NAG C Library Function Document

nag_kalman_sqrt_filt_cov_var (g13eac)

1 Purpose

nag_kalman_sqrt_filt_cov_var (g13eac) 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 covariance filter with the system matrices in their original form.

2 Specification

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

void nag_kalman_sqrt_filt_cov_var (Integer n, Integer m, Integer p, double s[],
     Integer tds, const double a[], Integer tda, const double b[], Integer tdb,
     const double q[], Integer tdq, const double c[], Integer tdc,
     const 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

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

the estimate of X_i given observations Y_1 to Y_{i-1} is denoted by $\hat{X}_{i|i-1}$ with $\text{var}\left(\hat{X}_{i|i-1}\right) = P_{i|i-1} = S_i S_i^{\mathrm{T}}$. nag_kalman_sqrt_filt_cov_var (g13eac) performs one recursion of the square root covariance filter algorithm, summarized as follows:

$$\underbrace{\begin{pmatrix} R_i^{1/2} & C_i S_i & 0 \\ 0 & A_i S_i & B_i Q_i^{1/2} \end{pmatrix}}_{\text{(Pre-array)}} U = \underbrace{\begin{pmatrix} H_i^{1/2} & 0 & 0 \\ G_i & S_{i+1} & 0 \end{pmatrix}}_{\text{(Post-array)}}$$

where U is an orthogonal transformation triangularizing the pre-array. The triangularization is carried out via Householder transformations exploiting the zero pattern in the pre-array. The measurement-update for the estimated state vector X is

$$\hat{X}_{i|i} = \hat{X}_{i|i-1} - K_i\left[C_i \hat{X}_{i|i-1} - Y_i\right] \tag{1}$$

where K_i is the Kalman gain matrix, whilst the time-update for X is

$$\hat{X}_{i+1|i} = A_i \hat{X}_{i|i} + D_i U_i \tag{2}$$

where $D_i U_i$ represents any deterministic control used. The relationship between the Kalman gain matrix K_i and G_i is given by

$$A_i K_i = G_i \left(H_i^{1/2}\right)^{-1}$$

The function returns the product of the matrices A_i and K_i represented as AK_i, and the state covariance matrix $P_{i|i-1}$ factorized as $P_{i|i-1} = S_i S_i^{\mathrm{T}}$ (see the g13 Chapter Introduction for more information concerning the covariance filter).

4 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 Hesenberg 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

5 Arguments

1: **n** – Integer *Input*

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

Constraint: **n** ≥ 1.

2: **m** – Integer *Input*

On entry: the actual input dimension, m, i.e., the order of the matrix $Q_i^{1/2}$.

Constraint: **m** ≥ 1.

3: **p** – Integer *Input*

On entry: the actual output dimension, p, i.e., the order of the matrix $R_i^{1/2}$.

Constraint: **p** ≥ 1.

4: **s**[**n** $\times$ **tds**] – double *Input/Output*

On entry: the leading n by n lower triangular part of this array must contain S_i, the left Cholesky factor of the state covariance matrix $P_{i|i-1}$.

On exit: the leading n by n lower triangular part of this array contains S_{i+1}, the left Cholesky factor of the state covariance matrix $P_{i+1|i}$.

5: **tds** – Integer *Input*

On entry: the second dimension of the array **s** as declared in the function from which nag_kalman_sqrt_filt_cov_var (g13eac) is called.

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

6: **a**[**n** $\times$ **tda**] – const double *Input*

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

7: **tda** – Integer *Input*

On entry: the second dimension of the array **a** as declared in the function from which nag_kalman_sqrt_filt_cov_var (g13eac) is called.

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

8: **b**[**n** $\times$ **tdb**] – const double *Input*

On entry: 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 $B_iQ_i^{1/2}$. B_i is the input weight matrix and Q_i is the noise covariance matrix.

9: **tdb** – Integer *Input*

On entry: the second dimension of the array **b** as declared in the function from which nag_kalman_sqrt_filt_cov_var (g13eac) is called.

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

10: **q**[**m** $\times$ **tdq**] – const double *Input*

On entry: 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 $B_iQ_i^{1/2}$ then the array **q** must be set to the null pointer, i.e., `(double *)0`.

11: **tdq** – Integer *Input*

On entry: the second dimension of the array **q** as declared in the function from which nag_kalman_sqrt_filt_cov_var (g13eac) is called.

Constraint: **tdq** $\geq$ **m** if **q** is defined.

12: **c**[**p** $\times$ **tdc**] – const double *Input*

On entry: the leading p by n part of this array must contain C_i, the output weight matrix of the discrete system.

13: **tdc** – Integer *Input*

On entry: the second dimension of the array **c** as declared in the function from which nag_kalman_sqrt_filt_cov_var (g13eac) is called.

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

14: **r**[**p** $\times$ **tdr**] – const double *Input*

On entry: 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.

15: **tdr** – Integer *Input*

On entry: the second dimension of the array **r** as declared in the function from which nag_kalman_sqrt_filt_cov_var (g13eac) is called.

Constraint: **tdr** $\geq$ **p**.

16: **k**[**n** $\times$ **tdk**] – double *Output*

On exit: 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`.

17: **tdk** – Integer *Input*

On entry: the second dimension of the array **k** as declared in the function from which nag_kalman_sqrt_filt_cov_var (g13eac) is called.

Constraint: **tdk** $\geq$ **p** if **k** is defined.

18: **h**[**p** $\times$ **tdh**] – double *Output*

On exit: 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`.

19: **tdh** – Integer *Input*

On entry: the second dimension of the array **h** as declared in the function from which nag_kalman_sqrt_filt_cov_var (g13eac) is called.

Constraint: **tdh** $\geq$ **p** if **k** and **h** are defined.

20: **tol** – double *Input*

On entry: 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.

21: **fail** – NagError * *Input/Output*

The NAG error parameter, see the Essential Introduction.

6 Error Indicators and Warnings

NE_2_INT_ARG_LT

On entry, **tds** = ⟨*value*⟩ while **n** = ⟨*value*⟩. These parameters must satisfy **tds** $\geq$ **n**.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_INT_ARG_LT

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

NE_MAT_SINGULAR

The matrix sqrt(H) is singular.

NE_NULL_ARRAY

Array **h** has null address.

7 Accuracy

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

8 Further Comments

The algorithm requires $\frac{7}{6}n^3 + n^2\left(\frac{5}{2}p + m\right) + n\left(\frac{1}{2}m^2 + p^2\right)$ operations and is backward stable (see Vanbegin *et al.* (1989)).

9 Example

For this function two examples are presented. There is a single example program for nag_kalman_sqrt_filt_cov_var (g13eac), with a main program and the code to solve the two example problems is given in the functions EX1 and EX2.

Example 1 (EX1)

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.

Example 2 (EX2)

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 optimization function

nag_opt_nlp_solve (e04wdc) 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))

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

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

$$A_k = \begin{pmatrix} \phi & 1 \\ 0 & 0 \end{pmatrix}, B_k = \begin{pmatrix} 1 \\ -\theta \end{pmatrix}$$
$$C_k = \begin{pmatrix} 1 & 0 \end{pmatrix}, Q_k = 0 \text{ and } R_k = 1.$$

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

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

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

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

Using $P_{1|0} = S_{1|0}S_{1|0}^{\mathrm{T}}$ gives an initial Cholesky 'square root' of

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

9.1 Program Text

```
/* nag_kalman_sqrt_filt_cov_var (g13eac)  Example Program.
 *
 * Copyright 1996 Numerical Algorithms Group
 *
 * Mark 4, 1996.
 * Mark 5 revised, 1998.
 * Mark 6 revised, 2000.
 * Mark 7 revised, 2001.
 * Mark 8 revised, 2004.
 *
 */

#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 __cplusplus
extern "C" {
#endif
  static void objfun(Integer n, double theta_phi[], double *objf,
```

```
                      double g[], Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

static int ex1(void);
static int ex2(void);

int main(void)
{
  Integer exit_status_ex1=0;
  Integer exit_status_ex2=0;
  exit_status_ex1 = ex1();
  exit_status_ex2 = ex2();
  return exit_status_ex1 == 0 && exit_status_ex2 == 0 ? 0 : 1;

}

#define NY 2000
static void objfun(Integer n, double theta_phi[], double *objf,
                   double g[], Nag_Comm *comm)
     /* Routine to evaluate objective function. */
{
  Integer ione=1, itwo=2, k, m1=1, n1=2, nsteps=NY, nsum=0, p1=1;
  NagError fail;
  double a[2][2], ak[2][1], b[2][1], c[1][2], h[1][1], hs, k11, logdet=0.0;
  double phi, q[1][1], r[1][1];         /* There is no measurement noise */
  double s[2][2], ss=0.0, temp1, temp2, theta, tol=0.0;
  double v, xp[2], *y;

  INIT_FAIL(fail);

  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)
    {
      /* nag_kalman_sqrt_filt_cov_var (g13eac).
       * One iteration step of the time-varying Kalman filter
       * recursion using the square root covariance implementation
       */
```

```
      nag_kalman_sqrt_filt_cov_var(n1, m1, p1, &s[0][0], itwo, &a[0][0],
                                   itwo, &b[0][0], ione, &q[0][0], ione,
                                   &c[0][0], itwo, &r[0][0], ione, &ak[0][0],
                                   ione, &h[0][0], 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 A(I,J) a[(I)*tda + J]
#define B(I,J) b[(I)*tdb + J]
#define C(I,J) c[(I)*tdc + J]
#define K(I,J) k[(I)*tdk + J]
#define Q(I,J) q[(I)*tdq + J]
#define R(I,J) r[(I)*tdr + J]
#define S(I,J) s[(I)*tds + J]
#define H(I,J) h[(I)*tdh + J]
static int ex1(void)
{
  double *a=0, *b=0, *c=0, *k=0, *q=0, *r=0, *s=0, *h=0;
  Integer exit_status=0;
  Integer i, j;
  Integer m, n, p;
  Integer istep;
  double tol;
  Integer tda, tdb, tdc, tdk, tdq, tdr, tds, tdh;
  NagError fail;

  INIT_FAIL(fail);
  Vprintf("nag_kalman_sqrt_filt_cov_var (g13eac) Example Program Results\n\n");
  Vprintf("Example 1\n");
  /* Skip the heading in the data file */
  Vscanf("%*[^\n]");

  Vscanf("%ld%ld%ld%lf",&n,&m,&p,&tol);
  if (n>=1 && m>=1 && p>=1)
    {
      if ( !( a = NAG_ALLOC(n*n, double)) ||
           !( b = NAG_ALLOC(n*m, double)) ||
           !( c = NAG_ALLOC(p*n, double)) ||
           !( k = NAG_ALLOC(n*p, double)) ||
           !( q = NAG_ALLOC(m*m, double)) ||
           !( r = NAG_ALLOC(p*p, double)) ||
           !( s = NAG_ALLOC(n*n, double)) ||
           !( h = NAG_ALLOC(n*p, double)) )
        {
          Vprintf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
      tda = n;
      tdb = m;
      tdc = n;
      tdk = p;
      tdq = m;
      tdr = p;
      tds = n;
      tdh = p;
    }
  else
    {
      Vprintf("Invalid n or m or p.\n");
      exit_status = 1;
```

```
      return exit_status;
    }

  /* 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));
  if (q)
    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)
    {
      /* nag_kalman_sqrt_filt_cov_var (g13eac), see above. */
      nag_kalman_sqrt_filt_cov_var(n, m, p, s, tds, a, tda, b, tdb, q, tdq,
                                   c, tdc, r, tdr, k, tdk, h, tdh, tol, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from nag_kalman_sqrt_filt_cov_var (g13eac).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }
    }
  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");
    }
  if (k)
    {
      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("%8.4f ", K(i,j));
          Vprintf("\n");
        }
    }
 END:
  if (a) NAG_FREE(a);
  if (b) NAG_FREE(b);
  if (c) NAG_FREE(c);
  if (k) NAG_FREE(k);
  if (q) NAG_FREE(q);
  if (r) NAG_FREE(r);
  if (s) NAG_FREE(s);
  if (h) NAG_FREE(h);
  return exit_status;
}


static int ex2 (void)
{
```

```
/*
  Example program to iluustrate 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 exit_status=0;
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];
Nag_Boolean start;
Integer seed = 1238;
Integer nterms = NY;
Nag_Comm comm;
Nag_E04_Opt options;
NagError fail;

INIT_FAIL(fail);

Vprintf("\n\nExample 2\n\n");
/* nag_random_init_repeatable (g05cbc).
 * Initialize random number generating functions to give
 * repeatable sequence
 */
nag_random_init_repeatable(seed);

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

start = Nag_TRUE;
/* nag_arma_time_series (g05hac).
 * ARMA time series of n terms
 */
nag_arma_time_series(start, ip, iq, sphi, stheta, mean, vara, nterms,
                     sy, ref, &fail);
if (fail.code != NE_NOERROR)
  {
    Vprintf("Error from nag_arma_time_series (g05hac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

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

bound = Nag_Bounds;
bl[0] = -1.0;
bu[0] = 1.0;
bl[1] = -1.0;
bu[1] = 1.0;
comm.user = &sy[0];
/* nag_opt_init (e04xxc).
 * Initialization function for option setting
 */
nag_opt_init(&options);
options.print_level = Nag_NoPrint;
options.list = Nag_FALSE;

/* nag_opt_bounds_no_deriv (e04jbc).
 * Bound constrained nonlinear minimization (no derivatives
 * required)
 */
nag_opt_bounds_no_deriv(n, objfun, bound, bl, bu, theta_phi, &objf, g,
                        &options, &comm, &fail);
if(fail.code != NE_NOERROR && fail.code != NW_COND_MIN)
```

```
    {
      Vprintf("Error from nag_opt_bounds_no_deriv (e04jbc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* nag_opt_free (e04xzc).
   * Memory freeing function for use with option setting
   */
  nag_opt_free(&options, "all", &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from nag_opt_free (e04xzc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  Vprintf("The estimates are :  theta = %7.3f, phi = %7.3f \n", theta_phi[0],
          theta_phi[1]);
 END:
  return exit_status;

}
```

9.2 Program Data

None.

9.3 Program Results

```
nag_kalman_sqrt_filt_cov_var (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


Example 2

The estimates are :  theta =   0.890, phi =   0.379
```