NAG Library Function Document

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

```
#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:

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

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 Σ_x, and the measurement noise by u_t, which is assumed to have mean zero and covariance structure Σ_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):

$$\mathcal{X}_t = \left[\hat{x}_{t-1} \quad \hat{x}_{t-1} + \gamma\sqrt{P_{t-1}} \quad \hat{x}_{t-1} - \gamma\sqrt{P_{t-1}}\right] \tag{1}$$

(b) Evaluate the known model function F:

$$\mathcal{F}_t = F(\mathcal{X}_t) \tag{2}$$

The function F is assumed to accept the $m_x \times n$ matrix, $\mathcal{X}_t$ and return an $m_x \times n$ matrix, $\mathcal{F}_t$. The columns of both $\mathcal{X}_t$ and $\mathcal{F}_t$ correspond to different possible states. The notation $\mathcal{F}_{t,i}$ is used to denote the ith column of $\mathcal{F}_t$, hence the result of applying F to the ith possible state.

(c) Time Update:

$$\hat{x}_t = \sum_{i=1}^{n} W_i^m \mathcal{F}_{t,i} \tag{3}$$

$$P_t = \sum_{i=1}^{n} W_i^c \left(\mathcal{F}_{t,i} - \hat{x}_t\right)\left(\mathcal{F}_{t,i} - \hat{x}_t\right)^{\mathrm{T}} + \Sigma_x \tag{4}$$

(d) Redraw another set of sigma points (see section Section 3.2):

$$\mathcal{Y}_t = \left[\hat{x}_t \quad \hat{x}_t + \gamma\sqrt{P_t} \quad \hat{x}_t - \gamma\sqrt{P_t}\right] \tag{5}$$

(e) Evaluate the known model function H:

$$\mathcal{H}_t = H\left(\mathcal{Y}_t\right) \tag{6}$$

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

(f) Measurement Update:

$$\hat{y}_t = \sum_{i=1}^{n} W_i^m \mathcal{H}_{t,i} \tag{7}$$

$$P_{yy_t} = \sum_{i=1}^{n} W_i^c \left(\mathcal{H}_{t,i} - \hat{y}_t\right)\left(\mathcal{H}_{t,i} - \hat{y}_t\right)^{\mathrm{T}} + \Sigma_y \tag{8}$$

$$P_{xy_t} = \sum_{i=1}^{n} W_i^c \left(\mathcal{F}_{t,i} - \hat{x}_t\right)\left(\mathcal{H}_{t,i} - \hat{y}_t\right)^{\mathrm{T}} \tag{9}$$

$$\mathcal{K}_t = P_{xy_t} P_{yy_t}^{-1} \tag{10}$$

$$\hat{x}_t = \hat{x}_t + \mathcal{K}_t(y_t - \hat{y}_t) \tag{11}$$

$$P_t = P_t - \mathcal{K}_t P_{yy_t} \mathcal{K}_t^{\mathrm{T}} \tag{12}$$

Here $\mathcal{K}_t$ 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 μ and covariance matrix Σ, the sigma points are usually constructed as:

$$\mathcal{X}_t = \left[\mu \quad \mu + \gamma\sqrt{\Sigma} \quad \mu - \gamma\sqrt{\Sigma}\right]$$

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, γ, are constructed as follows:

$$\lambda = \alpha^2(L+\kappa) - L$$
$$\gamma = \sqrt{L+\lambda}$$
$$W_{\rm i}^m = \begin{cases} \frac{\lambda}{L+\lambda} & i=1 \\ \frac{1}{2(L+\lambda)} & i=2,3,\ldots,2L+1 \end{cases}$$
$$W_{\rm i}^c = \begin{cases} \frac{\lambda}{L+\lambda} + 1 - \alpha^2 + \beta & i=1 \\ \frac{1}{2(L+\lambda)} & i=2,3,\ldots,2L+1 \end{cases}$$

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

$$\mathcal{Y}_t = \left[\mathcal{X}_t \quad \mathcal{X}_{t,1} + \gamma\sqrt{\Sigma_x} \quad \mathcal{X}_{t,1} - \gamma\sqrt{\Sigma_x}\right] \tag{13}$$

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

Julier S J (2002) The scaled unscented transformation *Proceedings of the 2002 American Control Conference (Volume 6)* 4555–4559

Julier S J and Uhlmann J K (1997a) A consistent, debiased method for converting between polar and Cartesian coordinate systems *Proceedings of AeroSense97, International Society for Optics and Phonotonics* 110–121

Julier S J and Uhlmann J K (1997b) A new extension of the Kalman Filter to nonlinear systems *International Symposium for Aerospace/Defense, Sensing, Simulation and Controls (Volume 3)* **26**

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 **irevcm**. Between intermediate exits and re-entries, **all arguments other than fxt must remain unchanged**.

1: **irevcm** – Integer * *Input/Output*

On initial entry: must be set to 0 or 3.

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

On intermediate exit: **irevcm** $= 1$ or 2. The value of **irevcm** specifies what intermediate values are returned by this function and what values the calling program must assign to arguments of 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.

On intermediate re-entry: **irevcm** must remain unchanged.

On final exit: **irevcm** $= 3$

Constraint: **irevcm** $= 0$, 1, 2 or 3.

2: **mx** – Integer *Input*

On entry: m_x, the number of state variables.

Constraint: $\mathbf{mx} \geq 1$.

3: **my** – Integer *Input*

On entry: m_y, the number of observed variables.

Constraint: $\mathbf{my} \geq 1$.

4: **y**[**my**] – const double *Input*

On entry: y_t, the observed data at the current time point.

5: **lx**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **lx** must be at least $\mathbf{pdlx} \times \mathbf{mx}$.

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

On entry: L_x, such that $L_x L_x^{\mathrm{T}} = \Sigma_x$, 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 **lx** is referenced.

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

If Σ_x is time dependent, then the value supplied should be for time t.

6: **pdlx** – Integer *Input*

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

Constraint: $\mathbf{pdlx} = 0$ or $\mathbf{pdlx} \geq \mathbf{mx}$.

7: **ly**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **ly** must be at least $\mathbf{pdly} \times \mathbf{my}$.

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

On entry: L_y, such that $L_y L_y^{\mathrm{T}} = \Sigma_y$, 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 **ly** is referenced.

If Σ_y is time dependent, then the value supplied should be for time t.

8: **pdly** – Integer *Input*

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

Constraint: $\mathbf{pdly} \geq \mathbf{my}$.

9: **x**[**mx**] – double *Input/Output*

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

On intermediate exit: when

$\mathbf{irevcm} = 1$
    x is unchanged.

$\mathbf{irevcm} = 2$
    $\hat{x}_t$.

On intermediate re-entry: **x** must remain unchanged.

On final exit: $\hat{x}_t$ the updated state vector.

10: **st**[dim] – double *Input/Output*

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

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

On initial entry: S_t, such that $S_{t-1}S_{t-1}^{\mathrm{T}} = P_{t-1}$, 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** $\geq$ **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_revcom (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** $\geq 2 \times$ **mx** $+ 1$;
otherwise **n** $\geq 4 \times$ **mx** $+ 1$.

13: **xt**[dim] – double *Input/Output*

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

On initial entry: need not be set.

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

For the jth sigma point, the value for the ith parameter is held in **xt**$[(j-1) \times$ **pdxt** $+ i - 1]$, for $i = 1, 2, \ldots,$ **mx** 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** $\geq$ **mx**.

15: **fxt**[*dim*] – double *Input/Output*

Note: the dimension, *dim*, of the array **fxt** must be at least $\mathbf{pdfxt} \times (\mathbf{n} + \max(\mathbf{mx}, \mathbf{my}))$.

On initial entry: need not be set.

On intermediate exit: the contents of **fxt** are undefined.

On intermediate re-entry: $F(X_t)$ when $\mathbf{irevcm} = 1$, 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 $\mathbf{fxt}[(j-1) \times \mathbf{pdfxt} + i - 1]$, for $j = 1, 2, \ldots, \mathbf{n}$. When $\mathbf{irevcm} = 1$, $i = 1, 2, \ldots, \mathbf{mx}$ and when $\mathbf{irevcm} = 2$, $i = 1, 2, \ldots, \mathbf{my}$.

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: $\mathbf{pdfxt} \geq \max(\mathbf{mx}, \mathbf{my})$.

17: **ropt**[**lropt**] – const double *Input*

On entry: optional arguments. The default value will be used for $\mathbf{ropt}[i-1]$ if $\mathbf{lropt} < i$. Setting $\mathbf{lropt} = 0$ 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., $\mathbf{ropt}[0] = 1$)

ropt[1]
: κ_x, value of κ used when constructing the first set of sigma points, $\mathcal{X}_t$.

Defaults to $3 - \mathbf{mx}$.

ropt[2]
: α_x, value of α used when constructing the first set of sigma points, $\mathcal{X}_t$.

Defaults to 1.

ropt[3]
: β_x, value of β used when constructing the first set of sigma points, $\mathcal{X}_t$.

Defaults to 2.

ropt[4]
: Value of κ used when constructing the second set of sigma points, $\mathcal{Y}_t$.

Defaults to $3 - 2 \times \mathbf{mx}$ when $\mathbf{pdlx} \neq 0$ and the second set of sigma points are augmented and κ_x otherwise.

ropt[5]
: Value of α used when constructing the second set of sigma points, $\mathcal{Y}_t$.

Defaults to α_x.

ropt[6]
: Value of β used when constructing the second set of sigma points, $\mathcal{Y}_t$.

Defaults to β_x.

Constraints:

$\mathbf{ropt}[0] = 1$ or 2;
$\mathbf{ropt}[1] > -\mathbf{mx}$;

ropt$[4] > -2 \times$ **mx** when **pdly** $\neq 0$ and the second set of sigma points are augmented, otherwise **ropt**$[4] > -$**mx**;
ropt$[i-1] > 0$, for $i = 3, 6$.

18: **lropt** – Integer *Input*

On entry: length of the options array **ropt**.

Constraint: $0 \leq$ **lropt** ≤ 7.

19: **icomm**[**licomm**] – Integer *Communication Array*

On initial entry: **icomm** need not be set.

On intermediate exit: **icomm** is used for storage between calls to nag_kalman_unscented_state_revcom (g13ejc).

On intermediate re-entry: **icomm** must remain unchanged.

On final exit: **icomm** is not defined.

20: **licomm** – Integer *Input*

On entry: the length of the array **icomm**. If **licomm** is too small and **licomm** ≥ 2 then **fail**.**code** = NE_TOO_SMALL is returned and the minimum value for **licomm** and **lrcomm** are given by **icomm**[0] and **icomm**[1] respectively.

Constraint: **licomm** ≥ 30.

21: **rcomm**[**lrcomm**] – double *Communication Array*

On initial entry: **rcomm** need not be set.

On intermediate exit: **rcomm** is used for storage between calls to nag_kalman_unscented_state_revcom (g13ejc).

On intermediate re-entry: **rcomm** must remain unchanged.

On final exit: **rcomm** is not defined.

22: **lrcomm** – Integer *Input*

On entry: the length of the array **rcomm**. If **lrcomm** is too small and **licomm** ≥ 2 then **fail**.**code** = NW_INT is returned and the minimum value for **licomm** and **lrcomm** are given by **icomm**[0] and **icomm**[1] respectively.

Suggested value: **lrcomm** $= 30 +$ **my** $+$ **mx** $\times$ **my** $+ (1 + nb) \times \max($**mx**, **my**$)$, where nb is the optimal ***block size***. In most cases a ***block size*** of 128 will be sufficient.

Constraint: **lrcomm** $\geq 30 +$ **my** $+$ **mx** $\times$ **my** $+ 2 \times \max($**mx**, **my**$)$.

23: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_ARRAY_SIZE

On entry, **pdfxt** $= \langle value \rangle$ and **mx** $= \langle value \rangle$.
Constraint: if **irevcm** $= 1$, **pdfxt** $\geq$ **mx**.

On entry, **pdfxt** $= \langle value \rangle$ and **my** $= \langle value \rangle$.
Constraint: if **irevcm** $= 2$, **pdfxt** $\geq$ **my**.

On entry, **pdlx** $= \langle value \rangle$ and **mx** $= \langle value \rangle$.
Constraint: **pdlx** $= 0$ or **pdlx** $\geq$ **mx**.

On entry, **pdly** $= \langle value \rangle$ and **my** $= \langle value \rangle$.
Constraint: **pdly** $\geq$ **my**.

On entry, **pdst** $= \langle value \rangle$ and **mx** $= \langle value \rangle$.
Constraint: **pdst** $\geq$ **mx**.

On entry, **pdxt** $= \langle value \rangle$ and **mx** $= \langle value \rangle$.
Constraint: **pdxt** $\geq$ **mx**.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_ILLEGAL_COMM

icomm has been corrupted between calls.

rcomm has been corrupted between calls.

NE_INT

On entry, **lropt** $= \langle value \rangle$.
Constraint: $0 \leq$ **lropt** ≤ 7.

On entry, **irevcm** $= \langle value \rangle$.
Constraint: **irevcm** $= 0$, 1, 2 or 3.

On entry, **mx** $= \langle value \rangle$.
Constraint: **mx** ≥ 1.

On entry, **my** $= \langle value \rangle$.
Constraint: **my** ≥ 1.

On entry, augmented sigma points requested, **n** $= \langle value \rangle$ and **mx** $= \langle value \rangle$.
Constraint: **n** $\geq \langle value \rangle$.

On entry, redrawn sigma points requested, **n** $= \langle value \rangle$ and **mx** $= \langle value \rangle$.
Constraint: **n** $\geq \langle value \rangle$.

NE_INT_CHANGED

mx has changed between calls.
On intermediate entry, **mx** $= \langle value \rangle$.
On initial entry, **mx** $= \langle value \rangle$.

my has changed between calls.
On intermediate entry, **my** $= \langle value \rangle$.
On initial entry, **my** $= \langle value \rangle$.

n has changed between calls.
On intermediate entry, **n** $= \langle value \rangle$.
On intermediate exit, **n** $= \langle value \rangle$.

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: $\kappa >$ ⟨*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** ≥ 2.
icomm is too small to return the required array sizes.

NW_INT

On entry, **licomm** = ⟨*value*⟩ and **lrcomm** = ⟨*value*⟩.
Constraint: **licomm** ≥ 30 and **lrcomm** $\geq 30 + \mathbf{my} + \mathbf{mx} \times \mathbf{my} + 2 \times \max(\mathbf{mx}, \mathbf{my})$.
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:

$$\begin{aligned} m_x &= 3 \\ x_{t+1} &= \begin{pmatrix} \xi_{t+1} & \eta_{t+1} & \theta_{t+1} \end{pmatrix}^{\mathrm{T}} \\ &= F(x_t) + v_t \\ &= x_t + \begin{pmatrix} \cos\theta_t & -\sin\theta_t & 0 \\ \sin\theta_t & \cos\theta_t & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.5r & 0.5r \\ 0 & 0 \\ r/d & -r/d \end{pmatrix} \begin{pmatrix} \phi_{Rt} \\ \phi_{Lt} \end{pmatrix} + v_t \end{aligned}$$

where r and d are known constants and ϕ_{Rt} and ϕ_{Lt} are time-dependent knowns. The measurement vector y and measurement function H is given by:

$$\begin{aligned} m_y &= 2 \\ y_t &= (\delta_t, \alpha_t)^{\mathrm{T}} \\ \\ &= H(x_t) + u_t \\ &= \begin{pmatrix} \Delta - \xi_t \cos A - \eta_t \sin A \\ \theta_t - A \end{pmatrix} + u_t \end{aligned}$$

where A and Δ are known constants. The initial values, x_0 and P_0, are given by

$$x_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad P_0 = \begin{pmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{pmatrix}$$

and the Cholesky factorizations of the error covariance matrices, L_x and L_x by

$$L_{\mathrm{x}} = \begin{pmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{pmatrix}, \quad L_{\mathrm{y}} = \begin{pmatrix} 0.01 & 0 \\ 0 & 0.01 \end{pmatrix}.$$

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 <nagg13.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(pdxt*(my > n ? my : n), double)) ||
      !(fxt = NAG_ALLOC(pdfxt*(n+(mx > my ? mx : my)), double)) ||
      !(icomm = 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
    }
```

```
#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
    }
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#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
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#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
    }
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#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\n ");
  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 %"NAG_IFMT"\n",t);
      printf("Data that was read in was for time point %"NAG_IFMT"\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

   Time          Estimate of State
 ---------------------------------------
   1          0.664     -0.092      0.104
   2          1.598      0.081      0.314
   3          2.128      0.213      0.378
   4          3.134      0.674      0.660
   5          3.809      1.181      0.906
   6          4.730      2.000      1.298
   7          4.429      2.474      1.762
   8          4.357      3.246      2.162
   9          3.907      3.852      2.246
  10          3.360      4.398      2.504
  11          2.552      4.741      2.750
  12          2.191      5.193      3.281
  13          1.309      5.018      3.610
  14          1.071      4.894      4.031
  15          0.618      4.322      4.124
```

```
Estimate of Cholesky Factorisation of the State
Covariance Matrix at the Last Time Point
   1.92e-01
  -3.82e-01   2.22e-02
   1.58e-06   2.23e-07   9.95e-03
```

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 (ξ, η) giving the x and y coordinates and θ 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 ϕ_{Rt} and the left at a velocity of ϕ_{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, Δ, from the origin and its angle, A, from the x-axis. The robot has a sensor that is able to measure y, with δ being the distance to the wall and α 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.

Example Program
Illustration of Position and Orientation
of Hypothetical Robot