

# NAG Library Routine Document

## G13EKF

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details.

### 1 Purpose

G13EKF applies the Unscented Kalman Filter (UKF) to a nonlinear state space model, with additive noise.

G13EKF uses direct communication for evaluating the nonlinear functionals of the state space model.

### 2 Specification

```

SUBROUTINE G13EKF (MX, MY, Y, LX, LY, F, H, X, ST, IUSER, RUSER, IFAIL)
INTEGER          MX, MY, IUSER(*), IFAIL
REAL (KIND=nag_wp) Y(MY), LX(MX,MX), LY(MY,MY), X(MX), ST(MX,MX),      &
RUSER(*)
EXTERNAL        F, H

```

### 3 Description

G13EKF 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  $\Sigma_x$ , 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 3.2):

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

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

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

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  $i$ th column of  $\mathcal{F}_t$ , hence the result of applying  $F$  to the  $i$ th possible state.

- (c) Time Update:

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

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

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

$$\mathcal{Y}_t = \begin{bmatrix} \hat{x}_t & \hat{x}_t + \gamma\sqrt{P_t} & \hat{x}_t - \gamma\sqrt{P_t} \end{bmatrix} \quad (5)$$

(e) Evaluate the known model function  $H$ :

$$\mathcal{H}_t = H(\mathcal{Y}_t) \quad (6)$$

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  $i$ th column of  $\mathcal{H}_t$ .

(f) Measurement Update:

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

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

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

$$\mathcal{K}_t = P_{xy_t} P_{yy_t}^{-1} \quad (10)$$

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

$$P_t = P_t - \mathcal{K}_t P_{yy_t} \mathcal{K}_t^T \quad (12)$$

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 G13EKF 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:

$$\mathcal{X}_t = \begin{bmatrix} \mu & \mu + \gamma\sqrt{\Sigma} & \mu - \gamma\sqrt{\Sigma} \end{bmatrix}$$

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:

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

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.

The constants,  $\kappa$ ,  $\alpha$  and  $\beta$  are given by  $\kappa = 3 - m_x$ ,  $\alpha = 1.0$  and  $\beta = 2$ . If more control is required over the construction of the sigma points then the reverse communication routine, G13EJF, can be used instead.

## 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, unbiased method for converting between polar and Cartesian coordinate systems *Proceedings of AeroSense97, International Society for Optics and Photonics* 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

- 1: MX – INTEGER *Input*  
*On entry:*  $m_x$ , the number of state variables.  
*Constraint:*  $MX \geq 1$ .
- 2: MY – INTEGER *Input*  
*On entry:*  $m_y$ , the number of observed variables.  
*Constraint:*  $MY \geq 1$ .
- 3: Y(MY) – REAL (KIND=nag\_wp) array *Input*  
*On entry:*  $y_t$ , the observed data at the current time point.
- 4: LX(MX, MX) – REAL (KIND=nag\_wp) array *Input*  
*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 LX is referenced.  
 If  $\Sigma_x$  is time dependent, then the value supplied should be for time  $t$ .
- 5: LY(MY, MY) – REAL (KIND=nag\_wp) array *Input*  
*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 LY is referenced.  
 If  $\Sigma_y$  is time dependent, then the value supplied should be for time  $t$ .
- 6: F – SUBROUTINE, supplied by the user. *External Procedure*  
 The state function,  $F$  as described in (b).

The specification of F is:

```
SUBROUTINE F (MX, N, XT, FXT, IUSER, RUSER, INFO)
  INTEGER          MX, N, IUSER(*), INFO
  REAL (KIND=nag_wp) XT(MX,N), FXT(MX,N), RUSER(*)
```

- 1: MX – INTEGER *Input*  
*On entry:*  $m_x$ , the number of state variables.

2:	N – INTEGER	<i>Input</i>
	<i>On entry:</i> $n$ , the number of sigma points.	
3:	XT(MX,N) – REAL (KIND=nag_wp) array	<i>Input</i>
	<i>On entry:</i> $X_t$ , the sigma points generated in (a). For the $j$ th sigma point, the value for the $i$ th parameter is held in $XT(i,j)$ , for $i = 1, 2, \dots, m_x$ and $j = 1, 2, \dots, n$ .	
4:	FXT(MX,N) – REAL (KIND=nag_wp) array	<i>Output</i>
	<i>On exit:</i> $F(X_t)$ .	
	For the $j$ th sigma point the value for the $i$ th parameter should be held in $FXT(i,j)$ , for $i = 1, 2, \dots, m_x$ and $j = 1, 2, \dots, n$ .	
5:	IUSER(*) – INTEGER array	<i>User Workspace</i>
6:	RUSER(*) – REAL (KIND=nag_wp) array	<i>User Workspace</i>
	F is called with the arguments IUSER and RUSER as supplied to G13EKF. You should use the arrays IUSER and RUSER to supply information to F.	
7:	INFO – INTEGER	<i>Input/Output</i>
	<i>On entry:</i> INFO = 0.	
	<i>On exit:</i> set INFO to a nonzero value if you wish G13EKF to terminate with IFAIL = 61.	

F must either be a module subprogram USED by, or declared as EXTERNAL in, the (sub) program from which G13EKF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

- 7: H – SUBROUTINE, supplied by the user. *External Procedure*  
 The measurement function,  $H$  as described in (e).

The specification of H is:		
SUBROUTINE H (MX, MY, N, YT, HYT, IUSER, RUSER, INFO)		
INTEGER                           MX, MY, N, IUSER(*), INFO		
REAL (KIND=nag_wp) YT(MX,N), HYT(MY,N), RUSER(*)		
1:	MX – INTEGER	<i>Input</i>
	<i>On entry:</i> $m_x$ , the number of state variables.	
2:	MY – INTEGER	<i>Input</i>
	<i>On entry:</i> $m_y$ , the number of observed variables.	
3:	N – INTEGER	<i>Input</i>
	<i>On entry:</i> $n$ , the number of sigma points.	
4:	YT(MX,N) – REAL (KIND=nag_wp) array	<i>Input</i>
	<i>On entry:</i> $Y_t$ , the sigma points generated in (d). For the $j$ th sigma point, the value for the $i$ th parameter is held in $YT(i,j)$ , for $i = 1, 2, \dots, m_x$ and $j = 1, 2, \dots, n$ , where $m_x$ is the number of state variables and $n$ is the number of sigma points.	
5:	HYT(MY,N) – REAL (KIND=nag_wp) array	<i>Output</i>
	<i>On exit:</i> $H(Y_t)$ .	

For the  $j$ th sigma point the value for the  $i$ th parameter should be held in  $\text{HYT}(i, j)$ , for  $i = 1, 2, \dots, m_y$  and  $j = 1, 2, \dots, n$ .

- 6: IUSER(\*) – INTEGER array *User Workspace*  
 7: RUSER(\*) – REAL (KIND=nag\_wp) array *User Workspace*

H is called with the arguments IUSER and RUSER as supplied to G13EKF. You should use the arrays IUSER and RUSER to supply information to H.

- 8: INFO – INTEGER *Input/Output*  
*On entry:* INFO = 0.  
*On exit:* set INFO to a nonzero value if you wish G13EKF to terminate with IFAIL = 71.

H must either be a module subprogram USED by, or declared as EXTERNAL in, the (sub) program from which G13EKF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

- 8: X(MX) – REAL (KIND=nag\_wp) array *Input/Output*

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

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

- 9: ST(MX, MX) – REAL (KIND=nag\_wp) array *Input/Output*

*On entry:*  $S_t$ , such that  $S_{t-1}S_{t-1}^T = P_{t-1}$ , 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 ST is referenced.

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

- 10: IUSER(\*) – INTEGER array *User Workspace*  
 11: RUSER(\*) – REAL (KIND=nag\_wp) array *User Workspace*

IUSER and RUSER are not used by G13EKF, but are passed directly to F and H and should be used to pass information to these routines.

- 12: IFAIL – INTEGER *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this argument, the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

*On exit:* IFAIL = 0 unless the routine detects an error or a warning has been flagged (see Section 6).

## 6 Error Indicators and Warnings

If on entry  $IFAIL = 0$  or  $-1$ , explanatory error messages are output on the current error message unit (as defined by  $X04AAF$ ).

Errors or warnings detected by the routine:

$IFAIL = 11$

On entry,  $MX = \langle value \rangle$ .  
Constraint:  $MX \geq 1$ .

$IFAIL = 21$

On entry,  $MY = \langle value \rangle$ .  
Constraint:  $MY \geq 1$ .

$IFAIL = 61$

User requested termination in F.

$IFAIL = 71$

User requested termination in H.

$IFAIL = 301$

A weight was negative and it was not possible to downdate the Cholesky factorization.

$IFAIL = 302$

Unable to calculate the Kalman gain matrix.

$IFAIL = 303$

Unable to calculate the Cholesky factorization of the updated state covariance matrix.

$IFAIL = -99$

An unexpected error has been triggered by this routine. Please contact NAG.  
See Section 3.9 in *How to Use the NAG Library and its Documentation* for further information.

$IFAIL = -399$

Your licence key may have expired or may not have been installed correctly.  
See Section 3.8 in *How to Use the NAG Library and its Documentation* for further information.

$IFAIL = -999$

Dynamic memory allocation failed.  
See Section 3.7 in *How to Use the NAG Library and its Documentation* for further information.

## 7 Accuracy

Not applicable.

## 8 Parallelism and Performance

G13EKF 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 routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

## 9 Further Comments

None.

## 10 Example

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

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

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

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

where  $A$  and  $\Delta$  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  $L_y$  by

$$L_x = \begin{pmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{pmatrix}, \quad L_y = \begin{pmatrix} 0.01 & 0 \\ 0 & 0.01 \end{pmatrix}$$

### 10.1 Program Text

```
! G13EKF Example Program Text
! Mark 26 Release. NAG Copyright 2016.

Module g13ekfe_mod

! G13EKF Example Program Module:
! User-defined Routines

! .. Use Statements ..
Use nag_library, Only: nag_wp
! .. Implicit None Statement ..
Implicit None
! .. Accessibility Statements ..
Private
Public :: f, h, read_problem_data
! .. Parameters ..
Integer, Parameter, Public :: mx = 3, my = 2, nin = 5, nout = 6
Contains
Subroutine f(mx,n,xt,fxt,iuser,ruser,info)
```

```

! .. Scalar Arguments ..
Integer, Intent (Inout)      :: info
Integer, Intent (In)        :: mx, n
! .. Array Arguments ..
Real (Kind=nag_wp), Intent (Out) :: fxt(mx,n)
Real (Kind=nag_wp), Intent (Inout) :: ruser(*)
Real (Kind=nag_wp), Intent (In) :: xt(mx,n)
Integer, Intent (Inout)      :: iuser(*)
! .. Local Scalars ..
Real (Kind=nag_wp)          :: d, phi_lt, phi_rt, r, t1, t3
Integer                      :: i
! .. Intrinsic Procedures ..
Intrinsic                   :: cos, sin
! .. Executable Statements ..
Continue

r = ruser(3)
d = ruser(4)
phi_rt = ruser(5)
phi_lt = ruser(6)

t1 = 0.5_nag_wp*r*(phi_rt+phi_lt)
t3 = (r/d)*(phi_rt-phi_lt)

Do i = 1, n
  fxt(1,i) = xt(1,i) + cos(xt(3,i))*t1
  fxt(2,i) = xt(2,i) + sin(xt(3,i))*t1
  fxt(3,i) = xt(3,i) + t3
End Do

! Set info nonzero to terminate execution for any reason.
info = 0

Return
End Subroutine f
Subroutine h(mx,my,n,yt,hyt,iuser,ruser,info)

! .. Use Statements ..
Use nag_library, Only: x01aaf
! .. Scalar Arguments ..
Integer, Intent (Inout)      :: info
Integer, Intent (In)        :: mx, my, n
! .. Array Arguments ..
Real (Kind=nag_wp), Intent (Out) :: hyt(my,n)
Real (Kind=nag_wp), Intent (Inout) :: ruser(*)
Real (Kind=nag_wp), Intent (In) :: yt(mx,n)
Integer, Intent (Inout)      :: iuser(*)
! .. Local Scalars ..
Real (Kind=nag_wp)          :: a, delta, tmp
Integer                      :: i
! .. Intrinsic Procedures ..
Intrinsic                   :: cos, sin
! .. Executable Statements ..
Continue

delta = ruser(1)
a = ruser(2)

Do i = 1, n
  hyt(1,i) = delta - yt(1,i)*cos(a) - yt(2,i)*sin(a)
  hyt(2,i) = yt(3,i) - a

! Make sure that the theta is in the same range as the observed
! data, which in this case is [0, 2*pi)
  If (hyt(2,i)<0.0_nag_wp) Then
    hyt(2,i) = hyt(2,i) + 2*x01aaf(tmp)
  End If
End Do

! Set info nonzero to terminate execution for any reason.
info = 0

```



```

      Return
End Subroutine h
Subroutine read_problem_data(t,iuser,ruser,read_ok)
!   Read in any data specific to the F and H subroutines

!   .. Scalar Arguments ..
Integer, Intent (In)           :: t
Logical, Intent (Out)          :: read_ok
!   .. Array Arguments ..
Real (Kind=nag_wp), Allocatable, Intent (Inout) :: ruser(:)
Integer, Allocatable, Intent (Inout) :: iuser(:)
!   .. Local Scalars ..
Real (Kind=nag_wp)            :: a, d, delta, phi_lt, phi_rt, r
Integer                        :: tt
!   .. Executable Statements ..
Continue

If (t==0) Then
!   Allocate the arrays to hold the data
Allocate (ruser(6),iuser(0))

!   Read in the data that is constant across all time points
Read (nin,*) r, d, delta, a

!   Store the data in RUSER
ruser(1) = delta
ruser(2) = a
ruser(3) = r
ruser(4) = d

read_ok = .True.
Else
!   Read in data for time point t
Read (nin,*) tt, phi_rt, phi_lt
If (tt/=t) Then
!   Sanity check
Write (nout,99999) 'Expected to read in data for time point ', t
Write (nout,99999) 'Data that was read in was for time point ', tt
99999 Format (A,E22.15)
read_ok = .False.
Else
read_ok = .True.
End If

!   Store the data in RUSER
ruser(5) = phi_rt
ruser(6) = phi_lt

End If
End Subroutine read_problem_data
End Module g13ekfe_mod

Program g13ekfe

!   .. Use Statements ..
Use nag_library, Only: g13ekf, nag_wp
Use g13ekfe_mod, Only: f, h, mx, my, nin, nout, read_problem_data
!   .. Implicit None Statement ..
Implicit None
!   .. Local Scalars ..
Integer                :: i, ifail, ntime, t
Logical                :: read_ok
!   .. Local Arrays ..
Real (Kind=nag_wp), Allocatable :: lx(:,,:), ly(:,,:), ruser(:), st(:,,:), &
x(:), y(:)
Integer, Allocatable :: iuser(:)
!   .. Intrinsic Procedures ..
Intrinsic              :: repeat
!   .. Executable Statements ..
Write (nout,*) 'G13EKF Example Program Results'
```

```

Write (nout,*)

! Skip heading in data file
Read (nin,*)

! Allocate arrays
Allocate (lx(mx,mx),ly(my,my),x(mx),st(mx,mx),y(my))

! Read in the Cholesky factorization of the covariance matrix for the
! process noise
Do i = 1, mx
  Read (nin,*) lx(i,1:i)
End Do

! Read in the Cholesky factorization of the covariance matrix for the
! observation noise
Do i = 1, my
  Read (nin,*) ly(i,1:i)
End Do

! Read in the initial state vector
Read (nin,*) x(1:mx)

! Read in the Cholesky factorization of the initial state covariance
! matrix
Do i = 1, mx
  Read (nin,*) st(i,1:i)
End Do

! Read in the number of time points to run the system for
Read (nin,*) ntime

! Read in any problem specific data that is constant
Call read_problem_data(0,iuser,ruser,read_ok)
If (.Not. read_ok) Then
  Go To 100
End If

! Title for first set of output
Write (nout,*) ' Time      ', repeat(' ',(11*mx-16)/2), 'Estimate of State'
Write (nout,*) repeat('-',7+11*mx)

! Loop over each time point
Do t = 1, ntime

!   Read in any problem specific data that is time dependent
Call read_problem_data(t,iuser,ruser,read_ok)
If (.Not. read_ok) Then
  Go To 100
End If

!   Read in the observed data for time t
Read (nin,*) y(1:my)

!   Call Unscented Kalman Filter routine
ifail = 0
Call g13ekf(mx,my,y,lx,ly,f,h,x,st,iuser,ruser,ifail)

!   Display the some of the current state estimate
Write (nout,99999) t, x(1:mx)
End Do

Write (nout,*)
Write (nout,*) 'Estimate of Cholesky Factorization of the State'
Write (nout,*) 'Covariance Matrix at the Last Time Point'
Do i = 1, mx
  Write (nout,99998) st(i,1:i)
End Do

```

```

100  Continue

99999 Format (1X,I3,4X,10(1X,F10.3))
99998 Format (10(1X,E10.3))
      End Program g13ekfe

```

## 10.2 Program Data

```

G13EKF 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

G13EKF 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 Factorization of the State
Covariance Matrix at the Last Time Point
0.192E+00
-0.382E+00  0.222E-01
0.158E-05  0.223E-06  0.995E-02
    
```

The example described above can be thought of 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,  $A$ , 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.

**Example Program**  
 Illustration of Position and Orientation  
 of a Hypothetical Robot

