nag_kalman_sqrt_filt_cov_invar (g13ebc)

1. Purpose

nag_kalman_sqrt_filt_cov_invar (g13ebc) 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 covariance filter with the system matrices transformed into condensed observer Hessenberg form.

2. Specification

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

void g13ebc(Integer n, Integer m, Integer p, double s[], Integer tds,
             double a[], Integer tda, double b[], Integer tdb, double q[],
             Integer tdq, double c[], Integer tdc, double r[], Integer tdr,
             double k[], Integer tdk, double h[], Integer tdh, double tol,
             NagError *fail)

3. Description

For the state space system defined by

\[ X_{i+1} = AX_i + BW_i \quad \text{var}(W_i) = Q_i \]
\[ Y_i = CX_i + V_i \quad \text{var}(V_i) = R_i \]

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

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

\[
\begin{pmatrix}
R_i^{1/2} & 0 & CS_i \\
0 & BQ_i^{1/2} & AS_i
\end{pmatrix}
U_1 =
\begin{pmatrix}
H_i^{1/2} & 0 & 0 \\
G_i & S_{i+1} & 0
\end{pmatrix}
\]

(Pre-array) \quad \text{(Post-array)}

where \( U_1 \) is an orthogonal transformation triangularizing the pre-array, and the matrix pair \((A,C)\) is in lower observer Hessenberg form. The triangularization is carried out via Householder transformations exploiting the zero pattern of the pre-array.

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

\[
\begin{pmatrix}
  x & x & x & x & x & x \\
  x & x & x & x & x & x \\
  x & x & x & x & x & x \\
  x & x & x & x & x & x \\
  x & x & x & x & x & x \\
  x & x & x & x & x & x \\
  x & x & x & x & x & x \\
\end{pmatrix}
\]

The measurement-update for the estimated state vector \( X \) is

\[ \hat{X}_{i|i} = \hat{X}_{i|i-1} - K_i [C \hat{X}_{i|i-1} - Y_i] \]
whilst the time-update for $X$ is
\[
\hat{X}_{i+1|i} = A\hat{X}_{i|i} + D_i U_i
\]
where $D_i U_i$ represents any deterministic control used. The relationship between the Kalman gain matrix $K_i$ and $G_i$ is
\[
AK_i = G_i \left( H_i^{1/2} \right)
\]
The function returns the product of the matrices $A$ and $K_i$, represented as $AK_i$, and the state covariance matrix $P_{i|i-1}$ factorised as $P_{i|i-1} = S_i S_i^T$ (see the Introduction to Chapter g13 for more information concerning the covariance filter).

4. Parameters

$n$
Input: The actual state dimension, $n$, i.e., the order of the matrices $S_i$ and $A$.
Constraint: $n \geq 1$.

$m$
Input: The actual input dimension, $m$, i.e., the order of the matrix $Q_i^{1/2}$.
Constraint: $m \geq 1$.

$p$
Input: The actual output dimension, $p$, i.e., the order of the matrix $R_i^{1/2}$.
Constraint: $p \geq 1$.

$s[n][tds]$
Input: The leading $n$ by $n$ lower triangular part of this array must contain $S_i$, the left Cholesky factor of the state covariance matrix $P_{i|i-1}$.
Output: The leading $n$ by $n$ lower triangular part of this array contains $S_{i+1}$, the left Cholesky factor of the state covariance matrix $P_{i+1|i}$.

$tds$
Input: The trailing dimension of array $s$ as declared in the calling program.
Constraint: $tds \geq n$.

$a[n][tda]$
Input: The leading $n$ by $n$ part of this array must contain the lower observer Hessenberg matrix $UAU^T$. Where $A$ is the state transition matrix of the discrete system and $U$ is the unitary transformation generated by the function nag_trans_hessenberg_observer (g13ewc).

$tda$
Input: The trailing dimension of array $a$ as declared in the calling program.
Constraint: $tda \geq n$.

$b[n][tdb]$
Input: If the array argument $q$ (below) has been defined then the leading $n$ by $m$ part of this array must contain the matrix $UB$, otherwise (if $q$ is the null pointer (double *)0) then the leading $n$ by $m$ part of the array must contain the matrix $UBQ_i^{1/2}$. $B$ is the input weight matrix, $Q_i$ is the noise covariance matrix and $U$ is the same unitary transformation used for defining array arguments $a$ and $c$. 
g13 – Time Series Analysis

**tdb**

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

Constraint: \( tdb \geq m \).

**q[m][tdq]**

Input: If the noise covariance matrix is to be supplied separately from the input weight matrix then the leading \( m \) by \( m \) lower triangular part of this array must contain \( Q_i^{1/2} \), the left Cholesky factor process noise covariance matrix. If the noise covariance matrix is to be input with the weight matrix as \( BQ_i^{1/2} \) then the array \( q \) must be set to the null pointer, i.e., (double *)0.

**tdq**

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

Constraint: \( tdq \geq m \) if q is defined.

**c[p][tdc]**

Input: The leading \( p \) by \( n \) part of this array must contain the lower observer Hessenberg matrix \( CU^T \). Where \( C \) is the output weight matrix of the discrete system and \( U \) is the unitary transformation matrix generated by the function nag_trans_hessenberg_observer (g13ewc).

**tdc**

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

Constraint: \( tdc \geq n \).

**r[p][tdr]**

Input: The leading \( p \) by \( p \) lower triangular part of this array must contain \( R_i^{1/2} \), the left Cholesky factor of the measurement noise covariance matrix.

**tdr**

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

Constraint: \( tdr \geq p \).

**k[n][tdk]**

Output: If \( k \) is defined, then the leading \( n \) by \( p \) part of this array contains the \( AK_i \), the product of the Kalman filter gain matrix \( K_i \) with the state transition matrix \( A \). If this is not required then the array \( k \) must be set to the null pointer, i.e., (double *)0.

**tdk**

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

Constraint: \( tdk \geq p \) if \( k \) is defined.

**h[p][tdh]**

Output: If \( h \) is defined, then the leading \( p \) by \( p \) lower triangular part of this array contains \( H_i^{1/2} \). If \( h \) has not been defined then array \( h \) is not referenced and may be set to the null pointer i.e., (double *)0.

**tdh**

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

Constraint: \( tdh \geq p \) if \( k \) and \( h \) are defined.
tol
Input: If k is defined, then tol is used to test for near singularity of the matrix $H_1^{1/2}$. If the user sets tol to be less than $p^2\epsilon$ then the tolerance is taken as $p^2\epsilon$, where $\epsilon$ is the machine precision. Otherwise, tol need not be set by the user.

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 tds = ⟨value⟩ while n = ⟨value⟩.
These parameters must satisfy tds ≥ n.
On entry tda = ⟨value⟩ while n = ⟨value⟩.
These parameters must satisfy tda ≥ n.
On entry tdb = ⟨value⟩ while m = ⟨value⟩.
These parameters must satisfy tdb ≥ n.
On entry tdc = ⟨value⟩ while n = ⟨value⟩.
These parameters must satisfy tdc ≥ n.
On entry tdr = ⟨value⟩ while p = ⟨value⟩.
These parameters must satisfy tdr ≥ p.
On entry tdq = ⟨value⟩ while m = ⟨value⟩.
These parameters must satisfy tdq ≥ m.
On entry tdk = ⟨value⟩ while p = ⟨value⟩.
These parameters must satisfy tdk ≥ p.
On entry tdh = ⟨value⟩ while p = ⟨value⟩.
These parameters must satisfy tdh ≥ p.

NE_MAT_SINGULAR
The matrix sqrt(H) is singular.

NE_NULL_ARRAY
Array h has null address.

NE_ALLOC_FAIL
Memory allocation failed.

6. Further Comments

The algorithm requires $\frac{1}{6}n^3 + n^2(\frac{3}{2}p + m) + 2np^2 + \frac{3}{2}p^3$ operations and is backward stable (see Verhaegen et al).

6.1. Accuracy

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

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

7. See Also

nag_kalman_sqrt_filt_cov_var (g13ec)

nag_trans_hessenberg_observer (g13ew)

8. Example 1

To apply three iterations of the Kalman filter (in square root covariance form) to the time-invariant
system \((A, B, C)\) supplied in lower observer Hessenberg form.

8.1. Program Text

/* nag_kalman_sqrt_filt_cov_invar(g13ebc) 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 <nagf13.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 10
#define MMAX 10
#define PMAX 10
#define TRADIM 10

#ifdef NAG_PROTO
static void ex1(void)
#else
    static void ex1()
#endif
double a[NMAX][TRADIM], b[NMAX][TRADIM], c[PMAX][TRADIM], k[NMAX][TRADIM],
q[MMAX][TRADIM], r[PMAX][TRADIM], s[NMAX][TRADIM], h[NMAX][TRADIM];
Integer i, j, m, n, p, istep;
double tol;
Integer nmax, mmax, pmax, tradim;
Vprintf("g13ebc 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 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", &s[i][j]);
for (i=0; i<n; ++i)
  for (j=0; j<m; ++j)
    Vscanf("%lf", &a[i][j]);
for (i=0; i<p; ++i)
  for (j=0; j<n; ++j)
    Vscanf("%lf", &c[i][j]);
for (i=0; i<p; ++i)
  for (j=0; j<p; ++j)
    Vscanf("%lf", &r[i][j]);
/* Perform three iterations of the Kalman filter recursion */
for (istep=1; istep<=3; ++istep)
g13ebc(n, m, p, (double *)s, tradim, (double *)a,
  tradim, (double *)b, tradim, (double *)q, tradim,
  (double *)c, tradim, (double *)r, tradim,
  (double *)k, tradim, (double *)h, tradim, tol, NAGERR_DEFAULT);
Vprintf("\nThe square root of the state covariance matrix is\n");
for (i=0; i<n; ++i)
  {
    for (j=0; j<n; ++j)
      Vprintf("%8.4f ", s[i][j]);
    Vprintf("\n");
  }
if (k)
  {
    Vprintf("\nThe matrix AK (the product of the Kalman gain) is\n");
    Vprintf("\n_matrix with the state transition matrix) is\n");
    for (i=0; i<n; ++i)
{  
  for (j=0; j<p; ++j)  
    Vprintf("%8.4f ", k[i][j]);  
    Vprintf("n");  
}  

#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  
g13ebc Example 1 Program Data  
4 2 2 0.0  
0.0000 0.0000 0.0000 0.0000  
0.0000 0.0000 0.0000 0.0000  
0.0000 0.0000 0.0000 0.0000  
0.0000 0.0000 0.0000 0.0000  
0.2113 0.8497 0.7263 0.0000  
0.7560 0.6857 0.1985 0.6525  
0.0002 0.8782 0.5442 0.3076  
0.3303 0.0863 0.2320 0.9329  
0.5618 0.5042  
0.5896 0.3453  
0.6853 0.3873  
0.8906 0.9222  
1.0000 0.0000  
0.0000 1.0000  
0.3616 0.0000 0.0000 0.0000  
0.2922 0.4826 0.0000 0.0000  
0.9488 0.0000  
0.3760 0.7340  

8.3. Program Results  
g13ebc Example 1 Program Results  
The square root of the state covariance matrix is  
-1.7223  0.0000  0.0000  0.0000  
-2.1073  0.5467  0.0000  0.0000  
-1.7649  0.1412 -0.1710  0.0000  
-1.8291  0.2058 -0.1497  0.7760  
The matrix AK (the product of the Kalman gain  
matrix with the state transition matrix) is  
-0.2135  1.6649  
-0.2345  2.1442  
-0.2147  1.7069  
-0.1345  1.4777  

9. Example 2  
To apply three iterations of the Kalman filter (in square root covariance form) to the  
general time-invariant system \((A, B, C)\). The use of the time-varying Kalman function  
nag_kalman_sqrt_filt_cov_var (g13eac) is compared with that of the time-invariant function  
nag_kalman_sqrt_filt_cov_invar (g13ebc). The same original data is used by both functions but  
additional transformations are required before it can be supplied to nag_kalman_sqrt_filt_cov_invar  
(g13ebc). It can be seen that (after the appropriate back-transformations on the output of  
nag_kalman_sqrt_filt_cov_invar (g13ebc)) the results of both nag_kalman_sqrt_filt_cov_var (g13eac)  
and nag_kalman_sqrt_filt_cov_invar (g13ebc) are the same.
9.1. Program Text

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

#ifdef NAG_PROTO
static void ex2(void)
#else
static void ex2()
#endif
{
    /* more general example which requires the data to be transformed. The
     results produced by g13eac and g13ebc are compared */

double a[NMAX][TRADIM], b[NMAX][TRADIM], c[PMAX][TRADIM],
    ke[NMAX][TRADIM], kf[NMAX][TRADIM], ub[NMAX][TRADIM], q[MMAX][TRADIM],
    r[PMAX][TRADIM], rwork[NMAX][TRADIM], sf[NMAX][TRADIM], se[NMAX][TRADIM],
    h[NMAX][TRADIM];

    double pf[NMAX][TRADIM], pe[NMAX][TRADIM], uaut[NMAX][TRADIM],
    cut[PMAX][TRADIM], u[NMAX][TRADIM];

double diag[NMAX];

    Integer i, j, m, n, p, istep;
    Integer nmax, mmax, pmax, tradim;

    Nag_ObserverForm reduceto = Nag_LH_Observer;
    double detf, tol, zero = 0.0, one = 1.0;
    Integer dete, ione = 1;

    Vprintf("g13ebc Example 2 Program Results \
    
    ");
    /* 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")
        n = %ld, m = %ld, p = %ld\n", n, m, p);
        exit(EXIT_FAILURE);
    }

    mat_io(n, n, (double *)se, tradim, read, "");
    mat_io(n, n, (double *)a, tradim, read, "");
    mat_io(n, m, (double *)b, tradim, read, "");
    if (q)
        mat_io(m, m, (double *)q, tradim, read,"");
    mat_io(p, n, (double *)c, tradim, read, "");
    mat_io(p, p, (double *)r, tradim, read,");
    for (i=0; i<n; ++i)
    {
        for (j=0; j<n; ++j)
        {
            if (i<p)
                cut[i][j] = c[i][j];
            sf[i][j] = se[i][j];
            uaut[i][j] = a[i][j];
            u[i][j] = zero;
        }
        u[i][i] = one;
    }

    /* Set up the matrix pair (A,C) in the lower observer hessenberg form */
    g13ewc(n, p, reduceto, (double *)uaut, tradim, (double *)cut, tradim,
        (double *)u, tradim, NAGERR_DEFAULT);
    for (j=0; j<m; ++j)
    for (i=0; i<n; ++i)
        ub[i][j] = f06eacn(&u[i][0], ione, &b[0][j], tradim);

    /* Generate noise covariance matrices PE and PF = U * PE * U' */
    f06yac(NoTranspose, Transpose, n, n, n, one, (double *)se, tradim,
        (double *)se, tradim, zero, (double *)pe, tradim);
}
```

3.g13ebc.8
/* Now find the lower triangular (left) cholesky factor of PF. */
for (i=0; i<n; ++i)
{
    sf[i][i] = one/diag[i];
    for (j=0; j<i; ++j)
        sf[i][j] = pf[i][j];
}
/* Perform three steps of the Kalman filter recursion */
for (istep=1; istep<=3; ++istep)
{
    g13eac(n, m, p, (double *)se, tradim, (double *)a,
    tradim, (double *)b, tradim, (double *)q, 
    tradim, (double *)c, tradim, (double *)r,
    tradim, (double *)h, tradim, tol, NAGERR_DEFAULT);
    g13ebc(n, m, p, (double *)sf, tradim, (double *)uaut,
    tradim, (double *)ub, tradim, (double *)q,
    tradim, (double *)cut,tradim, (double *)r,
    tradim, (double *)h, tradim, tol, NAGERR_DEFAULT);
}
/* Calculate PF = U' * PF * U */
f06yac(NoTranspose, Transpose, n, n, n, one, (double *)pe, tradim,
    (double *)u, tradim, zero, (double *)rwork, tradim);
f06yac(NoTranspose, NoTranspose, n, n, n, one, (double *)sf, tradim,
    (double *)pf, tradim, zero, (double *)rwork, tradim);
mat_io(n,n,(double*)pe,tradim,print,"Covariance matrix PE from g13eac is\n");
mat_io(n,n,(double*)sf,tradim,print,"Covariance matrix PF from g13ebc is\n");
/* calculate U' * K */
f06yac(Transpose, NoTranspose, n, p, n, one, (double *)u, tradim,
    (double *)kf, tradim, zero, (double *)rwork, tradim);
mat_io(n,p,(double*)u,tradim,print,"U' * KF is\n");
}
#endif

static void mat_io(Integer n, Integer m, double mat[], Integer tdmat,
ioflag flag, char *message)
{
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==print) Vprintf("%8.4f ", MAT(i,j));
        
        if (flag==read) Vscanf("%lf", &MAT(i,j));
    }
}}
9.2. Program Data

**g13ebc Example 2 Program Data**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>0.0</td>
</tr>
<tr>
<td>1.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.8400</td>
<td>0.9010</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.3000</td>
<td>0.7001</td>
<td>0.8300</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.5000</td>
<td>0.2300</td>
<td>0.1100</td>
<td>0.4303</td>
</tr>
<tr>
<td>0.2113</td>
<td>0.8497</td>
<td>0.7263</td>
<td>0.8833</td>
</tr>
<tr>
<td>0.7560</td>
<td>0.6857</td>
<td>0.1985</td>
<td>0.6525</td>
</tr>
<tr>
<td>0.0002</td>
<td>0.8782</td>
<td>0.5442</td>
<td>0.3076</td>
</tr>
<tr>
<td>0.3303</td>
<td>0.0683</td>
<td>0.2320</td>
<td>0.9329</td>
</tr>
<tr>
<td>0.5618</td>
<td>0.5042</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5896</td>
<td>0.3493</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6853</td>
<td>0.3873</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8906</td>
<td>0.9222</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0000</td>
<td>0.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0000</td>
<td>1.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3616</td>
<td>0.5664</td>
<td>0.5015</td>
<td>0.2693</td>
</tr>
<tr>
<td>0.2922</td>
<td>0.4826</td>
<td>0.4368</td>
<td>0.6325</td>
</tr>
<tr>
<td>0.9488</td>
<td>0.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3760</td>
<td>0.7340</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9.3. Program Results

**g13ebc Example 2 Program Results**

Covariance matrix PE from g13eac is

```
1.6761 1.4744 1.2519 1.6852
1.4744 1.3646 1.1367 1.4651
1.2519 1.1367 1.0668 1.3445
1.6852 1.4651 1.3445 2.2045
```

Covariance matrix PF from g13ebc is

```
5.0635 -1.5512 0.0231 1.1756
-1.5512 0.8503 -0.0492 -0.3631
0.0231 -0.0492 0.0648 -0.0217
1.1756 -0.3631 -0.0217 0.3336
```

Matrix \( U' \ast PF \ast U \) is

```
1.6761 1.4744 1.2519 1.6852
1.4744 1.3646 1.1367 1.4651
1.2519 1.1367 1.0668 1.3445
1.6852 1.4651 1.3445 2.2045
```

The matrix KE from g13eac is

```
0.3699 0.9447
0.3526 0.8199
0.2783 0.5375
0.1588 0.6704
```
The matrix KF from g13ebc is

\[
\begin{pmatrix}
-0.5857 & -1.4263 \\
-0.0280 & 0.2239 \\
0.0170 & 0.1200 \\
-0.1405 & -0.4519 \\
\end{pmatrix}
\]

\[
U' \cdot KF is
\begin{pmatrix}
0.3699 & 0.9447 \\
0.3526 & 0.8199 \\
0.2783 & 0.5375 \\
0.1588 & 0.6704 \\
\end{pmatrix}
\]