Volume 3 | Issue 1

Article 26

5-1-2004

JMASM10: A Fortran Routine For Sieve Bootstrap Prediction Intervals

Andrés M. Alonso
Universidad Autónoma de Madrid, andres.alonso@uam.es

Journal of Modern Applied Statistical Methods
May, 2004, Vol. 3, No. 1, 239-249

JMASM Algorithms and Code

JMASM10: A Fortran Routine For Sieve Bootstrap Prediction Intervals

Andrés M. Alonso
Department of Mathematics
Universidad Autónoma de Madrid

A Fortran routine for constructing nonparametric prediction intervals for a general class of linear processes is described. The approach uses the sieve bootstrap procedure of Bühlmann (1997) based on residual resampling from an autoregressive approximation to the given process.

Key words: Sieve bootstrap, prediction, time series

Introduction

When studying a time series, one of the goals is the estimation of forecast confidence intervals based on an observed trajectory of the process. The traditional approach of finding prediction intervals for a linear time series assumes that the distribution of the error process is known. Thus, these prediction intervals could be adversely affected by departures from the true underlying distribution.

Some bootstrap approaches have been proposed as a distribution free alternative to compute prediction intervals. Stine (1987) proposed a bootstrap method to estimate the prediction mean squared error of the estimated linear predictor of an AR(p) where p is known. Also, for an AR(p) process with known p, and relaxing the assumptions of Stine (1987), Thombs and Schucany (1990) propose a backward then forward bootstrap method to estimate prediction intervals. Cao *et al.* (1997) study a conditional bootstrap method alternative to Thombs and Schucany's proposal, which is computationally faster. Pascual *et al.* (2001) generalize this conditional bootstrap to ARMA(p, q) processes with known p and q and also include the parameter estimation variability.

Andrés M. Alonso is Assistant Professor of Statistics. Department of Mathematics, Universidad Autónoma de Madrid, 28049 Madrid, Spain. His areas of research interest are statistical computing, resampling methods and biostatistics. E-mail at: andres.alonso@uam.es.

This article describes a bootstrap method to construct nonparametric prediction intervals for a class of linear processes that can be written as a one-sided infinite-order moving average process with at most a polynomial decay of the coefficients $\{\psi_j\}_{j=0}^{+\infty}$. This class includes the stationary and invertible ARMA(p,q) processes. This approach uses the sieve bootstrap of Bühlmann (1997) based on residual resampling from a sequence of approximating autoregressions for $\{X_t\}_{t\in\mathbf{Z}}$ with order $p = p(n)$ that increases as a function of the sample size n.

This sieve bootstrap has a nice nonparametric property, being model-free within the considered class of linear processes. Thus, the proposed bootstrap prediction intervals could be applied to this more general class of linear models without specifying a finite dimensional model as in previous bootstrap proposals. Alonso *et al.* (2002) and (2003) studied the consistency and the finite sample properties of this sieve bootstrap.

Methodology

Let $\{X_t\}_{t\in\mathbf{Z}}$ be a real valued, stationary process with expectation $E[X_t] = \mu_X$ that admits a MA(∞) representation with $\sum_{j=0}^{+\infty}\psi_j^2 < \infty$. Under the additional assumption of invertibility $\{X_t\}_{t\in\mathbf{Z}}$ can be represented as a one-sided infinite-order autoregressive process:

$$\sum_{j=0}^{+\infty} \phi_j (X_{t-j} - \mu_X) = \varepsilon_t, \qquad \phi_0 = 1, t \in \mathbf{Z} \tag{1}$$

with coefficients $\{\phi_j\}_{j=0}^{+\infty}$ satisfying $\sum_{j=0}^{+\infty} \phi_j^2 < \infty$. This AR($\infty$) representation motivates Bühlmann's sieve bootstrap. The method proceeds as follows:

1. Given a sample $\{X_1, X_2, \ldots, X_n\}$, select the order $p = p(n)$ of the autoregressive approximation by AICC criterion: AICC = $-n\log(\sigma^2) + 2(p+1)n/(n-p-2)$, (cf. Section 9.3 of Brockwell &Davis, 1991).

The AICC criterion is a bias-corrected version of AIC (Akaike, 1973), and it has a more extreme penalty for large-order models which counteracts the overfitting nature of AIC. Other order selection criteria (such as BIC) could be used, but AICC is preferred assuming the view that the true model is complex and not of finite dimension, and also because the AICC is asymptotically efficient for autoregressive models, i.e., it chooses an AR model which achieves the optimal rate of convergence of the mean-square prediction error.

2. Construct some estimators of the autoregressive coefficients: $\left(\hat{\phi}_1, \hat{\phi}_2, \ldots, \hat{\phi}_p\right)$. Following Bühlmann (1997) the Yule-Walker estimates are taken.

3. Compute the residuals for $t \in (p+1, p+2, \ldots, n)$ by:

$$\hat{\varepsilon}_t = \sum_{j=0}^{p} \hat{\phi}_j (X_{t-j} - \overline{X}), \qquad \hat{\phi}_0 = 1. \tag{2}$$

4. Define the empirical distribution function of the centered residuals:

$$\hat{F}_{\tilde{\varepsilon}}(x) = (n-p)^{-1} \sum_{t=p+1}^{n} 1\{\tilde{\varepsilon}_t \le x\}, \tag{3}$$

where $\tilde{\varepsilon}_t = \hat{\varepsilon}_t - \hat{\varepsilon}^{(\bullet)}$ and

$$\hat{\varepsilon}^{(\bullet)} = (n-p)^{-1} \sum_{t=p+1}^{n} \hat{\varepsilon}_t .$$

5. Draw a resample ε_t^* of i.i.d. observations from $\hat{F}_{\tilde{\varepsilon}}$.

6. Define X_t^* by the recursion:

$$\sum_{j=0}^{p} \phi_j (X_{t-j}^* - \overline{X}) = \varepsilon_t^*, \tag{4}$$

where the starting p observations are equal to $\overline{X}$.

In practice an AR(p) resample is generated using (4) with sample size equal to n + 100 and then discard the first 100 observations. Up to this step, the resampling plan coincides with the sieve bootstrap, and is valid for bootstrapping some statistics defined as a functional of a m-dimensional distribution function (see details in Section 3.3 of Bühlmann, 1997). However, it is not effective for bootstrap prediction, because it does not replicate the conditional distribution of X_{n+h} given the observed data. But, proceeding as do Cao *et al.* (1997) by fixing the last p observations resamples of the future values can be obtained X_{n+h}^* given $X_{n-p+1}^* = X_{n-p+1}$, $X_{n-p+2}^* = X_{n-p+2}, \ldots, X_n^* = X_n$.

7. Compute the estimation of the autoregressive coefficients: $\left(\hat{\phi}_1^*, \hat{\phi}_2^*, \ldots, \hat{\phi}_p^*\right)$ as in step 1.

8. Compute the future bootstrap observations by the recursion:

$$X_{n+h}^* = \overline{X} - \sum_{j=1}^{p} \hat{\phi}_j^* (X_{t-j}^* - \overline{X}) + \varepsilon_t^*, \tag{5}$$

where $h > 0$, and $X_t^* = X_t$, for $t \le n$.

Finally, $F_{X_{n+h}^*}^*(x)$ the bootstrap distribution of X_{n+h}^* is used to approximate the unknown distribution of X_{n+h} given the observed sample. As usual, a Monte Carlo estimate $\hat{F}_{X_{n+h}^*}^*(x)$ is obtained by repeating the steps 5 to 8 B times. The (1-α)% prediction interval for X_{n+h} is given by $\left[Q^*(\alpha/2), Q^*(1-\alpha/2)\right]$, where $Q^*(.) = \hat{F}_{X_{n+h}^*}^*(.)$ are the quantiles of the estimated bootstrap distribution.

Fortran routines

Module TimeSeriesRoutines

In the module TimeSeriesRoutines are presented some routines required for the sieve bootstrap procedure: subroutine AutoCovarianceVector, subroutine YuleWalker, and subroutine AICCSelection.

```
SUBROUTINE AutoCovarianceVector(
ACVector, XSeries,MaxLag,Positions)
IMPLICIT NONE
REAL (KIND=8), DIMENSION(0:),
INTENT(OUT) :: ACVector
REAL (KIND=8), DIMENSION(:),
INTENT(IN) :: XSeries
INTEGER, INTENT(IN) :: MaxLag
INTEGER, DIMENSION(:), INTENT(IN),
OPTIONAL :: Positions
```

This routine estimates the autocovariances of the XSeries for the orders from 0 to MaxLag. Notice that the implementation allows possible missing observations in the specified Positions. The expression for the autocovariance estimates is given by:

$$\hat{\gamma}_k = \tfrac{1}{n-m}\sum_{t=1}^{n-k} w_t w_{t+k}(X_t - \bar{X})(X_{t+k} - \bar{X}), \tag{6}$$

where m is the number of missing observations, $\bar{X} = (n-m)^{-1}\sum_{t=1}^{n} w_t X_t$ and w_t is equal 0 if the observation t is missing and otherwise is equal to 1.

```
SUBROUTINE YuleWalker(XSeries,
ACMatrix,YWPhi,Residuals)
USE Msimsl
USE Imslf90
IMPLICIT NONE
REAL (KIND=8), DIMENSION(:),
INTENT(IN) :: XSeries
REAL (KIND=8), DIMENSION(:,:),
INTENT(IN) :: ACMatrix
REAL (KIND=8), DIMENSION(:),
INTENT(OUT) :: YWPhi
REAL (KIND=8), DIMENSION(:),
INTENT(OUT) :: Residuals
```

This routine calculates the Yule-Walker estimates of the autoregressive coefficient required in the steps 2 and 7 of sieve bootstrap procedure. It also calculates the residuals for the estimated model. The Yule-Walker estimators can be obtained from the following relation (cf. Section 8.1 of Brockwell and Davis (1991)):

$$\hat{\boldsymbol{\Gamma}}_p \hat{\boldsymbol{\varphi}}_{\boldsymbol{p}} = \hat{\boldsymbol{\gamma}}_{\boldsymbol{p}}, \tag{7}$$

where $\hat{\boldsymbol{\Gamma}}_p$ is the estimated autocovariance matrix $\left[\hat{\gamma}_{i-j}\right]_{i,j=1}^{p}$, $\hat{\boldsymbol{\gamma}}_{\boldsymbol{p}} = (\hat{\gamma}_1, \hat{\gamma}_2, \ldots, \hat{\gamma}_p)'$ and $\hat{\boldsymbol{\varphi}}_{\boldsymbol{p}} = (\hat{\phi}_1, \hat{\phi}_2, \ldots, \hat{\phi}_p)'$ is the coefficients vector. Using (2), the estimated residuals were obtained.

```
SUBROUTINE AICCSelection(XSeries,
ACVector,PMax,PHat)
IMPLICIT NONE
REAL (KIND=8), DIMENSION(:),
INTENT(IN) :: XSeries
REAL (KIND=8), DIMENSION(0:),
INTENT(IN) :: ACVector
INTEGER, INTENT(IN) :: PMax
INTEGER, INTENT(OUT) :: PHat
```

This routine implements the AICC method for selecting the order of the autoregressive model for XSeries. It considers models from $p = 0$ to $p =$ PMax. Instead of using the subroutine YuleWalker for the different values of p, it uses the Durbin-Levinson algorithm (cf. Section 8.2 of Brockwell and Davis (1991)) which avoids the matrix inversion required in the direct computation of $\hat{\boldsymbol{\varphi}}_{\boldsymbol{p}}$. The Durbin-Levinson algorithm uses the following recursions:

$$\hat{\phi}_{m,m} = (\hat{\gamma}_m - \sum_{j=1}^{m-1}\hat{\phi}_{m-1,j}\hat{\gamma}_{m-j}) / \hat{v}_{m-1}, \tag{8}$$

$$\begin{bmatrix} \hat{\phi}_{m,1} \\ \hat{\phi}_{m,2} \\ \vdots \\ \hat{\phi}_{m,m-1} \end{bmatrix} = \begin{bmatrix} \hat{\phi}_{m-1,1} \\ \hat{\phi}_{m-1,2} \\ \vdots \\ \hat{\phi}_{m-1,m-1} \end{bmatrix} - \hat{\phi}_{m,m} \begin{bmatrix} \hat{\phi}_{m-1,m-1} \\ \hat{\phi}_{m-1,m-2} \\ \vdots \\ \hat{\phi}_{m-1,1} \end{bmatrix}, \tag{9}$$

and

$$\hat{v}_m = \hat{v}_{m-1}(1 - \hat{\phi}_{m,m}^2), \tag{10}$$

with the following initial values: $\hat{\phi}_{1,1} = \hat{\gamma}_1 / \hat{\gamma}_0$ and $\hat{v}_1 = \hat{\gamma}_0(1-\hat{\phi}_{1,1}^2)$.

Notice that the subroutine AICCSelection can be easily modified in order to use other information criterion as AIC or BIC. Only the two following sentences required some minor changes:

```
MinimumAIC =
RXSize*LOG(ACVector(0)) +
2.0D0*(REAL(I,KIND=8)+1.0D0)
*RXSize/(RXSize - REAL(I+2,KIND=8))

WorkAIC =
RXSize*LOG(VarianceVector(I))
+2.0D0*(REAL(I,KIND=8)+1.0D0)
*RXSize/(RXSize - REAL(I+2,KIND=8))
```

Routine FESieves

Here are described the subroutine FESieves which implements the steps 2 to 8 of the sieve bootstrap procedure. Notice that the step 1 is implemented by subroutine AICCSelection.

```
SUBROUTINE
FESieves(EDF,XSeries,PHat)
USE Msimsl
USE Imslf90
USE TimeSeries
IMPLICIT NONE
REAL (KIND=8), DIMENSION (:,:),
INTENT(OUT) :: EDF
REAL (KIND=8), DIMENSION (:),
INTENT(IN) :: XSeries
INTEGER, INTENT(IN) :: PHat
```

The inputs of subroutine FESieves are: the sample XSeries = $\{X_1, X_2, \ldots, X_n\}$ and the selected order, PHat. The output is a MaxLag × *B* matrix, where MaxLag is the maximum prediction horizon to be considered and *B* is the number of resamples.

Step 2 and 7 are implemented by the following sentences:

```
CALL YuleWalker(XSeries,
ACMatrix(1:PHat+1, 1:PHat+1),
YWPhi, Residuals)
```

```
CALL YuleWalker(WSeries(101:XSize +
100), WACMatrix(1:PHat+1,
1:PHat+1), WYWPhi, YWResiduals)
```

where the WSeries are the resample obtained using recursion (4). The estimates YWPhi are used in recursion (4) and the bootstrap estimates WYWPhi are used in recursion (5). Also, in the first call to subroutine YuleWalker, the step 3 is performed. As mentioned in the previous section, a bootstrap resample was generated using (4) with sample size equal to XSize+100 and then discard the first 100 observations by WSeries(101:XSize + 100).

The resamples of step 5 are obtained by sampling with replacement from the vector of centered residuals, WResiduals = WResiduals - SUM(WResiduals) / REAL(XSize - PHat, KIND=8):

```
DO I = 1, XSize + 100 + MaxLag ! a
resample of centered residual
   CALL RNUND(1, XSize-PHat,
        RandomIndex)
   RResiduals(I)=
        WResiduals(RandomIndex)
END DO
```

Because recursions (4) and (5) are similar, here, it is only described the prediction recursion:

```
DO I = XSize+101, XSize+100+MaxLag
   WSeries(I) = RResiduals(I)
   DO Ip = 1, PHat
      WSeries(I) = WSeries(I) +
           WYWPhi(Ip)*WSeries(I-Ip)
   END DO
END DO
EDF(1:MaxLag,J)=WSeries(XSize+101:
XSize+100+MaxLag) + XMean
```

Table 1. Simulation results for Model 1.

Lag	Sample size	Method	Coverage (se)	Cov. (below /above)	Length (se)
h	*n*	Theoretical	95%	2.50% / 2.50%	3.92
1	25	Bootstrap	89.03 (0.82)	4.44 / 6.53	3.74 (0.07)
	50		92.59 (0.52)	4.25 / 3.16	3.86 (0.05)
	100		93.77 (0.33)	3.25 / 2.98	3.90 (0.04)
h	*n*	Theoretical	95%	2.50% / 2.50%	4.92
3	25	Bootstrap	87.50 (0.86)	5.41 / 7.09	4.30 (0.08)
	50		92.08 (0.49)	3.97 / 3.95	4.69 (0.05)
	100		93.21 (0.38)	3.53 / 3.26	4.77 (0.05)

Table 2. Simulation results for Model 2.

Lag	Sample size	Method	Coverage (se)	Cov. (below /above)	Length (se)
h	*n*	Theoretical	95%	2.50% / 2.50%	3.93
1	25	Bootstrap	89.53 (0.85)	5.72 / 4.75	4.12 (0.08)
	50		92.06 (0.62)	3.63 / 4.31	3.98 (0.06)
	100		93.31 (0.43)	3.49 / 3.20	3.96 (0.04)
h	*n*	Theoretical	95%	2.50% / 2.50%	4.93
3	25	Bootstrap	89.19 (0.79)	5.15 / 5.66	4.52 (0.09)
	50		91.50 (0.58)	3.85 / 4.65	4.62 (0.06)
	100		92.49 (0.39)	3.19 / 4.32	4.68 (0.05)

Table 3. Simulation results for Model 3.

Lag	Sample size	Method	Coverage (se)	Cov. (below /above)	Length (se)
h	*n*	Theoretical	95%	2.50% / 2.50%	3.79
1	25	Bootstrap	89.45 (0.66)	4.73 / 5.82	3.54 (0.06)
	50		92.44 (0.45)	4.19 / 3.37	3.62 (0.04)
	100		93.77 (0.36)	3.38 / 2.85	3.74 (0.04)
h	*n*	Theoretical	95%	2.50% / 2.50%	3.93
3	25	Bootstrap	89.20 (0.65)	4.90 / 5.90	3.58 (0.06)
	50		92.79 (0.39)	3.68 / 3.53	3.75 (0.05)
	100		93.84 (0.34)	3.03 / 3.13	3.88 (0.04)

Notice that in (5) the bootstrap autoregressive coefficient is used, WYWPhi**,** this allows us to incorporate the parameter estimation variability in the prediction intervals.

Finally, the $\alpha/2$ and $(1-\alpha/2)$ quantiles of the empirical density of forecasts, EDF, constitutes the prediction interval.

Results

In this section are briefly described the results of a simulation experiment using the Fortran subroutine presented in the previous section. The following models are used:

- Model 1: $X_t = 0.75\ X_{t-1} - 0.5\ X_{t-2} + \varepsilon_t$, where ε_t are i.i.d. N(0,1).
- Model 2: $X_t = \varepsilon_t - 0.3\ \varepsilon_{t-1} + 0.7\ \varepsilon_{t-2}$, where ε_t are i.i.d. N(0,1).

- Model 3: X_t is a Gaussian process with autocovariance generating function equal to $G(z) = \sum_{k=-\infty}^{+\infty} \gamma_k z^k$, where $\gamma_k = (|k|+1)^{-3}$.

The autoregressive model was considered by Cao *et al.* (1997), the moving average model 2 by Pascual *et al.* (2001) and the model 3 by Alonso *et al.* (2002). Notice that neither model 2 nor model 3 admit a finite AR representation. Moreover, model 3 does not have an ARMA representation.

To evaluate the prediction intervals, their mean coverage and length are used and the proportions of observations lying out to the left and to the right of the interval. These quantities are estimated as follows:

a) For a combination of model, sample size and error distribution, simulate a series, and generate $R = 1000$ future values X_{n+h}.

b) For the bootstrap procedure obtain the (1-α) prediction interval based on $B = 1000$ bootstrap resamples.

c) The coverage is estimated as $C = \#\{Q^*(\alpha/2) \le X_{n+h}^r \le Q^*(1-\alpha/2)\}/R$, where X_{n+h}^r with $r = 1,2,\ldots,R$ are the R future values generated in step a).

In steps a) and b) the "theoretical" and bootstrap interval lengths are obtained using

$$L_T = X_{n+h}^{\lceil R(1-\alpha/2)\rceil} - X_{n+h}^{\lceil R\alpha/2\rceil}$$

and

$$L_B = Q^*(1-\alpha/2) - Q^*(\alpha/2),$$

respectively. Finally, the steps a) – c) are repeated 100 times.

The results are presented in Tables 1 – 3, using three sample sizes n = 25, 50 and 100, nominal coverage 95% and the prediction lag h = 1 and 3. Essentially, similar results are obtained in all cases. Sieve bootstrap performs reasonably well in all considered models since the mean coverage and length tend to the nominal values as the sample size grows. Notice that for models 2 and 3 the sieve bootstrap never uses the correct model. The running time for these three experiments (using a Pentium 4, running at 2.66GHz) was 22.92, 24.40 and 27.82 seconds, respectively.

Conclusion

It has been shown by Alonso *et al.* (2002) and (2003) that, for general linear process, if an AR approximation that grows with the sample size is used, it can derive a bootstrap for building prediction intervals that has the two following properties: first, the procedure is consistent, that is, it generates as prediction a random variable that converges in conditional distribution to the concerning variable; second, Monte Carlo simulations show that the proposed procedure provides better coverage results than previous methods in general cases. This article describes a Fortran routine that implement this sieve bootstrap prediction procedure. Additional simulation experiments confirm the correct behavior of the proposed procedure in finite samples.

References

Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle, In: *2nd International Symposium on Information Theory*, B.N. Petrov, and F. Csaki, Eds., Akademiai Kiado, Budapest, 267-281.

Alonso, A. M., Romo, J. & Peña, D. (2002). Forecasting time series with sieve bootstrap. *Journal of Statistical Planning and Inference*, *100*, 1-11.

Alonso, A. M., Romo, J. & Peña, D. (2003). On sieve bootstrap prediction intervals. *Statistics and Probability Letters*, *65*, 13-20.

Brockwell, P. J. & Davis, R. A. (1991). *Time Series: Theory and Methods*, Springer-Verlag, New York.

Bühlmann, P. (1997). Sieve bootstrap for time series, *Bernoulli*, *3*, 123-148.

Cao, R., Febrero-Bande, M., González-Manteiga, W., Prada-Sánchez, J. M. & García-Jurado, I. (1997). Saving computer time in constructing consistent bootstrap prediction intervals for autoregressive processes, *Communications in Statistics. Theory and Methods*, *26*, 961-978.

Pascual, L., Romo, J. & Ruiz, E. (2001). Effects of parameter estimation on prediction densities: a bootstrap approach, *International Journal of Forecasting*, *17*, 83-103.

Stine, R. A. (1987). Estimating properties of autoregressive forecasts, *Journal of the American Statistical Association*, *82*, 1072-1078.

Thombs, L. A. & Schucany, W. R. (1990). Bootstrap prediction intervals for autoregression, *Journal of the American Statistical Association*, *85*, 486-492

Appendix I – Module TimeSeriesRoutines

```
MODULE TimeSeriesRoutines
 REAL (KIND=8), DIMENSION(:), ALLOCATABLE :: ZSeries
 INTEGER :: p, d, q, ps, ds, qs, season

CONTAINS

SUBROUTINE AutoCovarianceVector(ACVector,XSeries,MaxLag,Positions)
 IMPLICIT NONE
 REAL (KIND=8), DIMENSION(0:), INTENT(OUT) :: ACVector
 REAL (KIND=8), DIMENSION(:), INTENT(IN) :: XSeries
 INTEGER, INTENT(IN) :: MaxLag
 INTEGER, DIMENSION(:), INTENT(IN), OPTIONAL :: Positions
 ! Local variables
 INTEGER :: K, I, J, XSize, NMissings
 REAL (KIND=8) :: RXSize, XMean
 REAL (KIND=8), DIMENSION(:), ALLOCATABLE :: Weights

 ! First executable statement

 XSize = SIZE(XSeries, 1)
 ALLOCATE(Weights(XSize))
 Weights = 1.0D0

 IF (PRESENT(Positions)) THEN
      Weights(Positions) = 0.0D0
      NMissings = SIZE(Positions, 1)
      RXSize = REAL(XSize - NMissings, KIND=8)
 ELSE
      RXSize = REAL(XSize, KIND=8)
 END IF

 XMean = SUM(XSeries*Weights)/RXSize
 DO K = 0, MaxLag
      ACVector(K) = DOT_PRODUCT(&
       (XSeries((K+1):XSize) - XMean)*Weights((K+1):XSize), &
       (XSeries(1:(XSize-K)) - XMean)*Weights(1:(XSize-K)))/RXSize
 END DO

 DEALLOCATE(Weights)
END SUBROUTINE AutoCovarianceVector

SUBROUTINE AutoCovarianceMatrix(ACMatrix,XSeries,MaxLag,MSize)
 IMPLICIT NONE
 REAL (KIND=8), DIMENSION(:,:), INTENT(OUT) :: ACMatrix
 REAL (KIND=8), DIMENSION(:), INTENT(IN) :: XSeries
 INTEGER, INTENT(IN) :: MaxLag, MSize
! Local variables
 INTEGER :: K, I, J, XSize
 REAL (KIND=8) :: RXSize, XMean
 REAL (KIND=8), DIMENSION(0:MSize) :: ACVector

 ! First executable statement

 XSize = SIZE(XSeries, 1)
 RXSize = REAL(XSize, KIND=8)
 XMean = SUM(XSeries)/RXSize
 DO K = 0, MaxLag+1
      ACVector(K) = DOT_PRODUCT(XSeries((K+1):XSize) - XMean, &
            XSeries(1:(XSize-K)) - XMean)/RXSize
 END DO
 DO I = 1, MaxLag+1
      DO J = 1, MaxLag+1
            ACMatrix(I,J) = ACVector(ABS(I-J))
      END DO
 END DO
END SUBROUTINE AutoCovarianceMatrix

SUBROUTINE YuleWalker(XSeries,ACMatrix,YWPhi,Residuals)
 USE Msimsl
 USE Imslf90
 IMPLICIT NONE
 REAL (KIND=8), DIMENSION(:), INTENT(IN) :: XSeries
 REAL (KIND=8), DIMENSION(:,:), INTENT(IN) :: ACMatrix
 REAL (KIND=8), DIMENSION(:), INTENT(OUT) :: YWPhi
 REAL (KIND=8), DIMENSION(:), INTENT(OUT) :: Residuals
 ! Local variables
 INTEGER :: MSize, XSize, I, J
 REAL (KIND=8), DIMENSION(:,:), ALLOCATABLE :: A
 REAL (KIND=8), DIMENSION(:), ALLOCATABLE :: B
 INTEGER :: M, N, IERR, IOPT, IA, IB

 ! First executable statement

 MSize = SIZE(ACMatrix, 1)
 XSize = SIZE(XSeries)

 ! Initializing LSLDS variables
 ALLOCATE(A(MSize-1, MSize-1), B(MSize-1))
 A = ACMatrix(1:(MSize-1), 1:(MSize-1))
 B = ACMatrix(2:MSize, 1)
 M = MSize-1

 ! Solving the Yule-Walker equations
 CALL DLSLDS (M, A, M, B, YWPhi)

 ! Calculating the YW residuals
 Residuals = 0
 DO I = (MSize+1), XSize
      Residuals(I) = XSeries(I)
      DO J = 1, MSize-1
            Residuals(I) = Residuals(I) - YWPhi(J)*XSeries(I-J)
      END DO
 END DO

 DEALLOCATE(A, B)
END SUBROUTINE YuleWalker
```

```
SUBROUTINE AICCSelection(XSeries,ACVector,PMax,PHat)
 IMPLICIT NONE
 REAL (KIND=8), DIMENSION(:), INTENT(IN) :: XSeries
 REAL (KIND=8), DIMENSION(0:), INTENT(IN) :: ACVector
 INTEGER, INTENT(IN) :: PMax
 INTEGER, INTENT(OUT) :: PHat
 ! Local variables
 REAL (KIND=8), DIMENSION(:), ALLOCATABLE :: VarianceVector
 REAL (KIND=8), DIMENSION(:,:), ALLOCATABLE :: PPhi
 REAL (KIND=8) :: VWork, WorkAIC, MinimumAIC, RXSize
 INTEGER :: XSize, WorkP, I, J

 ! First executable statement

 ALLOCATE(VarianceVector(PMax))
 ALLOCATE(PPhi(PMax, PMax))
 XSize = SIZE(XSeries)
 RXSize = REAL(XSize, KIND=8)

 ! Durbin-Levinson Algorithm

 PPhi = 0.0D0
 PPhi(1, 1) = ACVector(1)/ACVector(0)
 VarianceVector(1) = ACVector(0)*(1.0D0 - PPhi(1, 1)**2)

 DO I = 2, PMax
      VWork = 0
      DO J = 1, I-1
            VWork = VWork + PPhi(I-1, J)*ACVector(I-J)
      ENDDO
      PPhi(I, I) = (ACVector(I) - VWork)/VarianceVector(I-1)
      DO J = 1, I-1
            PPhi(I, J) = PPhi(I-1, J) - PPhi(I, I)*PPhi(I-1, I-J)
      ENDDO
      VarianceVector(I) = VarianceVector(I-1)*(1.0D0 - PPhi(I, I)**2)
 ENDDO

 I = 0
 MinimumAIC = RXSize*LOG(ACVector(0))+2.0D0*(REAL(I, KIND=8)+1.0D0)* &
      RXSize/(RXSize - REAL(I+2, KIND=8))
 WorkP = 0
 DO I = 1, PMax
      WorkAIC = RXSize*LOG(VarianceVector(I))+2.0*(REAL(I, KIND=8) &
             +1.0)*RXSize/(RXSize - REAL(I+2, KIND=8))
      IF (WorkAIC < MinimumAIC) THEN
            MinimumAIC = WorkAIC
            WorkP = I
      END IF
 END DO
 PHat = WorkP

 DEALLOCATE(PPhi, VarianceVector)
END SUBROUTINE AICCSelection

END MODULE TimeSeriesRoutines
```

Appendix II – Routine FESieves

```
SUBROUTINE FESieves(EDF, XSeries, PHat)
 USE Msimsl
 USE Imslf90
 USE TimeSeries
 IMPLICIT NONE
 REAL (KIND=8), DIMENSION (:,:), INTENT(OUT) :: EDF
 REAL (KIND=8), DIMENSION (:), INTENT(IN) :: XSeries
 INTEGER, INTENT(IN) :: PHat
 ! Local variables
 INTEGER :: XSize, MaxLag, B
 REAL (KIND=8), DIMENSION(:), ALLOCATABLE :: WSeries
 REAL (KIND=8), DIMENSION(:,:), ALLOCATABLE :: ACMatrix
 REAL (KIND=8), DIMENSION(:), ALLOCATABLE :: Residuals
 REAL (KIND=8), DIMENSION(:), ALLOCATABLE :: YWPhi
 REAL (KIND=8) :: XMean

 ! First executable statement

 XSize = SIZE(XSeries, 1)
 MaxLag = SIZE(EDF, 1)
 B = SIZE(EDF, 2)
 ALLOCATE(WSeries(XSize), Residuals(XSize))
 ALLOCATE(ACMatrix(PHat+1, PHat+1), YWPhi(PHat))
 XMean = SUM(XSeries)/REAL(XSize, KIND=8)
 WSeries = XSeries – Xmean

 ! Steps 2 – 3
 CALL AutoCovarianceMatrix(ACMatrix(1:PHat+1, 1:PHat+1), WSeries, &
      PHat, PHat+1)
 CALL YuleWalker(WSeries, ACMatrix(1:PHat+1, 1:PHat+1), YWPhi, &
      Residuals)

 ! Steps 4 – 8
 CALL ESievesBootstrap(EDF,XSeries,YWPhi, Residuals, PHat, MaxLag, B)

 DEALLOCATE(ACMatrix, YWPhi, WSeries, Residuals)

CONTAINS

SUBROUTINE ESievesBootstrap(EDF,XSeries,YWPhi,Residuals,PHat,MaxLag,B)
 USE Msimsl
 USE Imslf90
 USE TimeSeries
 IMPLICIT NONE
 REAL (KIND=8), DIMENSION(:,:), INTENT(OUT) :: EDF
 REAL (KIND=8), DIMENSION(:), INTENT(IN) :: XSeries
 REAL (KIND=8), DIMENSION(:), INTENT(IN) :: YWPhi
 REAL (KIND=8), DIMENSION(:), INTENT(IN) :: Residuals
 INTEGER, INTENT(IN) :: PHat, MaxLag, B
 ! Local variables
 INTEGER :: XSize, I, J, Ip, RandomIndex, NOUT, ISEED
 REAL (KIND=8), DIMENSION(:), ALLOCATABLE :: WResiduals, RResiduals
 REAL (KIND=8), DIMENSION(:), ALLOCATABLE :: WSeries, WYWPhi
 REAL (KIND=8), DIMENSION(:,:), ALLOCATABLE :: WACMatrix
 REAL (KIND=8), DIMENSION(:), ALLOCATABLE :: YWResiduals
 REAL (KIND=8) :: XMean

 ! First executable statement

 XSize = SIZE(XSeries, 1)

 ALLOCATE(WSeries(XSize+100+MaxLag))
 XMean = SUM(XSeries)/REAL(XSize, KIND=8)
 WSeries(1:XSize) = XSeries - XMean

 ALLOCATE(WResiduals(XSize - PHat))
 WResiduals = Residuals(PHat+1:XSize)
 WResiduals = WResiduals - SUM(WResiduals)/REAL(XSize - PHat, KIND=8)

 ALLOCATE(RResiduals(XSize+100+MaxLag), WYWPhi(PHat), &
      WACMatrix(PHat+1, PHat+1), YWResiduals(XSize))

 CALL UMACH (2, NOUT)
 CALL RNGET (ISEED)
 CALL RNSET (ISEED)

 DO J = 1, B
      ! Steps 4 – 5
      DO I = 1, XSize+100+MaxLag
            CALL RNUND(1, XSize - PHat, RandomIndex)
            RResiduals(I) = WResiduals(RandomIndex)
      END DO

      ! Step 6
      WSeries = RResiduals
      DO I = PHat+1, XSize+100
            DO Ip = 1, PHat
                  WSeries(I) = WSeries(I) + YWPhi(Ip)*WSeries(I-Ip)
            END DO
      END DO

      ! Step 7
      CALL AutoCovarianceMatrix(WACMatrix(1:PHat+1, 1:PHat+1), &
            WSeries(101:XSize+100), PHat, PHat+1)
      CALL YuleWalker(WSeries(101:XSize + 100), &
            WACMatrix(1:PHat+1, 1:PHat+1), WYWPhi, YWResiduals)

      ! Prediction. Step 8
      WSeries(101:XSize+100) = XSeries - XMean
      DO I = XSize+101, XSize+100+MaxLag
            WSeries(I) = RResiduals(I)
            DO Ip = 1, PHat
                  WSeries(I) = WSeries(I) + WYWPhi(Ip)*WSeries(I-Ip)
            END DO
      END DO
      EDF(1:MaxLag, J) = WSeries(XSize+101:XSize+100+MaxLag) + XMean
 END DO
 DEALLOCATE(WSeries, Residuals, RResiduals, WYWPhi, YWResiduals, &
      WACMatrix)
END SUBROUTINE ESievesBootstrap
END SUBROUTINE FESieves
```