

## nag\_kalman\_sqrt\_filt\_info\_var (g13ecc)

### 1. Purpose

**nag\_kalman\_sqrt\_filt\_info\_var (g13ecc)** performs a combined measurement and time update of one iteration of the time-varying Kalman filter. The method employed for this update is the square root information filter with the system matrices in their original form.

### 2. Specification

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

void g13ecc(Integer n, Integer m, Integer p, Nag_ab_input inp_ab,
            double t[], Integer tdt, double ainv[], Integer tda,
            double b[], Integer tdb, double rinv[], Integer tdr,
            double c[], Integer tdc, double qinv[], Integer tdc,
            double x[], double rinvy[], double z[], double tol,
            NagError *fail)
```

### 3. Description

For the state space system defined by

$$\begin{aligned} X_{i+1} &= A_i X_i + B_i W_i & \text{var}(W_i) &= Q_i \\ Y_i &= C_i X_i + V_i & \text{var}(V_i) &= R_i \end{aligned}$$

the estimate of  $X_i$  given observations  $Y_1$  to  $Y_{i-1}$  is denoted by  $\hat{X}_{i|i-1}$  with  $\text{var}(\hat{X}_{i|i-1}) = P_{i|i-1} = S_i S_i^T$ .

The function performs one recursion of the square root information filter algorithm, summarized as follows:

$$U_1 \begin{pmatrix} Q_i^{-1/2} & 0 & Q_i^{-1/2} \bar{w}_i \\ S_i^{-1} A_i^{-1} B_i & S_i^{-1} A_i^{-1} & S_i^{-1} \hat{X}_{i|i} \\ 0 & R_{i+1}^{-1/2} C_{i+1} & R_{i+1}^{-1/2} Y_{i+1} \end{pmatrix} = \begin{pmatrix} F_{i+1}^{-1/2} & * & * \\ 0 & S_{i+1}^{-1} & \xi_{i+1|i+1} \\ 0 & 0 & E_{i+1} \end{pmatrix}$$

(Pre-array) (Post-array)

where  $U_1$  is an orthogonal transformation triangularizing the pre-array. The triangularization is done entirely via Householder transformations exploiting the zero pattern of the pre-array. The term  $\bar{w}_i$  is the mean process noise and  $E_{i+1}$  is the estimated error at instant  $i+1$ . The inverse of the state covariance matrix  $P_{i|i}$  is factored as follows

$$P_{i|i}^{-1} = (S_i^{-1})^T S_i^{-1}$$

where  $P_{i|i} = S_i S_i^T$  ( $S_i$  is lower triangular).

The new state filtered state estimate is computed via

$$\hat{X}_{i+1|i+1} = S_{i+1} \xi_{i+1|i+1}$$

The function returns  $S_{i+1}^{-1}$  and  $\hat{X}_{i+1|i+1}$  (see the Introduction to Chapter g13 for more information concerning the information filter).

## 4. Parameters

**n**Input: The actual state dimension,  $n$ , i.e., the order of the matrices  $S_i$  and  $A_i^{-1}$ .Constraint:  $\mathbf{n} \geq 1$ .**m**Input: The actual input dimension,  $m$ , i.e., the order of the matrix  $Q_i^{-1/2}$ .Constraint:  $\mathbf{m} \geq 1$ .**p**Input : The actual output dimension,  $p$ , i.e., the order of the matrix  $R_{i+1}^{-1/2}$ .Constraint:  $\mathbf{p} \geq 1$ .**inp\_ab**Input: Indicates how the matrix  $B_i$  is to be passed to the function.If **inp\_ab** = **Nag\_ab\_prod**, then array **b** must contain the product  $A_i^{-1}B_i$ .If **inp\_ab** = **Nag\_ab\_sep**, then array **b** must contain  $B_i$ .**t[n][tdt]**Input: The leading  $n$  by  $n$  upper triangular part of this array must contain  $S_i^{-1}$  the square root of the inverse of the state covariance matrix  $P_{i|i}$ .Output: The leading  $n$  by  $n$  upper triangular part of this array contains  $S_{i+1}^{-1}$ , the square root of the inverse of the of the state covariance matrix  $P_{i+1|i+1}$ .**tdt**Input: The trailing dimension of array **t** as declared in the calling program.Constraint: **tdt**  $\geq$  **n**.**ainv[n][tda]**Input: The leading  $n$  by  $n$  part of this array must contain  $A_i^{-1}$  the inverse of the state transition matrix.**tda**Input: The trailing dimension of array **ainv** as declared in the calling program.Constraint: **tda**  $\geq$  **n**.**b[n][tdb]**Input: The leading  $n$  by  $m$  part of this array must contain  $B_i$  (if **inp\_ab** = **Nag\_ab\_sep**) or its product with  $A_i^{-1}$  (if **inp\_ab** = **Nag\_ab\_prod**).**tdb**Input : The trailing dimension of array **b** as declared in the calling program.Constraint: **tdb**  $\geq$  **m**.**rinv[p][tdr]**Input: If the measurement noise covariance matrix is to be supplied separately from the output weight matrix, then the leading  $p$  by  $p$  upper triangular part of this array must contain  $R_{i+1}^{-1/2}$ , the right Cholesky factor of the inverse of the measurement noise covariance matrix. If this information is not to be input separately from the output weight matrix (see below) then the array **rinv** must be set to the null pointer, i.e., (double \*)0.

**tdr**

Input: The trailing dimension of array **rinv** as declared in the calling program.

Constraint: **tdr**  $\geq$  **p** if **rinv** is defined.

**c[p][tdc]**

Input: The leading  $p$  by  $n$  part of this array must contain  $C_{i+1}$ , the output weight matrix (or its product with  $R_{i+1}^{-1/2}$  if the array **rinv** has been set to the null pointer (double \*)0) of the discrete system at instant  $i + 1$ .

**tdc**

Input: The trailing dimension of array **c** as declared in the calling program.

Constraint: **tdc**  $\geq$  **n**.

**qinv[m][tdq]**

Input: The leading  $m$  by  $m$  upper triangular part of this array must contain  $Q_i^{-1/2}$  the right Cholesky factor of the inverse of the process noise covariance matrix.

**tdq**

Input: The trailing dimension of array **q** as declared in the calling program.

Constraint: **tdq**  $\geq$  **m**.

**x[n]**

Input: This array must contain the estimated state  $\hat{X}_{i|i}$ .

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

**rinvy[p]**

Input: This array must contain  $R_{i+1}^{-1/2}Y_{i+1}$ , the product of the upper triangular matrix  $R_{i+1}^{-1/2}$  and the measured output  $Y_{i+1}$ .

**z[m]**

Input: This array must contain  $\bar{w}_i$ , the mean value of the state process noise.

**tol**

Input: **tol** is used to test for near singularity of the matrix  $S_{i+1}^{-1}$ . If the user sets **tol** to be less than  $n^2 \times \epsilon$  then the tolerance is taken as  $n^2 \times \epsilon$ , where  $\epsilon$  is the **machine precision**.

**fail**

The NAG error parameter, see the Essential Introduction to the NAG C Library.

## 5. Error Indications and Warnings

### NE\_BAD\_PARAM

On entry parameter **inp\_ab** had an illegal value.

### NE\_INT\_ARG\_LT

On entry, **n** must not be less than 1: **n** = *value*.

On entry, **m** must not be less than 1: **m** = *value*.

On entry, **p** must not be less than 1: **p** = *value*.

**NE\_2\_INT\_ARG\_LT**

On entry **tdt** = *<value>* while **n** = *<value>*.  
These parameters must satisfy **tdt**  $\geq$  **n**.

On entry **tda** = *<value>* while **n** = *<value>*.  
These parameters must satisfy **tda**  $\geq$  **n**.

On entry **tdb** = *<value>* while **m** = *<value>*.  
These parameters must satisfy **tdb**  $\geq$  **m**.

On entry **tdc** = *<value>* while **n** = *<value>*.  
These parameters must satisfy **tdc**  $\geq$  **n**.

On entry **tdq** = *<value>* while **m** = *<value>*.  
These parameters must satisfy **tdq**  $\geq$  **m**.

On entry **tdr** = *<value>* while **p** = *<value>*.  
These parameters must satisfy **tdr**  $\geq$  **p**.

**NE\_MAT\_SINGULAR**

The matrix inverse(S) is singular.

**NE\_ALLOC\_FAIL**

Memory allocation failed.

**6. Further Comments**

The algorithm requires approximately  $\frac{7}{6}n^3 + n^2(\frac{7}{2}m + p) + n(\frac{1}{2}p^2 + m^2)$  operations and is backward stable (see Verhaegen and Van Dooren 1986).

**6.1. Accuracy**

The use of the square root algorithm improves the stability of the computations.

**6.2. References**

- Anderson B D O and Moore J B (1979) *Optimal Filtering* Prentice Hall, Englewood Cliffs, New Jersey.
- Vanbegin M, Van Dooren P and Verhaegen M H G (1989) Algorithm 675: FORTRAN Subroutines for Computing the Square Root Covariance Filter and Square Root Information Filter in Dense or Hessenberg Forms *ACM Trans. Math. Software* **15** 243–256.
- Verhaegen M H G and Van Dooren P (1986) Numerical Aspects of Different Kalman Filter Implementations *IEEE Trans. Auto. Contr.* **AC-31** 907–917.

**7. See Also**

nag\_kalman\_sqrt\_filt\_info\_invar (g13edc)

**8. Example**

To apply three iterations of the Kalman filter (in square root information form) to the system  $(A_i^{-1}, A_i^{-1}B_i, C_{i+1})$ . The same data is used for all three iterative steps.

## 8.1. Program Text

```

/* nag_kalman_sqrt_filt_info_var(g13ecc) Example Program
 *
 * Copyright 1994 Numerical Algorithms Group
 *
 * Mark 3, 1994.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagg13.h>

#define NMAX 20
#define MMAX 20
#define PMAX 20

#define TDA NMAX
#define TDB MMAX
#define TDC NMAX
#define TDQ MMAX
#define TDR PMAX
#define TDT NMAX

main()
{
    double ainv[NMAX][TDA];
    double qinv[MMAX][TDQ];
    double rinv[PMAX][TDR];
    double t[NMAX][TDT];
    double b[NMAX][TDB];
    double c[PMAX][TDC];

    Integer i, j, m, n, p;
    double x[NMAX], z[MMAX];
    Integer istep;
    double rinvy[PMAX];
    Nag_ab_input inp_ab;
    double tol;
    Integer nmax, mmax, pmax;

    Vprintf("g13ecc Example Program Results\n");

    /* Skip the heading in the data file */
    Vscanf("%*[^\\n]");

    nmax = NMAX;
    mmax = MMAX;
    pmax = PMAX;

    Vscanf("%ld%ld%ld%lf",&n,&m,&p,&tol);
    if (n<=0 || m<=0 || p<=0 ||
        n>nmax || m>mmax || p>pmax)
    {
        Vfprintf(stderr, "One of n m or p is out of range \
n = %ld, m = %ld, p = %ld\n", n, m, p);
        exit(EXIT_FAILURE);
    }

    inp_ab = Nag_ab_prod;

    /* Read data */
    for (i=0; i<n; ++i)
        for (j=0; j<n; ++j)
            Vscanf("%lf", &ainv[i][j]);
    for (i=0; i<p; ++i)
        for (j=0; j<n; ++j)
            Vscanf("%lf", &c[i][j]);

```

```

if (rinv)
  for (i=0; i<p; ++i)
    for (j=0; j<p; ++j)
      Vscanf("%lf", &rinv[i][j]);
for (i=0; i<n; ++i)
  for (j=0; j<m; ++j)
    Vscanf("%lf", &b[i][j]);
for (i=0; i<m; ++i)
  for (j=0; j<m; ++j)
    Vscanf("%lf", &qinv[i][j]);
for (i=0; i<n; ++i)
  for (j=0; j<n; ++j)
    Vscanf("%lf", &t[i][j]);
for (j=0; j<m; ++j)
  Vscanf("%lf", &z[j]);
for (j=0; j<n; ++j)
  Vscanf("%lf", &x[j]);
for (j=0; j<p; ++j)
  Vscanf("%lf", &rinvy[j]);

/* Perform three iterations of the (Kalman) filter recursion
   (in square root information form). */
for (istep=1; istep<=3; ++istep)
  g13ecc(n, m, p, inp_ab, (double *)t, (Integer)TDT,
        (double *)ainv, (Integer)TDA, (double *)b,
        (Integer)TDB, (double *)rinv, (Integer)TDR,
        (double *)c, (Integer)TDC, (double *)qinv,
        (Integer)TDQ, x, rinvy, z,
        tol, NAGERR_DEFAULT);

Vprintf("\nThe inverse of the square root of the state covariance \
matrix is\n\n");
for (i=0; i<n; ++i)
  {
    for (j=0; j<n; ++j)
      Vprintf ("%8.4f ", t[i][j]);
    Vprintf ("\n");
  }

Vprintf ("\nThe components of the estimated filtered state are\n\n");
Vprintf("k      x(k)  \n");
for (i=0; i<n; ++i)
  {
    Vprintf ("%ld ", i);
    Vprintf (" %8.4f  \n", x[i]);
  }

exit(EXIT_SUCCESS);
}

```

## 8.2. Program Data

f10acc Example Program Data

4	2	2	0.0
0.2113	0.7560	0.0002	0.3303
0.8497	0.6857	0.8782	0.0683
0.7263	0.1985	0.5442	0.2320
0.8833	0.6525	0.3076	0.9329
0.3616	0.5664	0.5015	0.2693
0.2922	0.4826	0.4368	0.6325
1.0000	0.0000		
0.0000	1.0000		
-0.8805	1.3257		
2.1039	0.5207		
-0.6075	1.0386		
-0.8531	1.1688		
1.1159	0.2305		
0.0000	0.6597		
1.0000	0.0000	0.0000	0.0000
0.0000	1.0000	0.0000	0.0000

```
0.0000 0.0000 1.0000 0.0000
0.0000 0.0000 0.0000 1.0000
0.0019
0.5075
0.4076
0.8408
0.5017
0.9128
0.2129
0.5591
```

### 8.3. Program Results

g13ecc Example Program Results

The inverse of the square root of the state covariance matrix is

```
0.6897 0.7721 0.7079 0.6102
0.0000 -0.3363 -0.2252 -0.2642
0.0000 0.0000 -0.1650 0.0319
0.0000 0.0000 0.0000 0.3708
```

The components of the estimated filtered state are

```
k      x(k)
0      -0.7125
1      -1.8324
2       1.7500
3       1.5854
```

---