# nag_kalman_sqrt_filt_info_invar (g13edc)

## 1. Purpose

**nag_kalman_sqrt_filt_info_invar (g13edc)** performs a combined measurement and time update of one iteration of the time-invariant Kalman filter. The method employed for this update is the square root information filter with the system matrices in condensed controller Hessenberg form.

## 2. Specification

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

void g13edc(Integer n, Integer m, Integer p, double t[], Integer tdt,
            double ainv[], Integer tda, double ainvb[], Integer tdai,
            double rinv[], Integer tdr, double c[], Integer tdc,
            double qinv[], Integer tdq, double x[], double rinvy[], double z[],
            double tol, NagError *fail)
```

## 3. Description

For the state space system defined by

$$\begin{aligned} X_{i+1} &= AX_i + BW_i \qquad \mathrm{var}(W_i) = Q_i \\ Y_i &= CX_i + V_i \qquad \mathrm{var}(V_i) = R_i \end{aligned}$$

the estimate of $X_i$ given observations $Y_1$ to $Y_{i-1}$ is denoted by $\hat{X}_{i|i-1}$ with $\mathrm{var}(\hat{X}_{i|i-1}) = P_{i|i-1} = S_i S_i^T$ (where $A$, $B$ and $C$ are time invariant).

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

$$U_1 \begin{pmatrix} Q_i^{-1/2} & 0 & Q_i^{-1/2}\bar{w} \\ 0 & R_i^{-1/2}C & R^{-1/2}Y_{i+1} \\ S_i^{-1}A^{-1}B & S_i^{-1}A^{-1} & S_i^{-1}X_{i|i} \end{pmatrix} = \begin{pmatrix} F_{i+1}^{-1/2} & * & * \\ 0 & S_{i+1}^{-1} & \xi_{i+1|i+1} \\ 0 & 0 & E_{i+1} \end{pmatrix}$$

(Pre-array) (Post-array)

where $U_1$ is an orthogonal transformation triangularizing the pre-array, and the matrix pair ($A^{-1}$,$A^{-1}B$) is in upper controller Hessenberg form. The triangularization is done entirely via Householder transformations exploiting the zero pattern of the pre-array.

An example of the pre-array is given below (where $n = 6$, $m = 2$ and $p = 3$):

$$\left(\begin{array}{cc|cccccc|c}
x & x & & & & & & & x \\
 & x & & & & & & & x \\
\hline
 & & x & x & x & x & x & x & x \\
 & & x & x & x & x & x & x & x \\
 & & x & x & x & x & x & x & x \\
\hline
x & x & x & x & x & x & x & x & x \\
 & x & x & x & x & x & x & x & x \\
 & & x & x & x & x & x & x & x \\
 & & & x & x & x & x & x & x \\
 & & & & x & x & x & x & x \\
 & & & & & x & x & x & x
\end{array}\right)$$

The term $\bar{w}$ 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

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

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

The new state filtered state estimate is computed via

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

The function returns $S_{i+1}^{-1}$ and, optionally, $\hat{X}_{i+1|i+1}$ (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 $S_i^{-1}$ and $A^{-1}$.

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 $R_i^{-1/2}$.

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

**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: $\mathbf{tdt} \geq \mathbf{n}$.

**ainv[n][tda]**

Input: The leading $n$ by $n$ part of this array must contain the upper controller Hessenberg matrix $UA^{-1}U^T$. Where $A^{-1}$ is the inverse of the state transition matrix, and $U$ is the unitary matrix generated by the function nag_trans_hessenberg_controller (g13exc).

**tda**

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

Constraint: $\mathbf{tda} \geq \mathbf{n}$.

**ainvb[n][tdai]**

Input: The leading $n$ by $m$ part of this array must contain the upper controller Hessenberg matrix $UA^{-1}B$. Where $A^{-1}$ is the inverse of the transition matrix, $B$ is the input weight matrix $B$, and $U$ is the unitary transformation generated by the function nag_trans_hessenberg_controller (g13exc).

**tdai**

The trailing dimension of array **ainvb** as declared in the calling program.

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

**rinv[p][tdr]**

Input: If the 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 $R^{-1/2}$ 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 **c** 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: If the array argument **rinv** (above) has been defined then the leading $p$ by $n$ part of this array must contain the matrix $CU^T$, otherwise (if **rinv** is the null pointer (double $*$)0) then the leading $p$ by $n$ part of the array must contain the matrix $R_i^{-1/2}CU^T$. $C$ is the output weight matrix, $R_i$ is the noise covariance matrix and $U$ is the same unitary transformation used for defining array arguments **ainv** and **ainvb**.

**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 **qinv** 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: This array contains the estimated state $\hat{X}_{i+1|i+1}$.

**rinvy[p]**

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

**z[m]**

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

**tol**

Input: **tol** is used to test for near singularity of the matrix $S_{i+1}$. 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_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** ≥ **n**.

On entry **tda** = ⟨*value*⟩ while **n** = ⟨*value*⟩.
These parameters must satisfy **tda** ≥ **n**.

On entry **tdai** = ⟨*value*⟩ while **m** = ⟨*value*⟩.
These parameters must satisfy **tdai** ≥ **m**.

On entry **tdc** = ⟨*value*⟩ while **n** = ⟨*value*⟩.
These parameters must satisfy **tdc** ≥ **n**.

On entry **tdq** = ⟨*value*⟩ while **m** = ⟨*value*⟩.
These parameters must satisfy **tdq** ≥ **m**.

On entry **tdr** = ⟨*value*⟩ while **p** = ⟨*value*⟩.
These parameters must satisfy **tdr** ≥ **p**.

**NE_MAT_SINGULAR**

The matrix inverse(S) is singular.

**NE_ALLOC_FAIL**

Memory allocation failed.

## 6. Further Comments

The algorithm requires approximately $\frac{1}{6}n^3 + n^2(\frac{3}{2}m + p) + 2nm^2 + \frac{2}{3}m^3$ 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.

Van Dooren P and Verhaegen M H G (1988) Condensed Forms for Efficient Time-Invariant Kalman Filtering *SIAM J. Sci. Stat. Comput.* **9** 516–530.

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_var (g13ecc)
nag_trans_hessenberg_controller (g13exc)

## 8. Example 1

To apply three iterations of the Kalman filter (in square root information form) to the time-invariant system $(A^{-1},\ A^{-1}B,\ C)$ supplied in upper controller Hessenberg form.

### 8.1. Program Text

```
/* nag_kalman_sqrt_filt_info_invar(g13edc)  Example Program
 *
 * Copyright 1994 Numerical Algorithms Group
 *
 * Mark 3, 1994.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagf06.h>
#include <nagf03.h>
#include <nagg13.h>

typedef enum {read, print} ioflag;

#ifdef NAG_PROTO
static void ex1(void);
static void ex2(void);
#else
static void ex1();
static void ex2();
#endif

main()
{
  ex1();
  ex2();
  exit(EXIT_SUCCESS);
}

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

#ifdef NAG_PROTO
static void ex1(void)
#else
     static void ex1()
#endif
{
  double ainv[NMAX][TRADIM],qinv[MMAX][TRADIM],rinv[PMAX][TRADIM],
  t[NMAX][TRADIM],ainvb[NMAX][TRADIM];
  double c[PMAX][TRADIM],x[NMAX],z[MMAX],rinvy[PMAX];
  Integer i,j,m,n,p,istep;
  double tol;
  Integer nmax, mmax, pmax, tradim;

  Vprintf("g13edc Example 1 Program Results\n");

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

  nmax = NMAX;
  mmax = MMAX;
  pmax = PMAX;
  tradim = TRADIM;
```

```
  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);
    }

  /* 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", &ainvb[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  */

  for (istep=1; istep<=3; ++istep)
    g13edc(n, m, p, (double *)t, tradim, (double *)ainv,
           tradim, (double *)ainvb, tradim, (double *)rinv,
           tradim, (double *)c, tradim, (double *)qinv,
           tradim, 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]);
    }
}

#ifdef NAG_PROTO
static void mat_io(Integer n, Integer m, double mat[], Integer tdmat,
                   ioflag flag, char *message);
#else
static void mat_io();
#endif
```

### 8.2. Program Data

```
 g13edc Example 1 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
   0.0000  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
   0.0000  0.5207
   0.0000  0.0000
   0.0000  0.0000
   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

```
g13edc Example 1 Program Results

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

 -0.8731  -1.1461  -1.0260  -0.8901
  0.0000  -0.2763  -0.1929  -0.3763
  0.0000   0.0000  -0.1110  -0.1051
  0.0000   0.0000   0.0000   0.3120

The components of the estimated filtered state are

   k       x(k)
   0     -2.0688
   1     -0.7814
   2      2.2181
   3      0.9298
```

## 9. Example 2

To apply three iterations of the Kalman filter (in square root information form) to the general time-invariant system $(A^{-1},\ A^{-1}B,\ C)$. The use of the time-varying Kalman function nag_kalman_sqrt_filt_info_var (g13ecc) is compared with that of the time-invariant function nag_kalman_sqrt_filt_info_invar. The same original data is used by both functions but additional transformations are required before it can be supplied to nag_kalman_sqrt_filt_info_invar. It can be seen that (after the appropriate back-transformations on the output of nag_kalman_sqrt_filt_info_invar) the results of both nag_kalman_sqrt_filt_info_var (g13ecc) and nag_kalman_sqrt_filt_info_invar are in agreeement.

### 9.1. Program Text

```
#ifdef NAG_PROTO
static void ex2(void)
#else
     static void ex2()
#endif
{

  double ainv[NMAX][TRADIM], ainvb[NMAX][TRADIM], c[PMAX][TRADIM],
  ainvu[NMAX][TRADIM], ainvbu[NMAX][TRADIM];
  double qinv[MMAX][TRADIM], rinv[PMAX][TRADIM], t[NMAX][TRADIM], x[NMAX],
  z[NMAX];
  double rwork[NMAX][TRADIM], tu[NMAX][TRADIM], rinvy[PMAX], ig[NMAX][TRADIM],
  ih[NMAX][TRADIM];
  double cu[PMAX][TRADIM], u[PMAX][TRADIM], ux[PMAX];
  double diag[NMAX];
  double detf, tol, zero = 0.0, one = 1.0;
  Integer dete, i, j, m, n, p, istep, ione = 1;
  Nag_ab_input inp_ab = Nag_ab_prod;
  Nag_ControllerForm reduceto  = Nag_UH_Controller;
  Integer nmax, mmax, pmax, tradim;

  Vprintf("\n\ng13edc Example 2 Program Results\n");
  /* skip the heading in the data file */
  Vscanf(" %*[^\n]");

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

  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);
    }

  /* Read data */
  mat_io(n, n, (double *)ainv, tradim, read, "");
  mat_io(p, n, (double *)c, tradim, read, "");
  if (rinv)
    mat_io(p, p, (double *)rinv, tradim, read,"");
  mat_io(n, m, (double *)ainvb, tradim, read, "");
  mat_io(m, m, (double *)qinv, tradim, read, "");
  mat_io(n, n, (double *)t, tradim, read, "");
  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]);

  for (i=0; i<n; ++i)     /*  Initialise the identity matrix u */
    {
      for (j=0; j<n; ++j)
        u[i][j] = zero;
      u[i][i] = one;
    }

  /*  Copy the arrays ainv[] and ainvb[] into ainvu[] and ainvbu[] */
  for (i=0; i< n; ++i)
    f06efc(n, &ainv[0][i], tradim, &ainvu[0][i], tradim);
  for (j=0; j<m; ++j)
    f06efc(n, &ainvb[0][j], tradim, &ainvbu[0][j], tradim);
```

```
/*  Transform (ainvu[],ainvbu[]) to reduceto controller Hessenberg form */
 g13exc(n, m, reduceto, (double *)ainvu, tradim, (double *)ainvbu, tradim,
        (double *)u, tradim, NAGERR_DEFAULT);

 /*  Calculate the matrix cu = c*u'    */
 f06yac(NoTranspose, Transpose, p, n, n, one, (double *)c, tradim,
        (double *)u, tradim, zero, (double *)cu, tradim);

 /*  Calculate the vector ux = u*x     */
 f06pac(NoTranspose, n, n, one, (double *)u, tradim, (double *)x, ione,
        zero, (double *)ux, ione);

 /* Form the information matrices ih = u*ig*u' and ig = t'*t     */
 f06yac(Transpose, NoTranspose, n, n, n, one, (double *)t, tradim,
        (double *)t, tradim, zero, (double *)ig, tradim);
 f06yac(NoTranspose, Transpose, n, n, n, one, (double *)ig, tradim,
        (double *)u, tradim, zero, (double *)rwork, tradim);
 f06yac(NoTranspose, NoTranspose, n, n, n, one, (double *)u, tradim,
        (double *)rwork, tradim, zero, (double *)ih, tradim);

 /* Now find the reduceto triangular (right) cholesky factor of ih */
 f03aec(n, (double *)ih, tradim, diag, &detf, &dete, NAGERR_DEFAULT);
 for (i=0; i< n; ++i)
   {
     tu[i][i] = one/diag[i];
     for (j=0; j<i; ++j)
       {
         tu[j][i] = ih[i][j];
         tu[i][j] = zero;
       }
   }

 /* Do three iterations of the Kalman filter recursion  */
 for (istep=1; istep<=3; ++istep)
   {
     g13ecc(n, m, p, inp_ab, (double *)t, tradim, (double *)ainv,
            tradim, (double *)ainvb, tradim, (double *)rinv, tradim,
            (double *)c, tradim, (double *)qinv, tradim, (double *)x,
            (double *)rinvy, (double *)z, tol,NAGERR_DEFAULT);
     g13edc(n, m, p, (double *)tu, tradim, (double *)ainvu, tradim,
            (double *)ainvbu, tradim, (double *)rinv, tradim, (double *)cu,
            tradim, (double *)qinv, tradim, (double *)ux, (double *)rinvy,
            (double *)z, tol, NAGERR_DEFAULT);
   }

 /* Print Results */
 Vprintf("Results from g13ecc \n\n");     /* let ig = t' * t */
 f06yac(Transpose, NoTranspose, n, n, n, one, (double *)t, tradim,
        (double *)t, tradim, zero, (double *)ig, tradim);
 mat_io(n, n, (double *)ig, tradim, print,"The information matrix ig is\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   %8.4f \n", i, x[i]);
 Vprintf("\nResults from g13edc \n\n");    /* let ih = tu' * tu */
 f06yac(Transpose, NoTranspose, n, n, n, one, (double *)tu, tradim,
        (double *)tu, tradim, zero, (double *)ih, tradim);
 mat_io(n, n, (double *)ih, tradim, print,"The information matrix ih is\n");

 /*  Calculate ih = u'*ih*u   */
 f06yac(NoTranspose, NoTranspose, n, n, n, one, (double *)ih, tradim,
        (double *)u, tradim, zero, (double *)rwork, tradim);
 f06yac(Transpose, NoTranspose, n, n, n, one, (double *)u, tradim,
        (double *)rwork, tradim, zero, (double *)ih, tradim);
 mat_io(n, n, (double *)ih, tradim, print,"\nThe matrix u' * ih * u is\n");

 /*  Calculate x = u' * ux */
 f06pac(Transpose, n, n, one, (double *)u, tradim, (double *)ux, ione,
        zero, (double *)x, ione);
 Vprintf("\nThe components of the estimated filtered state are \n\n");
```

```
  Vprintf("  k       x(k)  \n");
  for (i=0; i<n; ++i)
    Vprintf("  %ld   %8.4f \n", i, x[i]);
}

#ifdef NAG_PROTO
static void mat_io(Integer n, Integer m, double mat[], Integer tdmat,
                   ioflag flag, char * message)
#else
     static void mat_io(n, m, mat, tdmat, flag, message)
     Integer n, m, tdmat;
     double mat[];
     ioflag flag;
     char * message;
#endif
{  Integer i, j;
#define MAT(I,J) mat[((I)-1) * tdmat + (J) - 1]
   if (flag==print) Vprintf("%s \n", message);
   for (i=1; i<=n; ++i)
     {
       for (j=1; j<=m; ++j)
         {
           if (flag==read) Vscanf("%lf", &MAT(i,j));
           if (flag==print) Vprintf("%8.4f ", MAT(i,j));
         }
       if (flag==print) Vprintf("\n");
     }
 } /* mat_io */
```

## 9.2. Program Data

```
g12edc Example 2 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  2.1000  0.1400   0.0000
   0.0000  0.6010  2.8000  -1.3400
   0.0000  0.0000  1.3000  -0.8000
   0.0000  0.0000  0.0000   1.4100
   0.0019
   0.5075
   0.4076
   0.8408
   0.5017
   0.9128
   0.2129
   0.5591
```

## 9.3. Program Results

```
g13edc Example 2 Program Results

Results from g13ecc

The information matrix ig is

  0.4661   0.5290   0.4826   0.4134
  0.5290   0.7196   0.6158   0.5657
  0.4826   0.6158   0.5781   0.4776
  0.4134   0.5657   0.4776   0.5825

The components of the estimated filtered state are

  k       x(k)
  0    -0.8369
  1    -1.4649
  2     1.4877
  3     1.5276

Results from g13edc

The information matrix ih is

  0.0399  -0.0805  -0.0137  -0.0174
 -0.0805   2.1143   0.2453   0.0406
 -0.0137   0.2453   0.0770  -0.0294
 -0.0174   0.0406  -0.0294   0.1151

The matrix u' * ih * u is

  0.4661   0.5290   0.4826   0.4134
  0.5290   0.7196   0.6158   0.5657
  0.4826   0.6158   0.5781   0.4776
  0.4134   0.5657   0.4776   0.5825

The components of the estimated filtered state are

  k       x(k)
  0    -0.8369
  1    -1.4649
  2     1.4877
  3     1.5276
```