 \), for \( i = 3, 6 \).

18: \hspace{1cm} \text{lropt} \text{ – Integer} \hspace{1cm} \text{Input}
   
   \textit{On entry}: length of the options array \text{ropt}.
   
   \textit{Constraint}: \( 0 \leq \text{lropt} \leq 7 \).

19: \hspace{1cm} \text{icomm[licomm]} \text{ – Integer} \hspace{1cm} \text{Communication Array}
   
   \textit{On initial entry}: \text{icomm} need not be set.
   
   \textit{On intermediate exit}: \text{icomm} is used for storage between calls to
   \text{nag_kalman_unscented_state_revcom (g13ejc)}.
   
   \textit{On intermediate re-entry}: \text{icomm} must remain unchanged.
   
   \textit{On final exit}: \text{icomm} is not defined.

20: \hspace{1cm} \text{licomm} \text{ – Integer} \hspace{1cm} \text{Input}
   
   \textit{On entry}: the length of the array \text{icomm}. If \text{licomm} is too small and
   \text{licomm} \geq 2 then \text{fail.code} = \text{NE_TOO_SMALL} is returned and the minimum value for
   \text{licomm} and \text{lrcomm} are given by \text{icomm}[0] and \text{icomm}[1] respectively.
   
   \textit{Constraint}: \text{licomm} \geq 30.

21: \hspace{1cm} \text{rcomm[lrcomm]} \text{ – double} \hspace{1cm} \text{Communication Array}
   
   \textit{On initial entry}: \text{rcomm} need not be set.
   
   \textit{On intermediate exit}: \text{rcomm} is used for storage between calls to
   \text{nag_kalman_unscented_state_revcom (g13ejc)}.
   
   \textit{On intermediate re-entry}: \text{rcomm} must remain unchanged.
   
   \textit{On final exit}: \text{rcomm} is not defined.

22: \hspace{1cm} \text{lrcomm} \text{ – Integer} \hspace{1cm} \text{Input}
   
   \textit{On entry}: the length of the array \text{rcomm}. If \text{lrcomm} is too small and
   \text{licomm} \geq 2 then \text{fail.code} = \text{NW_INT} is returned and the minimum value for
   \text{licomm} and \text{lrcomm} are given by \text{icomm}[0] and \text{icomm}[1] respectively.
   
   \textit{Suggested value}: \text{lrcomm} = 30 + \text{my} + \text{mx} \times \text{my} + (1 + \text{nb}) \times \text{max(mx,my)},
   where \text{nb} is the optimal \text{block size}. In most cases a \text{block size} of 128 will be sufficient.
   
   \textit{Constraint}: \text{lrcomm} \geq 30 + \text{my} + \text{mx} \times \text{my} + 2 \times \text{max(mx,my)}.

23: \hspace{1cm} \text{fail} \text{ – NagError } * \hspace{1cm} \text{Input/Output}
   
   The NAG error argument (see Section 3.6 in the Essential Introduction).

6 \hspace{1cm} \textbf{Error Indicators and Warnings} \hspace{1cm} \text{g13ejc.7}

\textbf{NE_ALLOC_FAIL}

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE_ARRAY_SIZE}

\textit{On entry}, \text{pdxt} = \langle \text{value} \rangle \text{ and } \text{mx} = \langle \text{value} \rangle.

\textit{Constraint}: if \text{irevcm} = 1, \text{pdxt} \geq \text{mx}.
On entry, \(pdfxt = \langle \text{value} \rangle\) and \(my = \langle \text{value} \rangle\).
Constraint: if \(irevcm = 2\), \(pdfxt \geq my\).

On entry, \(pdx = \langle \text{value} \rangle\) and \(mx = \langle \text{value} \rangle\).
Constraint: \(pdx = 0\) or \(pdx \geq mx\).

On entry, \(pdly = \langle \text{value} \rangle\) and \(my = \langle \text{value} \rangle\).
Constraint: \(pdly \geq my\).

On entry, \(pdst = \langle \text{value} \rangle\) and \(mx = \langle \text{value} \rangle\).
Constraint: \(pdst \geq mx\).

On entry, \(pdxt = \langle \text{value} \rangle\) and \(mx = \langle \text{value} \rangle\).
Constraint: \(pdxt \geq mx\).

**NE_BAD_PARAM**

On entry, argument \(\langle \text{value} \rangle\) had an illegal value.

**NE_ILLEGAL_COMM**

\(icomm\) has been corrupted between calls.
\(rcomm\) has been corrupted between calls.

**NE_INT**

On entry, \(lropt = \langle \text{value} \rangle\).
Constraint: \(0 \leq lropt \leq 7\).

On entry, \(irevcm = \langle \text{value} \rangle\).
Constraint: \(irevcm = 0, 1, 2\) or \(3\).

On entry, \(mx = \langle \text{value} \rangle\).
Constraint: \(mx \geq 1\).

On entry, \(my = \langle \text{value} \rangle\).
Constraint: \(my \geq 1\).

On entry, augmented sigma points requested, \(n = \langle \text{value} \rangle\) and \(mx = \langle \text{value} \rangle\).
Constraint: \(n \geq \langle \text{value} \rangle\).

On entry, redrawn sigma points requested, \(n = \langle \text{value} \rangle\) and \(mx = \langle \text{value} \rangle\).
Constraint: \(n \geq \langle \text{value} \rangle\).

**NE_INT_CHANGED**

\(mx\) has changed between calls.

On intermediate entry, \(mx = \langle \text{value} \rangle\).
On initial entry, \(mx = \langle \text{value} \rangle\).

\(my\) has changed between calls.

On intermediate entry, \(my = \langle \text{value} \rangle\).
On initial entry, \(my = \langle \text{value} \rangle\).

\(n\) has changed between calls.

On intermediate entry, \(n = \langle \text{value} \rangle\).
On intermediate exit, \(n = \langle \text{value} \rangle\).

**NE_INTERNAL_ERROR**

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.
NE_INVALID_OPTION
On entry, ropt[0] = <value>.
Constraint: ropt[0] = 1 or 2.
On entry, ropt[(value)] = <value>.
Constraint: \( \alpha > 0 \).
On entry, ropt[(value)] = <value>.
Constraint: \( \kappa > <\text{value}\).'

NE_MAT_NOT_POS_DEF
A weight was negative and it was not possible to downdate the Cholesky factorization.
Unable to calculate the Cholesky factorization of the updated state covariance matrix.
Unable to calculate the Kalman gain matrix.

NE_NO_LICENCE
Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_TOO_SMALL
On entry, licomm = <value>.
Constraint: licomm \geq 2.
icomm is too small to return the required array sizes.

NW_INT
On entry, licomm = <value> and lrcomm = <value>.
Constraint: licomm \geq 30 and lrcomm \geq 30 + \text{my} \times \text{mx} + 2 \times \max(\text{mx}, \text{my})
The minimum required values for licomm and lrcomm are returned in icomm[0] and icomm[1]
respectively.

7 Accuracy
Not applicable.

8 Parallelism and Performance
nag_kalman_unscented_state_revcom (g13ejc) is not threaded by NAG in any implementation.
nag_kalman_unscented_state_revcom (g13ejc) makes calls to BLAS and/or LAPACK routines, which
may be threaded within the vendor library used by this implementation. Consult the documentation for
the vendor library for further information.
Please consult the X06 Chapter Introduction for information on how to control and interrogate the
OpenMP environment used within this function. Please also consult the Users’ Note for your
implementation for any additional implementation-specific information.

9 Further Comments
As well as implementing the Unscented Kalman Filter, nag_kalman_unscented_state_revcom (g13ejc)
can also be used to apply the Unscented Transform (see Julier (2002)) to the function \( F \), by setting
pdlx = 0 and terminating the calling sequence when irevcm = 2 rather than irevcm = 3. In this
situation, on initial entry, x and st would hold the mean and Cholesky factorization of the covariance
matrix of the untransformed sample and on exit (when irevcm = 2) they would hold the mean and
Cholesky factorization of the covariance matrix of the transformed sample.
10 Example

This example implements the following nonlinear state space model, with the state vector \( x \) and state update function \( F \) given by:

\[
\begin{align*}
    m_x &= 3 \\
    x_{t+1} &= (\xi_{t+1}, \eta_{t+1}, \theta_{t+1})^T \\
    &= F(x_t) + v_t \\
    &= x_t + \begin{pmatrix}
        \cos \theta_t & -\sin \theta_t & 0 \\
        \sin \theta_t & \cos \theta_t & 0 \\
        0 & 0 & 1
    \end{pmatrix}
    \begin{pmatrix}
        0.5r & 0.5r & 0 \\
        0 & 0 & r/d \\
        r/d & -r/d & 0
    \end{pmatrix}
    \begin{pmatrix}
        \phi_{rt} \\
        \phi_{lt}
    \end{pmatrix} + v_t
\end{align*}
\]

where \( r \) and \( d \) are known constants and \( \phi_{rt} \) and \( \phi_{lt} \) are time-dependent knowns. The measurement vector \( y \) and measurement function \( H \) is given by:

\[
\begin{align*}
    m_y &= 2 \\
    y_t &= (\delta_t, \alpha_t)^T \\
    &= H(x_t) + u_t \\
    &= \begin{pmatrix}
        \Delta - \xi_t & A - \eta t \sin A
    \end{pmatrix} + u_t
\end{align*}
\]

where \( A \) and \( \Delta \) are known constants. The initial values, \( x_0 \) and \( P_0 \), are given by

\[
\begin{align*}
    x_0 &= \begin{pmatrix}
        0 \\
        0 \\
        0
    \end{pmatrix}, \\
    P_0 &= \begin{pmatrix}
        0.1 & 0 & 0 \\
        0 & 0.1 & 0 \\
        0 & 0 & 0.1
    \end{pmatrix}
\end{align*}
\]

and the Cholesky factorizations of the error covariance matrices, \( L_x \) and \( L_x \) by

\[
\begin{align*}
    L_x &= \begin{pmatrix}
        0.1 & 0 & 0 \\
        0 & 0.1 & 0 \\
        0 & 0 & 0.1
    \end{pmatrix}, \\
    L_y &= \begin{pmatrix}
        0.01 & 0 \\
        0 & 0.01
    \end{pmatrix}
\end{align*}
\]

10.1 Program Text

/* nag_kalman_unscented_state_revcom (g13ejc) Example Program. 
* Copyright 2014 Numerical Algorithms Group. 
* * Mark 25, 2014. */ 
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagx01.h>
#include <naghl3.h>
#include <nagx01.h>
#define LY(I,J) ly[(J) * pdly + (I)]
#define LX(I,J) lx[(J) * pdlx + (I)]
#define ST(I,J) st[(J) * pdst + (I)]
#define XT(I,J) xt[(J) * pdxt + (I)]
#define FXT(I,J) fxt[(J) * pdfxt + (I)]

typedef struct g13_problem_data {
    double delta, a, r, d;
    double phi_rt, phi_lt;
} g13_problem_data;

const Integer mx = 3, my = 2;
void f(Integer n, double *xt, Integer pdxt, double *fxt, Integer pdfxt, g13_problem_data dat);
void h(Integer n, double *xt, Integer pdxt, double *fxt, Integer pdfxt, g13_problem_data dat);
void read_problem_dat(Integer t, g13_problem_data *dat);

int main(void)
{
    /* Integer scalar and array declarations */
    Integer i, irevcm, pdfxt, pdlx, pdly, pdst, pdxt, licomm, lrcomm, lropt,
    n, ntime, t, j;
    Integer *icomm = 0;
    Integer exit_status = 0;

    /* NAG structures and types */
    NagError fail;

    /* Double scalar and array declarations */
    double *fxt = 0, *lx = 0, *ly = 0, *rcomm = 0, *ropt = 0,
    *st = 0, *x = 0, *xt = 0, *y = 0;

    /* Other structures */
    g13_problem_data dat;

    /* Initialise the error structure */
    INIT_FAIL(fail);

    printf("nag_kalman_unscented_state_revcom (g13ejc) "
    "Example Program Results\n\n");

    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[`\n] ");
    #else
    scanf("%*[`\n] ");
    #endif

    /* Using default optional arguments */
    lropt = 0;

    /* Allocate arrays */
    n = 2*mx + 1;
    if (lropt >= 1 && fabs(ropt[0]-2.0)<=0.0) {
        n += 2*mx;
    }
    pdlx = pdst = pdxt = mx;
    pdly = my;
    pdfxt = (mx > my) ? mx : my;
    licomm = 30;
    lrcomm = 30 + my + mx*my + 2*(mx > my) ? mx : my;
    if (!((lx = NAG_ALLOC(pdlx*mx, double)) ||
         (ly = NAG_ALLOC(pdly*my, double)) ||
         (x = NAG_ALLOC(mx, double)) ||
         (st = NAG_ALLOC(pdst*mx, double)) ||
         (xt = NAG_ALLOC(pdfxt*(mx > my) ? mx : my), double)) ||
         (fxt = NAG_ALLOC(pdfxt*(n+(mx > my) ? mx : my), double)) ||
         (licomm = NAG_ALLOC(licomm, Integer)) ||
         (rcomm = NAG_ALLOC(lrcomm, double)) ||
         (y = NAG_ALLOC(my, double)) ) {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Read in the cholesky factorisation of the covariance matrix for the
    process noise */
    for (i = 0; i < mx; i++) {
        for (j = 0; j <= i; j++) {
            #ifdef _WIN32
            scanf_s("%lf",&LX(i,j));
            #else
            scanf("%lf",&LX(i,j));
            #endif
        }
    }

    return 0;
}

Mark 25
#ifdef _WIN32
  scanf_s("%*[\n] ");
#else
  scanf("%*[\n] ");
#endif

}/* Read in the cholesky factorisation of the covariance matrix for the observation noise */
  for (i = 0; i < my; i++) {
    for (j = 0; j <= i; j++) {
      #ifdef _WIN32
        scanf_s("%lf",&LY(i,j));
      #else
        scanf("%lf",&LY(i,j));
      #endif
    }
  }

}/* Read in the initial state vector */
  for (i = 0; i < mx; i++) {
    #ifdef _WIN32
      scanf_s("%lf",&x[i]);
    #else
      scanf("%lf",&x[i]);
    #endif
  }

}/* Read in the cholesky factorisation of the initial state covariance matrix */
  for (i = 0; i < mx; i++) {
    for (j = 0; j <= i; j++) {
      #ifdef _WIN32
        scanf_s("%lf",&ST(i,j));
      #else
        scanf("%lf",&ST(i,j));
      #endif
    }
  }

}/* Read in the number of time points to run the system for */
  #ifdef _WIN32
    scanf_s("%NAG_IFMT"%*[\n] ",&ntime);
  #else
    scanf("%NAG_IFMT"%*[\n] ",&ntime);
  #endif

}/* Read in any problem specific data that is constant */
read_problem_dat(0, &dat);

/* Title for first set of output */
printf(" Time ");
for (i = 0; i < (11*mx- 16)/2; i++) putchar(’ ’);
printf("Estimate of State
 ");
for (i = 0; i < 7+11*mx; i++) putchar(’-’);
printf("\n");
/* Loop over each time point */
irevcm = 0;
for (t = 0; t < ntime; t++) {
    /* Read in any problem specific data that is time dependent */
    read_problem_dat(t+1, &dat);

    /* Read in the observed data for time t */
    for (i = 0; i < my; i++) {
        #ifdef _WIN32
            scanf_s("%lf",&y[i]);
        #else
            scanf("%lf",&y[i]);
        #endif
    }

    #ifdef _WIN32
        scanf_s("%*[\n ");
    #else
        scanf("%*[\n ");
    #endif

    /* Call Unscented Kalman Filter routine (g13ejc) */
    do {
        nag_kalman_unscented_state_revcom(&irevcm, mx, my, y, lx, pdlx, ly, pdly,
                                             x, st, pdst, &n, xt, pdxt, fxt, pdfxt,
                                             ropt, lropt, icomm, licomm, rcomm,
                                             lrcomm, &fail);
        switch(irevcm) {
            case 1:
                /* Evaluate F(X) */
                f(n,xt,pdxt,fxt,pdfxt,dat);
                break;
            case 2:
                /* Evaluate H(X) */
                h(n,xt,pdxt,fxt,pdfxt,dat);
                break;
            default:
                /* irevcm = 3, finished */
                if (fail.code != NE_NOERROR) {
                    printf("Error from nag_kalman_unscented_state_revcom (g13ejc)\.n%s\n",
                           fail.message);
                    exit_status = 1;
                    goto END;
                }
                break;
        }
    } while(irevcm != 3);

    /* Display the some of the current state estimate */
    printf(" %3"NAG_IFMT" ",t+1);
    for (i = 0; i < mx; i++) {
        printf(" %10.3f",x[i]);
    }
    printf("\n");

    printf("%n");
    printf("Estimate of Cholesky Factorisation of the State\n");
    printf("Covariance Matrix at the Last Time Point\n");
    for (i = 0; i < mx; i++) {
        for (j = 0; j <= i; j++) {
            printf(" %10.2e",ST(i,j));
        }
        printf("\n");
    }

    END:
    NAG_FREE(icomm);
NAG_FREE(fxt);
NAG_FREE(lx);
NAG_FREE(ly);
NAG_FREE(rcomm);
NAG_FREE(ropt);
NAG_FREE(st);
NAG_FREE(x);
NAG_FREE(xt);
NAG_FREE(y);

return(exit_status);
}

void f(Integer n, double *xt, Integer pdxt, double *fxt, Integer pdfxt,
       g13_problem_data dat) {
    double t1, t3;
    Integer i;
    t1 = 0.5 * dat.r * (dat.phi.rt+dat.phi.lt);
    t3 = (dat.r/dat.d)*(dat.phi.rt-dat.phi.lt);
    for (i = 0; i < n; i++) {
        FXT(0,i) = XT(0,i) + cos(XT(2,i))*t1;
        FXT(1,i) = XT(1,i) + sin(XT(2,i))*t1;
        FXT(2,i) = XT(2,i) + t3;
    }
}

void h(Integer n, double *xt, Integer pdxt, double *fxt, Integer pdfxt,
       g13_problem_data dat) {
    Integer i;
    for (i = 0; i < n; i++) {
        FXT(0,i) = dat.delta - XT(0,i)*cos(dat.a) - XT(1,i)*sin(dat.a);
        FXT(1,i) = XT(2,i) - dat.a;
        /* Make sure that the theta is in the same range as the observed data,
           which in this case is [0, 2*pi) */
        if (FXT(1,i) < 0.0)
            FXT(1,i) += 2 * X01AAC;
    }
}

void read_problem_dat(Integer t, g13_problem_data *dat) {
    /* Read in any data specific to the f and h functions */
    Integer tt;
    if (t==0) {
        /* Read in the data that is constant across all time points */
        #ifdef _WIN32
            scanf_s("%lf%lf%lf%lf%*\n", &(dat->r), &(dat->d), &(dat->delta),
                    &(dat->a));
        #else
            scanf("%lf%lf%lf%lf%*\n", &(dat->r), &(dat->d), &(dat->delta),
                    &(dat->a));
        #endif
    } else {
        /* Read in data for time point t */
        #ifdef _WIN32
            scanf_s("%NAG_IFMT%lf%lf%*\n", &tt, &(dat->phi.rt), &(dat->phi.lt));
        #else
            scanf("%NAG_IFMT%lf%lf%*\n", &tt, &(dat->phi.rt), &(dat->phi.lt));
        #endif
        if (tt!=t) {
            /* Sanity check */
            printf("Expected to read in data for time point \n",t);
            printf("Data that was read in was for time point \n",tt);
        }
    }
}
10.2 Program Data

nag_kalman_unscented_state_revcom (g13ejc) Example Program Data
0.1
0.0 0.1
0.0 0.0 0.1 :: End of lx
0.01
0.0 0.01 :: End of ly
0.0 0.0 0.0 :: Initial value for x
0.1
0.0 0.1
0.0 0.0 0.1 :: End of initial value for st
15 :: Number of time points
3.0 4.0 5.814 0.464 :: r, d, Delta, A
1 0.4 0.1
5.262 5.923
2 0.4 0.1
4.347 5.783
3 0.4 0.1
3.818 6.181
4 0.4 0.1
2.706 0.085
5 0.4 0.1
1.878 0.442
6 0.4 0.1
0.684 0.836
7 0.4 0.1
0.752 1.300
8 0.4 0.1
0.464 1.700
9 0.4 0.1
0.597 1.781
10 0.4 0.1
0.842 2.040
11 0.4 0.1
1.412 2.286
12 0.4 0.1
1.527 2.820
13 0.4 0.1
2.399 3.147
14 0.4 0.1
2.661 3.569
15 0.4 0.1
3.327 3.659 :: t, phi_rt, phi_lt, y = (delta_t, alpha_a)

10.3 Program Results

nag_kalman_unscented_state_revcom (g13ejc) Example Program Results

<table>
<thead>
<tr>
<th>Time</th>
<th>Estimate of State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.664 -0.092 0.104</td>
</tr>
<tr>
<td>2</td>
<td>1.598 0.081 0.314</td>
</tr>
<tr>
<td>3</td>
<td>2.128 0.213 0.378</td>
</tr>
<tr>
<td>4</td>
<td>3.134 0.674 0.660</td>
</tr>
<tr>
<td>5</td>
<td>3.809 1.181 0.906</td>
</tr>
<tr>
<td>6</td>
<td>4.730 2.000 1.298</td>
</tr>
<tr>
<td>7</td>
<td>4.429 2.474 1.762</td>
</tr>
<tr>
<td>8</td>
<td>4.357 3.246 2.162</td>
</tr>
<tr>
<td>9</td>
<td>3.907 3.852 2.246</td>
</tr>
<tr>
<td>10</td>
<td>3.360 4.398 2.504</td>
</tr>
<tr>
<td>11</td>
<td>2.552 4.741 2.750</td>
</tr>
<tr>
<td>12</td>
<td>2.191 5.193 3.281</td>
</tr>
<tr>
<td>13</td>
<td>1.309 5.018 3.610</td>
</tr>
<tr>
<td>14</td>
<td>1.071 4.894 4.031</td>
</tr>
<tr>
<td>15</td>
<td>0.618 4.322 4.124</td>
</tr>
</tbody>
</table>
Estimate of Cholesky Factorisation of the State Covariance Matrix at the Last Time Point

\[
\begin{pmatrix}
1.92e-01 & \quad -3.82e-01 & \quad 2.22e-02 \\
-3.82e-01 & \quad 1.58e-06 & \quad 2.23e-07 \\
2.22e-02 & \quad 2.23e-07 & \quad 9.95e-03
\end{pmatrix}
\]

The example described above can be thought of as relating to the movement of a hypothetical robot. The unknown state, \( x \), is the position of the robot (with respect to a reference frame) and facing, with \((\xi, \eta)\) giving the \( x \) and \( y \) coordinates and \( \theta \) the angle (with respect to the \( x \)-axis) that the robot is facing. The robot has two drive wheels, of radius \( r \) on an axle of length \( d \). During time period \( t \) the right wheel is believed to rotate at a velocity of \( \phi_{Rt} \) and the left at a velocity of \( \phi_{Lt} \). In this example, these velocities are fixed with \( \phi_{Rt} = 0.4 \) and \( \phi_{Lt} = 0.1 \). The state update function, \( F \), calculates where the robot should be at each time point, given its previous position. However, in reality, there is some random fluctuation in the velocity of the wheels, for example, due to slippage. Therefore the actual position of the robot and the position given by equation \( F \) will differ.

In the area that the robot is moving there is a single wall. The position of the wall is known and defined by its distance, \( \Delta \), from the origin and its angle, \( \alpha \), from the \( x \)-axis. The robot has a sensor that is able to measure \( y \), with \( \delta \) being the distance to the wall and \( \alpha \) the angle to the wall. The measurement function \( H \) gives the expected distance and angle to the wall if the robot’s position is given by \( x_t \). Therefore the state space model allows the robot to incorporate the sensor information to update the estimate of its position.