A Program for Design and Performance Evaluation of Multivariate Exponentially Weighted Moving Average Control Charts

Douglas M. Hawkins

Sungwoon Choi and Sanghoon Lee

School of Statistics, University of Minnesota

Technical Report #641

May 13, 2002

A Program for Design and Performance Evaluation of Multivariate Exponentially Weighted Moving Average Control Charts

DOUGLAS M. HAWKINS[1]
University of Minnesota, Minneapolis, Minnesota 55455

SUNGWOON CHOI[2] and SANGHOON LEE[3]
Kyung Won University, Seongnam City, Kyungki Do, Korea

The in-control and out-of-control performance of the traditional multivariate exponentially weighted control charts is a function of a number of parameters – the dimensionality, the in-control target average run length, the weighting scheme and the non-centrality of the shifted data. This dependence on four parameters makes the provision of printed tables a daunting task. When one goes beyond the simple diagonal-weighting scheme, this list is extended to include the covariance matrix of the data and the weighting matrix, turning an already difficult task into a practical impossibility. The program addresses this difficulty by allowing the user to specify the design parameters. It then uses simulation to estimate the control limit that gives the target in-control ARL, and estimates the out-of-control ARL. Its execution on standard desktop computers is sufficiently fast to make experimentation with design choices easy.

The general multivariate exponentially-weighted moving average chart.

Suppose that a process yield a vector of measurements that that follows a multivariate $N(\mu,\Sigma)$ distribution. The full exponentially weighted moving average (FEWMA) chart is defined by Choi, Lee and Hawkins (2002) by the recursion

$$\mathbf{y}_0 = \mathbf{0}$$
$$\mathbf{y}_n = \mathbf{R}(\mathbf{x}_n - \mu) + (\mathbf{I} - \mathbf{R})\mathbf{y}_{n-1}$$

Writing Σ_n for the covariance matrix of $\mathbf{y}_n$, we compute the squared Mahalanobis distance

$$D_n = \mathbf{y}_n' \Sigma_n^{-1} \mathbf{y}_n$$

and signal if D_n exceeds a control limit h.

Lowry *et al* (1992) first discussed multivariate exponentially weighted moving average control charts. In their definition, they restricted the weight matrix **R** to be a multiple of the identity matrix, $\mathbf{R}=r\mathbf{I}$. While this choice certainly simplifies working with the multivariate EWMA by

---

[1]Dr. Hawkins is Professor, School of Statistics. He is a Fellow of ASQ. His research was partially supported by National Science Foundation (NSF) grant DMS 9803622 and ACI 9619020.

[2]Dr. Choi is an associate professor of the department of Industrial Engineering.

[3]Dr. Lee is an associate professor of the department of Industrial Engineering.

reducing the necessary design choices on **R** to that of the single scalar *r*, doing so reduces the performance that could be obtained from a more general form. For example, Choi *et al* (2002) argue that extending **R** to the form **R**= a**I** + b**J** where **J** is a matrix of 1's can lead to appreciably faster response to a shift in mean.

For all *n*, y_n follows a multivariate normal distribution with mean vector μ. Its covariance matrix is given recursively as

$$\Sigma_0 = \mathbf{0},$$
$$\Sigma_n = \mathbf{R}\Sigma\mathbf{R}'+(\mathbf{I}-\mathbf{R})\Sigma_{n-1}(\mathbf{I}-\mathbf{R})'$$

As n increases, this covariance matrix tends to an asymptotic 'steady state' form, provided the eigenvalues of **R** lie between 0 and 1, as is generally the case. This steady state form is given by the system of linear equations

$$\Sigma_\infty - (\mathbf{I}-\mathbf{R})\Sigma_\infty(\mathbf{I}-\mathbf{R})' = \mathbf{R}\Sigma\mathbf{R}'$$

There are two average run lengths (ARL's) that are of interest. The 'initial state' ARL assumes that the monitoring of D_n begins with the very first observation. The 'steady state' ARL assumes that the monitoring does not begin until the FEWMA has run long enough for the covariance matrix of y_n is effectively at its steady state value Σ_∞.

Program operation

Analytical results for the ARL of the FEWMA do not appear to be to hand, and so the program operates by simulation. It has two modes of operation – the design mode is used to get the control limit *h* needed to give the target in-control ARL. The program also finds the out-of-control ARL at a user-specified shift in the mean vector. The second mode – evaluation – is used when the control limit *h* is specified and the ARL is required.

In the design mode a sequence of trial *h* values is generated using the Robbins-Monro (R-M) algorithm. In this, we start with some initial value h_0. After this the successive h_k values are generated according to the scheme

Let r_k be the observed length of run *k* using its control limit h_k.
If $r_k < T$ (where T is the target ARL), then define $r_{k+1} = r_k * (1 + e^{-1}a/(k+b))$
If $r_k > T$ then define $r_{k+1} = r_k * (1 - (1-e^{-1})\, a/(k+b))$

The tuning constants *a* and *b* are set to 5 and 100 respectively, values that were reached after some experimentation but that are certainly not immutable.

The R-M algorithm as described would commonly be used to estimate the *h* having T as its upper $(1-e^{-1})$ fractile – the sequence h_k converges to this fractile. To the extent that the run length distribution can be approximated by a large-mean geometric distribution (which is quite close to the truth in the null case) this fractile would correspond to the *h* giving an ARL of T. For our purposes though, the main virtue of the R-M algorithm does not lie in this property, but rather in the fact that it produces a sequence of different h_k values that cluster around the required *h*. We then use this sequence of (h_k, r_k) values to fit the linear regression

$$r_k = \beta_0 + \beta_1 h_k + e_k$$

where e_k is a zero-mean error term.

The required h is then found by solving

$$T = b_0 + b_1 h$$

where b_0 and b_1 are the least squares estimates of the intercept and slope respectively.

An approximate 95% confidence interval for h is found by solving for the values of h at which T will be on the upper and the lower edge of the 95% confidence band for the true regression line:-

$$T = b_0 + b_1 h \pm 2\sqrt{\frac{1}{K} + \frac{(h-\bar{h})^2}{(K-1)s^2}} \qquad (*)$$

where K is the number of (h,r) pairs used in the calibration, and $\bar{h}$ and s are the mean and standard deviation respectively of the h values.

In a design phase, the out-of-control ARL is estimated in a similar way, by regressing the observed out-of-control run lengths on the trial control limits h_k and evaluating this regression line at the estimated h. The approximate 95% confidence interval for the out-of-control case is given by using the confidence interval (*) for this regression.

Estimating the ARL for a given h is done in a more conventional way. Data are simulated and the run lengths measured. The average is reported along with a two standard error band as the approximate confidence interval. Usually this mode will be used following a design run to investigate the out-of-control performance of the chart for different shifts, but there is nothing to prevent its use for looking at the null case for a fixed h by setting the shift vector equal to zero.

The heart of the code is the generation of sequences of random $N(\mathbf{0},\Sigma)$ vectors. This is done by generating vectors of p independent N(0,1) variables. Multiplying each such vector by the Cholesky triangular factor of Σ then gives a vector with the desired distribution. The random normal generator uses the ratio/rejection method outlined on page 313 of Ripley (1983) (but note that Ripley's equations 2a and 2b have the rejection inequality reversed).

Initial state or steady state

The recursive equations for the FEWMA given in the introduction implicitly define the initial state FEWMA. If the steady state is to be evaluated, then the initial y_0 is sampled randomly from the steady-state distribution rather than being set to zero. Whether the initial state or steady state is used, the out-of-control ARL is estimated by adding the shift in the mean vector from the first simulated data vector onward.

Data required

The program operates by interrogating the keyboard for its input. This information is:-

1. The dimensionality of the problem
2. The covariance matrix of the vectors $\boldsymbol{x}_n$

3. The smoothing matrix **R**. As implemented, the program uses **R** of the form a**I**+b**J**, and parameterizes this by two constants r and c. The constant r defines the total weight given to the current data vector, and c defines how this weight is to be shared between the diagonal and off-diagonal elements of **R**. Specifically
   $a = \frac{r(1-c)}{1+(p-1)c}$, $b = \frac{cr}{1+(p-1)c}$, where p is the dimensionality of the data.
   Setting $c = 0$ gives the diagonal EWMA defined by Lowry *et al* (1992).
4. Whether the situation to be simulated is a initial state or steady state
5. Whether the run is a design run to estimate h and out-of-control performance, or an evaluation run to estimate out-of-control performance of a specified scheme.
6. If the former, then the in-control ARL must be provided
7. If the latter, then the value of h must be provided
8. The vector of the shifts in the mean vector in going to the out-of-control state
9. The number of runs to simulate.
10. Two integer seeds for the random number generator. If the same seeds are provided on different runs, the identical results will be obtained.

Example and timing.

We illustrate with an example testing the benefit to be got from off-diagonal weights in the **R** matrix. This is of an 8-component measurement vector with all correlations equal to 0.8. Two runs were made. In both, the diagonal elements of **R** were set to 0.1. In one, the off-diagonal elements were set to zero, giving the Lowry *et al* DEWMA; in the other the off-diagonal elements were set to 0.1.

The chart is designed for an in-control ARL of 300, and the out-of-control performance evaluated for a shift of 0.25 in the first two components and 0 in the other components. We chose the settings r =0.06, so that the most recent observation vector is given a weight of 0.06 and the history a weight of 0.94, and c =0.75, so that three quarters of the total weight is given to the off-diagonal elements of the vector. The output from the program (which includes an echo of the inputted answer to the keyboard queries) follows. The information supplied by the user is italicized.

```
Program to find the control limit for the
general multivariate EWMA and/or its performance
 Give the number of dimensions

 8
What r and c values do you want to use?

   0.060   0.750
Now give the elements of the covariance matrix
(you only need to give the lower triangle)
  1.000
  0.800   1.000
  0.800   0.800   1.000
  0.800   0.800   0.800   1.000
  0.800   0.800   0.800   0.800   1.000
  0.800   0.800   0.800   0.800   0.800   1.000
  0.800   0.800   0.800   0.800   0.800   0.800   1.000
  0.800   0.800   0.800   0.800   0.800   0.800   0.800   1.000
```

```
Do you want initial state or steady state?

 Initial state

Steady state covariance matrix
  0.0257   0.0255   0.0255   0.0255   0.0255   0.0255   0.0255   0.0255
  0.0255   0.0257   0.0255   0.0255   0.0255   0.0255   0.0255   0.0255
  0.0255   0.0255   0.0257   0.0255   0.0255   0.0255   0.0255   0.0255
  0.0255   0.0255   0.0255   0.0257   0.0255   0.0255   0.0255   0.0255
  0.0255   0.0255   0.0255   0.0255   0.0257   0.0255   0.0255   0.0255
  0.0255   0.0255   0.0255   0.0255   0.0255   0.0257   0.0255   0.0255
  0.0255   0.0255   0.0255   0.0255   0.0255   0.0255   0.0257   0.0255
  0.0255   0.0255   0.0255   0.0255   0.0255   0.0255   0.0255   0.0257
 Do you want to find h or do you already know it?
 (answer Yes to get  h or No to do the  OOC ARL)
 Yes - get h and OOC ARL
 What is the null ARL?

     300.
Now give the vector of shifts in the mean

 0.2500  0.2500  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
This gives root noncentrality       0.688
DEWMA steady state noncentrality    3.913
FEWMA steady state noncentrality   19.756
How many runs do you want simulated?

10000
Give two integer seeds less than 30000

    30     40
 Empiric steady-state covariance matrix
   0.0257    0.0255    0.0255    0.0255    0.0255    0.0255    0.0255
0.0255
   0.0255    0.0257    0.0255    0.0255    0.0255    0.0255    0.0255
0.0255
   0.0255    0.0255    0.0257    0.0255    0.0255    0.0255    0.0255
0.0255
   0.0255    0.0255    0.0255    0.0257    0.0255    0.0255    0.0255
0.0255
   0.0255    0.0255    0.0255    0.0255    0.0257    0.0255    0.0255
0.0255
   0.0255    0.0255    0.0255    0.0255    0.0255    0.0257    0.0255
0.0255
   0.0255    0.0255    0.0255    0.0255    0.0255    0.0255    0.0257
0.0255
   0.0255    0.0255    0.0255    0.0255    0.0255    0.0255    0.0255
0.0257
  Current EWMA of shift
   0.1937    0.1937    0.0188    0.0188    0.0188    0.0188    0.0188
0.0188

 Run    1000  h  15.812 IC RL   1019. OOC RL      9.
 Estimated h is   15.129 with CI   14.471   15.453
 OOC ARL is   13.805  with CI   12.873   14.736

(lines of progress-report output deleted.)

 Run   10000  h  15.818 IC RL     27. OOC RL     16.
 Estimated h is   15.071 with CI   14.645   15.272
 OOC ARL is   13.875  with CI   13.270   14.480
```

The program reports the steady state covariance matrix of the FEWMA. It lists three non-centrality parameters. The first is the square root of the non-centrality parameter $(\mu-\mu_o)'\Sigma^{-1}(\mu-\mu_o)$, where μ_o and μ represent the in-control and out-of-control mean vectors respectively. It also gives the asymptotic FEWMA noncentrality parameter $(\mu-\mu_o)'\Sigma_\infty^{-1}(\mu-\mu_o)$ which predicts the FEWMA's ability to respond to small shifts in the direction given. The DEWMA noncentrality parameter is the corresponding figure for the diagonal EWMA with the same *r*. The relative magnitude of the DEWMA and FEWMA noncentralities indicates the benefit of using a non-zero *c*.

The program gives intermediate 'progress reports', 8 of which have been deleted from the transcript but whose format is the same as the last three lines shown. These show the most recent trial *h* and its in-control and out-of-control run lengths, along with the current estimates of *h* and the out-of-control ARL along with 95% approximate confidence intervals, which could be used to terminate the run early if the user thought the current estimates were good enough.

This run took 210 seconds on a 400 MHz Pentium computer. Note that the approximation obtained in about 20 seconds computing simulating 1000 runs was already adequate for exploratory design purposes, an illustration of how using the program makes exploration of the design choices quick and easy.

Another run with $c = 0$ and all other parameters unchanged shows the comparative performance of the diagonal EWMA of Lowry *et al.* The out-of-control ARL of the two schemes was estimated as:

Full matrix EWMA	13.9
Diagonal EWMA	22.9

The run length using the restricted diagonal weighting matrix was 50% higher than that using the more general form, showing a substantial benefit from using a full weighting matrix **R**.

The program can also be checked against Table 2 of Lowry *et al.* Our reconstruction of that table (along with confidence intervals) is:-

		p		
λ		2	3	4
	r	0.06	0.06	0.06
0.5	h	7.876 ± 0.028	9.982 ± 0.032	11.857 ± 0.026
	OOC ARL	25.49 ± 0.15	28.37 ± 0.17	30.52 ± 0.19
	r	0.16	0.16	0.14
1.0	h	9.411 ± 0.030	11.659 ± 0.030	13.417 ± 0.032
	OOC ARL	9.614 ± 0.05	10.78 ± 0.06	11.46 ± 0.06
	r	0.24	0.22	0.20
1.5	h	9.898 ± 0.027	12.063 ± 0.024	13.927 ± 0.030
	OOC ARL	5.26 ± 0.03	5.81 ± 0.03	6.21 ± 0.03
	r	0.34	0.30	0.28
2.0	h	10.209 + 0.025	12.372 + 0.027	14.322 + 0.036
	OOC ARL	3.42 ± 0.02	3.80 ± 0.02	4.03 ± .02

These values are close to those of Lowry *et al.*, though their values do not, for the most part, lie within the confidence intervals given by FEWMA. However as no precision is given with the Lowry *et a.l* table, it is impossible to say whether the two sets of results agree to within random sampling fluctuations.

Language and code

The code for FEWMA was written in FORTRAN 77 to make it as widely useful as possible. Self-contained source is an appendix to this report.

References

Choi, S., Lee, S., and Hawkins, D. M. (2002) 'A General Multivariate Exponentially Weighted Moving Average Control Chart' Technical report 640, School of Statistics, University of Minnesota.

Lowry, C.A., Woodall, W.H., Champ, C.W. and Rigdon, S.E. (1992), 'A Multivariate Exponentially Weighted Moving Average Control Chart,' *Technometrics*, 34, 46-53.

Ripley, B. D., (1983), 'Computer generation of random variables: A tutorial', *International Statistical Review*,51, 301—319.

Appendix Fortran 77 FEWMA source code

```
      program fewma
c
c     Fortran 77 program to find the in-control and out-of-control
c     ARL of a general multivariate exponentially weighted moving
c     average chart.

      implicit double precision (a-h,o-z)
      parameter (maxnor = 10, maxpar=55)
      dimension sigma(maxnor,maxnor),gmsig(maxnor,maxnor),
     1 sig1(maxnor,maxnor),rmat(maxnor,maxnor),rbar(maxnor,maxnor),
     2 prod(maxnor,maxnor),gminv(maxnor,maxnor+2),
     3 facsig(maxnor,maxnor), sigold(maxnor,maxnor),
     4 dew(maxpar), work(maxpar), data(maxnor), delta(maxnor),
     5 work2(maxpar),stedsl(maxpar,maxpar+2),facstd(maxnor,maxnor),
     6 dewdl(maxnor),work3(maxpar),work4(maxpar)
      character*60 filnam
      write(*,*) 'Program to find the control limit for the '
      write(*,*) 'general multivariate EWMA and/or its performance'
      write(*,1001) 'Give the name of the file to write results'
      read(*,'(a)') filnam
      open(11,file=filnam)
      write(11,*) ' Program to find the control limit for the '
      write(11,*) 'general multivariate EWMA and/or its performance'
      write(*,1001) 'Give the number of dimensions'
 1001 format(1x,a/)
```

```
      read(*,*) nord
      write(11,1001) 'Give the number of dimensions'
      write(11,*) nord
      npar = nord * (nord+1) / 2
      call dorest(sigma,gmsig,sig1,rmat,rbar,prod,gminv,dew,work,
     1 work2,work3,work4,
     1 dewdl,facsig,facstd,stedsl,sigold,data,delta,nord,npar)
      end

      subroutine dorest(sigma,gmsig,sig1,rmat,rbar,prod,gminv,dew,work,
     1 work2,work3,work4,
     1 dewdl,facsig,facstd,stedsl,sigold,data,delta,nord,npar)
      implicit double precision (a-h,o-z)
      dimension sigma(nord,nord),gmsig(nord,nord),sig1(nord,nord),
     1 rmat(nord,nord),rbar(nord,nord),prod(nord,nord),
     2 gminv(nord,nord+2),dew(npar),work(npar),sigold(nord,nord),
     3 facsig(nord,nord),data(nord),delta(nord),work2(npar),
     4 work3(npar),work4(npar),
     5 dewdl(nord),stedsl(npar,npar+2),facstd(nord,nord)

      logical steady,whchmt,over,gotin,gotout,echo,design,tripper
      character answer

      data zero/0.d0/, one /1.d0/, two /2.d0/, four /4.d0/,
     1 half /0.5d0/numfac/ 5.d0/, denfac/100.d0/,einv/0.368d0/,
     2 oneein/0.632d0/

      write(*,1001) 'What r and c values do you want to use?'
 1001 format(1x,a/)
      read(*,*) r, c
      write(*,'(2f8.3)') r, c
      write(11,1001) 'What r and c values do you want to use?'
      write(11,'(2f8.3)') r, c
      fnord = nord
      den = one + (fnord - one) * c
      od = r * c / den
      write(*,*) 'Now give the elements of the covariance matrix'
      write(*,1001) '(you only need to give the lower triangle)'
      write(11,*) 'Now give the elements of the covariance matrix'
      write(11,*) '(you only need to give the lower triangle)'
      ai = r / den - od
      do 10 i = 1, nord
        write(*,*) ' Row',i,': '
        read(*,*) (sigma(i,j),j=1,i)
        write(11,'(10f8.3)') (sigma(i,j),j=1,i)
        rmat(i,i) = r / den
        rbar(i,i) = one - r / den
        do 10 j = 1, i-1
          sigma(j,i) = sigma(i,j)
          rmat(i,j) = od
          rmat(j,i) = od
          rbar(i,j) = -od
          rbar(j,i) = -od
 10     continue
c
c     Get R sigma R'
c
      do 30 i = 1, nord
        do 30 j = 1, nord
          sum = zero
```

```
            do 40 k = 1, nord
            sum = sum + sigma(i,k) * rmat(j,k)
 40         continue
          prod(i,j) = sum
 30     continue

      do 50 i = 1, nord
        do 50 j = 1, nord
          sum = zero
          do 60 k = 1, nord
            sum = sum + rmat(i,k) * prod(k,j)
 60         continue
          sig1(i,j) = sum
 50     continue

c
c     Get triangular factor of sigma
c
      call matfac(sigma,facsig,nord)

      write(* ,1001) 'Do you want initial state or steady state?'
      write(11,1001) 'Do you want initial state or steady state?'
      read(*,'(a)') answer
      steady = answer .eq. 'S' .or. answer .eq. 's'
      if (steady) then
        write(11,*) ' Steady state'
        else
        write(11,*) ' Initial state'
        endif
c
c     Get asymptotic covariance matrix
c
      nravl = 0
      do 510 i = 1, nord
        do 510 j = i, nord
          nravl = nravl + 1
          do 520 k = 1, npar
 520      stedsl(nravl,k) = zero
        stedsl(nravl,nravl) = one
        work3(nravl) = sig1(i,j)
        work4(nravl) = zero
        mravl = 0
        do 530 k = 1, nord
          mravl = mravl + 1
          stedsl(nravl,mravl) = stedsl(nravl,mravl)-rbar(i,k)*rbar(j,k)
          do 530 m = k+1, nord
            mravl = mravl + 1
            stedsl(nravl,mravl) = stedsl(nravl,mravl) - rbar(i,k) *
     1        rbar(j,m) - rbar(i,m) * rbar(j,k)
 530     continue
 510  continue

      echo = .false.
      call solve(stedsl,work3,work4,work,work2,npar,echo)
      nravl = 0
      write(11,'(/'' Steady state covariance matrix'')')
      do 540 i = 1, nord
        do 550 j = i, nord
          nravl = nravl + 1
          gmsig(i,j) = work(nravl)
```

```
          gmsig(j,i) = work(nravl)
 550      continue
        write(11,'(8f9.4)') (gmsig(i,j),j=1,nord)
 540    continue

      call matfac(gmsig,facstd,nord)

      write(*,*) 'Do you want to find h or do you already know it?'
      write(*,1001) '(answer Yes to get  h or No to do the  OOC ARL)'
      read(*,'(a)') answer
      design = answer .eq. 'Y' .or. answer .eq. 'y'
      write(11,*) ' Do you want to find h or do you already know it?'
      write(11,*) ' (answer Yes to get  h or No to do the  OOC ARL)'
      if (design) then
        write(11,*) ' Yes - get h and OOC ARL'
        baseh = fnord
        write(*,1001) ' What is the null ARL?'
        read(*,*) tararl
        write(11,1001) ' What is the null ARL?'
        write(11,'(f10.0)') tararl
        else
        write(11,*) ' No - I have h, just get the OOC ARL'
        write(*,1001) ' What is the in-control h?'
        write(11,*) ' What is the in-control h?'
        read(*,*) baseh
        write(11,'(f10.3)') baseh
        endif
      write(*,1001) 'Now give the vector of shifts in the mean'
      write(11,1001) 'Now give the vector of shifts in the mean'
      read(*,*) (delta(i),i=1,nord)
      write(11,'(10f8.4)') (delta(i),i=1,nord)
c
c     Get noncentrality
c
      sumdel = zero
      do 890 i = 1, nord
 890    sumdel = sumdel + delta(i)
      do 880 i = 1, nord
 880    work3(i) = ai * delta(i) + od * sumdel
      flam = zero
      flamz = zero
      do 900 i = 1, nord
        sum = delta(i)
        sumz = delta(i)
        do 910 j = 1, i-1
          sum = sum - facsig(i,j) * work(j)
          sumz = sumz - facstd(i,j) * work2(j)
 910      continue
        work(i) = sum / facsig(i,i)
        work2(i) = sumz / facstd(i,i)
        flam = flam + work(i) ** 2
        flamz = flamz + work2(i) ** 2
 900    continue
      flam = sqrt(flam)
      write(* ,'('' This gives root noncentrality   '',f9.3)') flam
      write(11,'('' This gives root noncentrality   '',f9.3)') flam
      dflam = flam * sqrt((two - r) / r)
      write(* ,'('' DEWMA steady state noncentrality'',f9.3)') dflam
      write(11,'('' DEWMA steady state noncentrality'',f9.3)') dflam
      flamz = sqrt(flamz)
```

```
      write(11,'('' FEWMA steady state noncentrality'',f9.3)') flamz
      write(* ,'('' FEWMA steady state noncentrality'',f9.3)') flamz
c
      write(*,1001) 'How many runs do you want simulated?'
      write(11,1001) 'How many runs do you want simulated?'
      read(*,*) nrun
      write(11,*) nrun
      write(*,1001) 'Give two integer seeds less than 30000'
      read(*,*) ij,kl
      write(11,1001) 'Give two integer seeds less than 30000'
      write(11,'(2i7)') ij,kl
      ngap = nrun / 10
      write(*,*) '.....Run starting.....'
c
      tripper = .not. steady

      xbar = zero
      ybar = zero
      ssx  = zero
      ssy  = zero
      sxy  = zero
      fn   = zero
      xbaro = zero
      ybaro = zero
      ssxo  = zero
      ssyo  = zero
      sxyo  = zero
      fno   = zero
      over = .false.
      echo = .false.
      do 20 irun = 1, nrun

      do 600 i = 1, nord
        dew(i) = zero
        dewdl(i) = zero
 600    continue

      if (.not. steady) then
        do 610 i = 1, nord
          do 610 j = 1, nord
 610      sigold(i,j) = zero
        endif

      run = zero
      tsqmax = zero
      gotin  = .not. design
      gotout = .false.

c
c     Loop point for generating new case
c

 70   continue
      run = run + one
      if (run .gt. 50000) then
        write(*,*) 'Seems to be stuck. run',run,' h',baseh,' tsq',
     1   tsq,' tsqmax',tsqmax,' t2sq',t2sq
        write(*,1002) 'gmsig',gmsig
        write(*,1002) 'gminv',gminv
        write(*,1002) 'data',(data(ii),ii=1,nord)
```

```
        write(*,1002) 'dew',(dew(ii),ii=1,nord)
        write(*,1002) 'delta',(delta(ii),ii=1,nord)
        write(*,1002) 'work ',(work(ii),ii=1,nord)
        write(*,1002) 'work2',(work2(ii),ii=1,nord)
 1002   format(/1x,a,(10g14.7))
        echo = .true.
        endif
c
c     Transform data
c
      innerc = 0
      whchmt = steady .and. run .lt. 1.1
 680  continue
      call randn(data,nord,ij,kl)
      if (whchmt) call randn(dew,nord,ij,kl)
      do 640 i = 1, nord
        work2(i) = dew(i)
 640    work(i) = data(i)
      do 620 i = 1, nord
        data(i) = zero
        if (whchmt) dew(i) = zero
        do 620 j = 1, nord
          if (whchmt) dew(i) = dew(i) + facstd(i,j) * work2(j)
          data(i) = data(i) + facsig(i,j) * work(j)
 620    continue
      if (whchmt .and. over) then
c
c       Make sure a steady state run starts out in control
c
        innerc = innerc + 1
        if (innerc .gt. 100) then
          write(*,*) '100 tries to get a steady state start have failed'
          stop
          endif
        do 660 i = 1, nord
          dewdl(i) = zero
          do 660 j = 1, nord
 660        gminv(i,j) = gmsig(i,j)
        call solve(gminv,dew,dewdl,work,work2,nord,echo)
        tsq = zero
        do 670 i = 1, nord
 670      tsq = tsq + dew(i) * work(i)
        if (tsq .gt. baseh) then
          if (innerc .gt. 10) write(*,'('' T squared, h, EWMA'',i5,
     1      (8f8.3))') tsq, baseh,(dew(i),i=1,nord)
          go to 680
          endif
        endif
c
      do 90 i = 1, nord
        work(i) = dew(i)
        work2(i) = dewdl(i)
        dew(i) = zero
        dewdl(i) = zero
 90     continue
      do 95 i = 1, nord
        do 95 j = 1, nord
          dew(i)   = dew(i)  +rmat(i,j)*data(j) +rbar(i,j)*work (j)
          dewdl(i) = dewdl(i)+rmat(i,j)*delta(j)+rbar(i,j)*work2(j)
 95     continue
```

```
      if (.not. steady) then
        do 80 i = 1, nord
          do 80 j = 1, nord
            gmsig(i,j) = sig1(i,j)
            sum = zero
            do 700 k = 1, nord
700           sum = sum + sigold(i,k) * rbar(k,j)
            prod(i,j) = sum
80        continue
        do 100 i = 1, nord
          do 100 j = 1, nord
            do 100 k = 1, nord
100       gmsig(i,j) = gmsig(i,j) + rbar(i,k) * prod(k,j)
        endif

      do 110 i = 1, nord
        do 110 j = 1, nord
          sigold(i,j) = gmsig(i,j)
110       gminv(i,j) = gmsig(i,j)

      if (tripper .and. run .gt. 500) then
        tripper = .false.
        write(11,*) ' Empiric steady-state covariance matrix'
        do 760 i = 1, nord
760       write(11,'(8f10.4)') (gmsig(i,j),j=1,nord)
        write(11,*) ' and EWMA of shift'
        write(11,'(8f10.4)') (dewdl(i),i=1,nord)
        endif

      if (echo) write(*,*) ' dew', dew, '  dewdl',dewdl

      call solve(gminv,dew,dewdl,work,work2,nord,echo)
      tsq = zero
      t2sq = zero
      do 130 i = 1, nord
        tsq = tsq + dew(i) * work(i)
        t2sq = t2sq + (dew(i) - dewdl(i)) * (work(i) - work2(i))
130     continue
      tsqmax = max(tsq,tsqmax)


      if (.not. gotin) then
        if (tsq .gt. baseh .or. run .gt. 50002) then

          gotin = .true.
          runic = run
          y =    (run)
          x = baseh

          corrf = numfac / (float(irun) + denfac)
          if (run .lt. tararl) then
            baseh = baseh * (one + einv * corrf)
            else
            baseh = baseh * (one - oneein * corrf)
            over = .true.
            endif
          if (.not. over) go to 20
          devx = x - xbar
          devy = y - ybar
          fn1 = fn + one
```

```
          xbar = xbar + devx / fn1
          ybar = ybar + devy / fn1
          ratio = fn / fn1
          ssx = ssx + ratio * devx ** 2
          ssy = ssy + ratio * devy ** 2
          sxy = sxy + ratio * devx * devy
          fn = fn1
          endif
        endif
      if (.not. gotout) then
        if (t2sq .gt. baseh .or. run .gt. 50002) then
          runooc = run
          gotout = .true.
          x = baseh
          y = run
          devx = x - xbaro
          devy = y - ybaro
          fno1 = fno + one
          xbaro = xbaro + devx / fno1
          ybaro = ybaro + devy / fno1
          ssxo = ssxo + fno * devx ** 2 / fno1
          ssyo = ssyo + fno * devy ** 2 / fno1
          sxyo = sxyo + fno * devx * devy / fno1
          fno = fno1
          endif
        endif
      if (.not. (gotin .and. gotout)) go to 70
      if (mod(irun,ngap) .ne. 0) go to 20

      if (design) then
        write(*,103) irun,baseh,runic,runooc
103     format(' Run',i8,'  h',f8.3,' IC RL',f8.0,' OOC RL',f8.0)
        write(11,103) irun,baseh,runic,runooc
        slope = sxy / ssx
        cut = ybar - slope * xbar
        resvar = (ssy - sxy**2 / ssx) / (fn - two)
        comter = four * resvar / ssx
        tarlog = tararl
        ater = slope ** 2 - comter
        bter = two * (slope * (cut - tarlog) + comter * xbar)
        cter = (cut - tarlog) ** 2 - four * resvar * (one / fn +
     1   xbar ** 2 / ssx)
        predh = (tarlog - cut) / slope
        disc = bter ** 2 - four * ater * cter

        if (disc .lt. zero) then
           write(*,101) predh
           write(11,101) predh
           else
           disc = sqrt(disc)
           alow = (-bter - disc) / (two * ater)
           ahi  = (-bter + disc) / (two * ater)
           write(*,101) predh,alow,ahi
           write(11,101) predh,alow,ahi
           endif
101   format(' Estimated h is',f9.3,:,' with CI',2f9.3)
        endif

      if (design) then
        slopeo = sxyo / ssxo
```

```
        cuto = ybaro - slopeo * xbaro
        resvro = (ssyo - sxyo**2 / ssxo) / (fno - two)
        predo = cuto + slopeo * predh
        seo = sqrt(resvro * (one / fno + (predh - xbaro) ** 2 / ssxo))
        write(*,102) predo,predo-two*seo,predo+two*seo
        write(11,102) predo,predo-two*seo,predo+two*seo
 102    format(' OOC ARL is',f9.3,'  with CI',2f9.3/)

        else

        seo = sqrt(ssyo / (fno * (fno - one)))
        write(* ,104) irun,ybaro, ybaro-two*seo, ybaro+2*seo
 104    format(' Run',i7,' OOC ARL is',f9.3,'  with CI',2f9.3/)
        write(11,104) irun,ybaro, ybaro-two*seo, ybaro+2*seo
        endif
 20   continue
      return
      end

      subroutine solve(a,y1,y2,x1,x2,nord,echo)
c
c     Subroutine to solve a system of linear equations for two right sides
c
      implicit double precision (a-h,o-z)
      dimension a(nord,nord+2),x1(nord),x2(nord),y1(nord),y2(nord)
      logical echo
      do 10 i = 1, nord
        a(i,nord+1) = y1(i)
        a(i,nord+2) = y2(i)
        if (echo) write(*,'('' B4'',(8f10.4))') (a(i,j),j=1,nord+2)
 10     continue
      do 20 i = 1, nord-1
        do 30 j = i+1, nord
          pivot = a(j,i) / a(i,i)
          do 30 k = i, nord+2
            a(j,k) = a(j,k) - a(i,k) * pivot
 30       continue
 20     continue
      if (echo)  write(*,*) 'Solution'
      do 40 i = nord,1,-1
        sum1 = a(i,nord+1)
        sum2 = a(i,nord+2)
        do 50 j = i+1, nord
          sum1 = sum1 - a(i,j) * x1(j)
          sum2 = sum2 - a(i,j) * x2(j)
 50       continue
        x1(i) = sum1 / a(i,i)
        x2(i) = sum2 / a(i,i)
 40     continue
      if (echo) then
        write(*,'(1x,7f11.5)') x1
        write(*,*)
        write(*,'(1x,7f11.5)') x2
        endif
      echo = .false.
      return
      end

      subroutine matfac(sigma,factor,nord)
c
```

```
c     Subroutine to find the lower triangular factor of a  symmetric matrix
c
      implicit double precision (a-h,o-z)
      dimension sigma(nord,nord),factor(nord,nord)
      data zero /0.d0/
      factor(1,1) = sqrt(sigma(1,1))
      do 10 i = 2, nord
        do 20 j = 1, i
          sum = zero
          do 30 k = 1, j-1
 30         sum = sum + factor(i,k) * factor(j,k)
        if (j .lt. i) then
          factor(i,j) = (sigma(i,j) - sum) / factor(j,j)
          factor(j,i) = zero
          else
          rutarg = sigma(i,i) - sum
          if (rutarg .le. zero) then
            write(11,'('' Error in MATFAC.  Variable'',i5,'' rutarg'',
     1        g15.7)') i,rutarg
            write(11,*) ' Matrix'
            write(*,'('' Error in MATFAC.  Variable'',i5,'' rutarg'',
     1        g15.7)') i,rutarg
            write(*,*) ' Matrix'
            do 50 k = 1, nord
              write(*,'(10f8.3)') (sigma(k,m),m=1,nord)
 50           write(11,'(10f8.3)') (sigma(k,m),m=1,nord)
            write(*,*) ' Factor so far'
            write(11,*) ' Factor so far'
            do 60 k = 1, i
              write(*,'(10f8.3)') (factor(k,m),m=1,k)
 60           write(11,'(10f8.3)') (factor(k,m),m=1,k)
            rutarg = abs(rutarg)
            endif
          factor(i,i) = sqrt(rutarg)
          endif
 20       continue
 10     continue
      return
      end

      subroutine randn(x,nord,ij,kl)
c
c     Subroutime to produce a vector of pseudorandom N(0,1) values
c     It calls ranmar to get random uniforms
c
      parameter(nuni=200)
      implicit double precision (a-h,o-z)
      dimension x(nord),work(nuni)
      save work,index
      data index/nuni/, quart/0.25d0/, half/0.5d0/, one/1.d0/,
     1 zero /0.d0/, cons/1.71552777d0/, c/0.259d0/, onep/-0.350927217d0/
      do 10 i = 1, nord
 20     index = index + 2
        if (index .gt. nuni-1) then
          call ranmar(work,nuni,ij,kl)
          index = 1
          endif
        u = work(index)
        if (u .eq. zero) then
          u = 1.d-10
```

```
          endif
        v = cons * (work(index+1) - half)
        x(i) = v / u
        z = quart * x(i) ** 2
        if (z .gt. one - u) then
          if (z .gt. c / u - onep) go to 20
          if (z .gt. -log(u)) go to 20
          endif
 10     continue
      return
      end

      subroutine ranmar(rvec, len, ij, kl)
c     this is the random number generator proposed by george marsaglia in
c     florida state university report: fsu-scri-87-50
c     it was slightly modified by f. james to produce an array of
pseudorandom
c     numbers.
      implicit double precision(a-h,o-z)
      dimension  rvec(*), u(97)
      integer i97, j97
      logical test
      save test, i97, j97, c, cd, cm, u
      data test /.false./
      integer ivec

      if( .not. test ) then
c this is the initialization routine for the random number generator
ranmar()
c note: the seed variables can have values between:    0 <= ij <= 31328
c                                                     0 <= kl <= 30081
c the random number sequences created by these two seeds are of sufficient
c length to complete an entire calculation with. for example, if sveral
c different groups are working on different parts of the same calculation,
c each group could be assigned its own ij seed. this would leave each group
c with 30000 choices for the second seed. that is to say, this random
c number generator can create 900 million different subsequences -- with
c each subsequence having a length of approximately 10^30.
c
c use ij = 1802 & kl = 9373 to test the random number generator. the
c subroutine ranmar should be used to generate 20000 random numbers.
c then display the next six random numbers generated multiplied by
4096*4096
c if the random number generator is working properly, the random numbers
c should be:
c           6533892.0  14220222.0  7275067.0
c           6172232.0  8354498.0   10633180.0

        i = mod(ij/177, 177) + 2
        j = mod(ij    , 177) + 2
        k = mod(kl/169, 178) + 1
        l = mod(kl,     169)

        do 2 ii = 1, 97
          s = 0.0
          t = 0.5
          do 3 jj = 1, 24
            m = mod(mod(i*j, 179)*k, 179)
            i = j
            j = k
```

```
            k = m
            l = mod(53*l+1, 169)
            if (mod(l*m, 64) .ge. 32) then
              s = s + t
              endif
            t = 0.5 * t
3           continue
          u(ii) = s
2         continue

        c = 362436.0 / 16777216.0
        cd = 7654321.0 / 16777216.0
        cm = 16777213.0 /16777216.0

        i97 = 97
        j97 = 33

        test = .true.
        endif

      do 100 ivec = 1, len
        uni = u(i97) - u(j97)
        if( uni .lt. 0.0 ) uni = uni + 1.0
        u(i97) = uni
        i97 = i97 - 1
        if(i97 .eq. 0) i97 = 97
        j97 = j97 - 1
        if(j97 .eq. 0) j97 = 97
        c = c - cd
        if( c .lt. 0.0 ) c = c + cm
        uni = uni - c
        if( uni .lt. 0.0 ) uni = uni + 1.0
        rvec(ivec) = uni
100     continue
      return
      end
```