

4. Parameters

n

Input: The actual state dimension, n , i.e., the order of the matrices S_i and A_i .

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

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: $\mathbf{tds} \geq \mathbf{n}$.

a[n][tda]

Input: The leading n by n part of this array must contain A_i , the state transition matrix of the discrete system.

tda

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

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

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 B_i , otherwise (if **q** is the null pointer (double *)0) then the leading n by m part of the array must contain the matrix $B_i Q_i^{1/2}$. B_i is the input weight matrix and Q_i is the noise covariance matrix.

tdb

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

Constraint: $\mathbf{tdb} \geq \mathbf{m}$.

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 of the input process noise covariance matrix. If the noise covariance matrix is to be input with the weight matrix as $B_i Q_i^{1/2}$ 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 C_i , the output weight matrix of the discrete system.

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_i . If this is not required then the array **k** is not referenced and 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 **k** is defined, then the leading p by p lower triangular part of this array contains $H_i^{1/2}$. If **k** 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 $H_i^{1/2}$. If the user sets **tol** to be less than $p^2\epsilon$ then the tolerance is taken as $p^2\epsilon$, where ϵ 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** = $\langle value \rangle$.
 On entry, **m** must not be less than 1: **m** = $\langle value \rangle$.
 On entry, **p** must not be less than 1: **p** = $\langle value \rangle$.

NE_2_INT_ARG_LT

On entry **tds** = $\langle value \rangle$ while **n** = $\langle value \rangle$.
 These parameters must satisfy **tds** $\geq$ **n**.
 On entry **tda** = $\langle value \rangle$ while **n** = $\langle value \rangle$.
 These parameters must satisfy **tda** $\geq$ **n**.
 On entry **tdb** = $\langle value \rangle$ while **m** = $\langle value \rangle$.
 These parameters must satisfy **tdb** $\geq$ **m**.
 On entry **tdc** = $\langle value \rangle$ while **n** = $\langle value \rangle$.
 These parameters must satisfy **tdc** $\geq$ **n**.
 On entry **tdr** = $\langle value \rangle$ while **p** = $\langle value \rangle$.
 These parameters must satisfy **tdr** $\geq$ **p**.
 On entry **tdq** = $\langle value \rangle$ while **m** = $\langle value \rangle$.
 These parameters must satisfy **tdq** $\geq$ **m**.
 On entry **tdk** = $\langle value \rangle$ while **p** = $\langle value \rangle$.
 These parameters must satisfy **tdk** $\geq$ **p**.
 On entry **tdh** = $\langle value \rangle$ while **p** = $\langle value \rangle$.
 These parameters must satisfy **tdh** $\geq$ **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 $\frac{7}{6}n^3 + n^2(\frac{5}{2}p + m) + n(\frac{1}{2}m^2 + p^2)$ 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.
 Harvey A C and Phillips G D A (1979) Maximum likelihood estimation of regression models with autoregressive — moving average disturbances *Biometrika* **66** 49–58.
 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.

Wei W W S (1990) *Time Series Analysis: Univariate and Multivariate Methods* Addison-Wesley.

7. See Also

nag_kalman_sqrt_filt_cov_invar (g13ebc)

8. Example 1

To apply three iterations of the Kalman filter (in square root covariance form) to the system (A_i, B_i, C_i) . The same data is used for all three iterative steps.

8.1. Program Text

```
/* nag_kalman_sqrt_filt_cov_var(g13eac) Example Program
 *
 * Copyright 1996 Numerical Algorithms Group
 *
 * Mark 4, 1996.
 */
#include <nag.h>
#include <math.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nage04.h>
#include <nagf06.h>
#include <nagg05.h>
#include <nagg13.h>
#include <nagx02.h>

#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 NY 2000

#ifdef NAG_PROTO
static void objfun(Integer n, double theta_phi[], double *objf,
                  double g[], Nag_Comm *comm)
#else
static void objfun(n, theta_phi, objf, g, comm)
Integer n;
double theta_phi[];
double *objf;
double g[];
Nag_Comm *comm;
#endif
/* Routine to evaluate objective function. */
{
    double ak[2][1], h[1][1];
    double tol = 0.0;
    double xp[2];
    double q[1][1];
```

```

double c[1][2];
double r[1][1];          /* There is no measurement noise */
double a[2][2];
double s[2][2];
double b[2][1];
double hs, v, ss = 0.0, logdet = 0.0;
Integer k, nsum = 0, ione = 1, itwo = 2;
Integer nsteps = NY, n1 = 2, m1 = 1, p1 = 1;
double phi, theta, temp1, temp2, k11;
double *y;

y = comm->user;

xp[0] = 0.0; /* The expectation of the mean of an
              * ARMA(1,1) is 0.0 */
xp[1] = 0.0;
q[0][0] = 1.0;

c[0][0] = 1.0;
c[0][1] = 0.0;

r[0][0] = 0.0; /* There is no measurement noise */
theta = theta_phi[0];
phi = theta_phi[1];

b[0][0] = 1.0;
b[1][0] = - theta;

a[0][0] = phi;
a[1][0] = 0.0;
a[0][1] = 1.0;
a[1][1] = 0.0;

/* set value for cholesky factor of state covariance matrix */
temp1 = 1.0 + (theta * theta) - (2.0 * theta * phi);
temp2 = 1.0 - (phi * phi);
k11 = temp1/temp2;
s[0][0] = sqrt(k11);
s[0][1] = 0.0;
s[1][0] = - theta /s[0][0];
s[1][1] = theta * sqrt(1.0 - (1.0/k11));

/* iterate kalman filter for number of observations */
for (k=1; k <= nsteps; ++k)
{
    g13eac(n1, m1, p1, (double *)s, itwo, (double *)a,
           itwo, (double *)b, ione, (double *)q, ione, (double *)c,
           itwo, (double *)r, ione, (double *)ak, ione, (double *)h, ione,
           tol, NAGERR_DEFAULT);

    v = y[k-1] - c[0][0]*xp[0];
    hs = h[0][0] * h[0][0];
    logdet = logdet + log(hs);
    ss = ss + (v * v/ hs);
    nsum = nsum + 1;

    xp[0] = a[0][0]* xp[0] + a[0][1] * xp[1] + ak[0][0] * v;
    xp[1] = ak[1][0] * v;
}
*objf = nsum * log (ss/nsum) + logdet;
}
/* objfun */

#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 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;
    Integer m, n, p;
    Integer istep;
    double tol;
    Integer nmax, mmax, pmax, tradim;

    Vprintf("g13eac Example Program Results\n\n");
    Vprintf("Example 1\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",&s[i][j]);
    for (i=0; i<n; ++i)
        for (j=0; j<n; ++j)
            Vscanf("%lf",&a[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", &q[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)
        g13eac(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\n");
    for (i=0; i<n; ++i)
    {
        for (j=0; j<n; ++j)
            Vprintf("%8.4f ", s[i][j]);
        Vprintf("\n");
    }
    Vprintf("\nThe matrix AK (the product of the Kalman gain\n");
    Vprintf("matrix with the state transition matrix) is\n\n");
    for (i=0; i<n; ++i)
    {
        for (j=0; j<p; ++j)

```

```

        Vprintf("%.4f ", k[i][j]);
        Vprintf("\n");
    }
}

```

8.2. Program Data

```

g13eac 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 0.8833
0.7560 0.6857 0.1985 0.6525
0.0002 0.8782 0.5442 0.3076
0.3303 0.0683 0.2320 0.9329
0.5618 0.5042
0.5896 0.3493
0.6853 0.3873
0.8906 0.9222
1.0000 0.0000
0.0000 1.0000
0.3616 0.5664 0.5015 0.2693
0.2922 0.4826 0.4368 0.6325
0.9488 0.0000
0.3760 0.7340

```

8.3. Program Results

```
g13eac Example Program Results
```

```
Example 1
```

The square root of the state covariance matrix is

```

-1.2936 0.0000 0.0000 0.0000
-1.1382 -0.2579 0.0000 0.0000
-0.9622 -0.1529 0.2974 0.0000
-1.3076 0.0936 0.4508 -0.4897

```

The matrix AK (the product of the Kalman gain matrix with the state transition matrix) is

```

0.3638 0.9469
0.3532 0.8179
0.2471 0.5542
0.1982 0.6471

```

9. Example 2

In the second example 2000 terms of an ARMA(1,1) time series (with $\sigma^2 = 1.0$, $\theta = 0.9$ and $\phi = 0.4$) are generated using the function `nag_arma_time_series` (g05hac). The Kalman filter and optimisation function `nag_opt_bounds_no_deriv` (e04jbc) are then used to find the maximum likelihood estimate for the time series parameters θ and ϕ . The ARMA(1,1) time series is defined by

$$y_k = \phi y_{k-1} + \epsilon_k - \theta \epsilon_{k-1}$$

This has the following state space representation (Harvey and Phillips 1979)

$$x_k = \begin{pmatrix} \phi & 1 \\ 0 & 0 \end{pmatrix} x_{k-1} + \begin{pmatrix} 1 \\ -\theta \end{pmatrix} \epsilon_k$$

$$y_k = (1 \ 0) x_k$$

where the state vector $x_k = \begin{pmatrix} y_k \\ -\theta\epsilon_k \end{pmatrix}$ and ϵ_k is uncorrelated white noise with zero mean and variance σ^2 , i.e.,

$$E[\epsilon_k] = 0, E[\epsilon_k\epsilon_k] = \sigma^2, E[y_k\epsilon_k] = \sigma^2 \text{ and } E[\epsilon_k\epsilon_{k-1}] = 0$$

Since $\sigma^2 = 1$ we arrive at the following Kalman Filter matrices

$$A_k = \begin{pmatrix} \phi & 1 \\ 0 & 0 \end{pmatrix}, B_k = \begin{pmatrix} 1 \\ -\theta \end{pmatrix}$$

$$C_k = (1 \ 0), Q_k = 0 \text{ and } R_k = 1$$

The initial estimates for the state vector, $x_{1|0}$, and state covariance matrix, $P_{1|0}$, are:

$$x_{1|0} = E[x_k] = 0 \text{ and } P_{1|0} = E[x_k x_k^T] = \begin{pmatrix} E[y_k y_k] & -\theta E[y_k \epsilon_k] \\ -\theta E[y_k \epsilon_k] & \theta^2 E[\epsilon_k \epsilon_k] \end{pmatrix}$$

Since $E[y_k y_k] = \gamma_o = \frac{(1 + \theta^2 - 2\phi\theta)\sigma^2}{(1 - \phi^2)}$ (Wei 1990)

$$P_{1|0} = \begin{pmatrix} \gamma_o & -\theta \\ -\theta & \theta^2 \end{pmatrix}$$

Using $P_{1|0} = S_{1|0} S_{1|0}^T$ gives an initial Cholesky “square root” of

$$S_{1|0} = \begin{pmatrix} \sqrt{\gamma_o} & 0 \\ \frac{-\theta}{\sqrt{\gamma_o}} & \theta \sqrt{\frac{\gamma_o - 1}{\gamma_o}} \end{pmatrix}$$

9.1. Program Text

```
#ifndef NAG_PROTO
static void ex2 (void)
#else
    static void ex2 ()
#endif
{
    /*
    Example program to illustrate the use of the kalman filter to estimate the
    parameters of an ARMA(1,1) time series model.
    Note : theta_phi[0] contains theta (moving average coefficient), and
    theta_phi[1] contains phi (autoregressive coefficient)
    */

    double theta_phi[2], g[2], bl[2], bu[2];
    double objf;
    Integer n = 2;
    Nag_BoundType bound;
    Integer ip = 1, iq = 1;
    double sphi[1], stheta[1], sy[NY];
    double mean = 0.0, vara = 1.0;
    double ref[20];
    Boolean start;
    Integer seed = 1238;
    Integer nterms = NY;
    static NagError fail;
    static Nag_Comm comm;
    static Nag_E04_Opt options;

    Vprintf("\n\nExample 2\n\n");
    g05cbc(seed);
```

```
stheta[0] = 0.9;
sphi[0]   = 0.4;

start = TRUE;
g05hac(start, ip, iq, sphi, stheta, mean, vara, nterms,
        sy, ref, NAGERR_DEFAULT);

theta_phi[0] = 0.5;    /* initial guess */
theta_phi[1] = 0.5;

fail.print = TRUE;
bound = Nag_Bounds;
bl[0] = -1.0;
bu[0] = 1.0;
bl[1] = -1.0;
bu[1] = 1.0;
comm.user = &sy[0];
e04xxc(&options);
options.print_level = Nag_NoPrint;
options.list = FALSE;

e04jbc(n, objfun, bound, bl, bu, theta_phi, &objf, g, &options, &comm, &fail);

if(fail.code != NE_NOERROR && fail.code != NW_COND_MIN) {
    Vprintf("An error has occurred \n");
    exit(EXIT_FAILURE);
}

Vprintf("The estimates are :  theta = %7.3f, phi = %7.3f \n", theta_phi[0],
        theta_phi[1]);
exit(EXIT_SUCCESS);
}
```

9.2. Program Data

None.

9.3. Program Results

Example 2

The estimates are : theta = 0.890, phi = 0.379
