Package: ltsa (via r-universe)

September 8, 2024

Version 1.4.6

Date 2015-12-20

Title Linear Time Series Analysis

Author A.I. McLeod, Hao Yu, Zinovi Krougly

Maintainer A.I. McLeod <aimcleod@uwo.ca>

Depends R (>= 2.1.0)

Description Methods of developing linear time series modelling. Methods are given for loglikelihood computation, forecasting and simulation.

Classification/ACM G.3, G.4, I.5.1

Classification/MSC 62M10, 91B84

License GPL (>= 2)

URL http://www.stats.uwo.ca/faculty/aim

NeedsCompilation yes

Date/Publication 2015-12-21 08:55:04

Repository https://angusian.r-universe.dev

RemoteUrl https://github.com/cran/ltsa

RemoteRef HEAD

RemoteSha 0a03bbb7cf19e479dc77592ed09621eeb8afb470

Contents

`ltsa-package` *Linear Time Series Analysis*

Description

Linear time series modelling. Methods are given for loglikelihood computation, forecasting and simulation.

Details

Package:	ltsa
Type:	Package
Version:	1.4.5
Date:	2015-08-22
License:	GPL (>= 2)

FUNCTION	SUMMARY
DHSimulate	Davies and Harte algorithm for time series simulation
DLAcfToAR	from Acf to AR using Durbin-Levinson recursion
DLLoglikelihood	exact loglikelihood using Durbin-Levinson algorithm
DLResiduals	exact one-step residuals, Durbin-Levision algorithm
DLSimulate	exact simulation of Gaussian time series using DL
is.toeplitz	test for Toeplitz matrix
PredictionVariance	two methods provided
tacvfARMA	theoretical autocovariances
ToeplitzInverseUpdate	update inverse
TrenchForecast	general algorithm for forecasting
TrenchInverse	efficient algorithm for inverse of Toeplitz matrix
TrenchLogLikelihood	exact loglikelihood
TrenchMean	exact MLE for mean

Author(s)

A. I. McLeod, Hao Yu and Zinovi Krougly.

Maintainer: aimcleod@uwo.ca

References

Hipel, K.W. and McLeod, A.I., (2005). Time Series Modelling of Water Resources and Environmental Systems. Electronic reprint of our book orginally published in 1994. http://www.stats.uwo.ca/faculty/aim/1994Book/.

McLeod, A.I., Yu, Hao, Krougly, Zinovi L. (2007). Algorithms for Linear Time Series Analysis, Journal of Statistical Software.

See Also

DHSimulate, DLAcfToAR, DLLoglikelihood, DLResiduals, DLSimulate, exactLoglikelihood, is.toeplitz, PredictionVariance, tacvfARMA, ToeplitzInverseUpdate, TrenchForecast, TrenchInverse, TrenchLoglikelihood, TrenchMean,

Examples

```
#Example 1: DHSimulate
#First define acf for fractionally-differenced white noise and then simulate using DHSimulate
`tacvfFdwn` <-
function(d, maxlag)
{
    x <- numeric(maxlag + 1)
    x[1] <- gamma(1 - 2 * d)/gamma(1 - d)^2
    for(i in 1:maxlag)
        x[i + 1] <- ((i - 1 + d)/(i - d)) * x[i]
    x
}
n<-1000
rZ<-tacvfFdwn(0.25, n-1) #length 1000
Z<-DHSimulate(n, rZ)
acf(Z)

#Example 2: DLAcfToAR
#
n<-10
d<-0.4
r<-tacvfFdwn(d, n)
r<-(r/r[1])[-1]
HoskingPacf<-d/(-d+(1:n))
cbind(DLAcfToAR(r),HoskingPacf)

#Example 3: DLLoglikelihood
#Using Z and rZ in Example 1.
DLLoglikelihood(rZ, Z)

#Example 4: DLResiduals
```

```
#Using Z and rZ in Example 1.
DLResiduals(rZ, Z)

#Example 5: DLSimulate
#Using Z in Example 1.
z<-DLSimulate(n, rZ)
plot.ts(z)

#Example 6: is.toeplitz
is.toeplitz(toeplitz(1:5))

#Example 7: PredictionVariance
#Compare with predict.Arima
#general script, just change z, p, q, ML
z<-sqrt(sunspot.year)
n<-length(z)
p<-9
q<-0
ML<-10
#for different data/model just reset above
out<-arima(z, order=c(p,0,q))
sda<-as.vector(predict(out, n.ahead=ML)$se)
#
phi<-theta<-numeric(0)
if (p>0) phi<-coef(out)[1:p]
if (q>0) theta<-coef(out)[(p+1):(p+q)]
zm<-coef(out)[p+q+1]
sigma2<-out$sigma2
r<-sigma2*tacvfARMA(phi, theta, maxLag=n+ML-1)
sdb<-sqrt(PredictionVariance(r, maxLead=ML))
cbind(sda,sdb)

#Example 8: tacfARMA
#There are two methods: tacvfARMA and ARMAacf.
#tacvfARMA is more general since it computes the autocovariances function
# given the ARMA parameters and the innovation variance whereas ARMAacf
# only computes the autocorrelations. Sometimes tacvfARMA is more suitable
# for what is needed and provides a better result than ARMAacf as in the
# the following example.
#
#general script, just change z, p, q, ML
z<-sqrt(sunspot.year)
n<-length(z)
p<-9
q<-0
ML<-5
#for different data/model just reset above
out<-arima(z, order=c(p,0,q))
phi<-theta<-numeric(0)
if (p>0) phi<-coef(out)[1:p]
if (q>0) theta<-coef(out)[(p+1):(p+q)]
zm<-coef(out)[p+q+1]
sigma2<-out$sigma2
```

```
rA<-tacvfARMA(phi, theta, maxLag=n+ML-1, sigma2=sigma2)
rB<-var(z)*ARMAacf(ar=phi, ma=theta, lag.max=n+ML-1)
#rA and rB are slighly different
cbind(rA[1:5],rB[1:5])


#Example 9: ToeplitzInverseUpdate
#In this example we compute the update inverse directly and using ToeplitzInverseUpdate and
#compare the result.
phi<-0.8
sde<-30
n<-30
r<-arima.sim(n=30,list(ar=phi),sd=sde)
r<-phi^(0:(n-1))/(1-phi^2)*sde^2
n1<-25
G<-toeplitz(r[1:n1])
GI<-solve(G) #could also use TrenchInverse
GIupdate<-ToeplitzInverseUpdate(GI,r[1:n1],r[n1+1])
GIdirect<-solve(toeplitz(r[1:(n1+1)]))
ERR<-sum(abs(GIupdate-GIdirect))
ERR


#Example 10: TrenchForecast
#Compare TrenchForecast and predict.Arima
#general script, just change z, p, q, ML
z<-sqrt(sunspot.year)
n<-length(z)
p<-9
q<-0
ML<-10
#for different data/model just reset above
out<-arima(z, order=c(p,0,q))
Fp<-predict(out, n.ahead=ML)
phi<-theta<-numeric(0)
if (p>0) phi<-coef(out)[1:p]
if (q>0) theta<-coef(out)[(p+1):(p+q)]
zm<-coef(out)[p+q+1]
sigma2<-out$sigma2
#r<-var(z)*ARMAacf(ar=phi, ma=theta, lag.max=n+ML-1)
#When r is computed as above, it is not identical to below
r<-sigma2*tacvfARMA(phi, theta, maxLag=n+ML-1)
F<-TrenchForecast(z, r, zm, n, maxLead=ML)
#the forecasts are identical using tacvfARMA
#


#Example 11: TrenchInverse
#invert a matrix of order n and compute the maximum absolute error
# in the product of this inverse with the original matrix
n<-5
r<-0.8^(0:(n-1))
G<-toeplitz(r)
```

```
Gi<-TrenchInverse(G)
GGi<-crossprod(t(G),Gi)
id<-matrix(0, nrow=n, ncol=n)
diag(id)<-1
err<-max(abs(id-GGi))
err


#Example 12: TrenchLoglikelihood
#simulate a time series and compute the concentrated loglikelihood using DLLoglikelihood and
#compare this with the value given by TrenchLoglikelihood.
phi<-0.8
n<-200
r<-phi^(0:(n-1))
z<-arima.sim(model=list(ar=phi), n=n)
LD<-DLLoglikelihood(r,z)
LT<-TrenchLoglikelihood(r,z)
ans<-c(LD,LT)
names(ans)<-c("DLLoglikelihood","TrenchLoglikelihood")

#Example 13: TrenchMean
phi<- -0.9
a<-rnorm(100)
z<-numeric(length(a))
phi<- -0.9
n<-100
a<-rnorm(n)
z<-numeric(n)
mu<-100
sig<-10
z[1]<-a[1]*sig/sqrt(1-phi^2)
for (i in 2:n)
 z[i]<-phi*z[i-1]+a[i]*sig
z<-z+mu
r<-phi^(0:(n-1))
meanMLE<-TrenchMean(r,z)
meanBLUE<-mean(z)
ans<-c(meanMLE, meanBLUE)
names(ans)<-c("BLUE", "MLE")
ans
```

DHSimulate *Simulate General Linear Process*

Description

Uses the Davies-Harte algorithm to simulate a Gaussian time series with specified autocovariance function.

Usage

```
DHSimulate(n, r, ReportTestOnly = FALSE, rand.gen = rnorm, ...)
```

Arguments

n	length of time series to be generated
r	autocovariances at lags 0,1,...
ReportTestOnly	FALSE – Run normally so terminates with an error if Davies-Harte condition does not hold. Othewise if TRUE, then output is TRUE if the Davies-Harte condition holds and FALSE if it does not.
rand.gen	random number generator to use. It is assumed to have mean zero and variance one.
...	optional arguments passed to rand.gen

Details

The method uses the FFT and so is most efficient if the series length, n, is a power of 2. The method requires that a complicated non-negativity condition be satisfed. Craigmile (2003) discusses this condition in more detail and shows for anti-persistent time series this condition will always be satisfied. Sometimes, as in the case of fractinally differenced white noise with parameter d=0.45 and n=5000, this condition fails and the algorithm doesn't work. In this case, an error message is generated and the function halts.

Value

Either a vector of length containing the simulated time series if Davies-Harte condition holds and ReportTestOnly = FALSE. If argument ReportTestOnly is set to TRUE, then output is logical variable indicating if Davies-Harte condition holds, TRUE, or if it does not, FALSE.

Author(s)

A.I. McLeod

References

Craigmile, P.F. (2003). Simulating a class of stationary Gaussian processes using the Davies-Harte algorithm, with application to long memory processes. Journal of Time Series Analysis, 24, 505-511.

Davies, R. B. and Harte, D. S. (1987). Tests for Hurst Effect. Biometrika 74, 95–101.

McLeod, A.I., Yu, Hao, Krougly, Zinovi L. (2007). Algorithms for Linear Time Series Analysis, Journal of Statistical Software.

See Also

\DLSimulate , SimGLP, arima.sim

Examples

```
#simulate a process with autocovariance function 1/(k+1), k=0,1,...
# and plot it
n<-2000
r<-1/sqrt(1:n)
z<-DHSimulate(n, r)
plot.ts(z)

#simulate AR(1) and produce a table comparing the theoretical and sample
# autocovariances and autocorrelations
phi<- -0.8
n<-4096
g0<-1/(1-phi^2)
#theoretical autocovariances
tacvf<-g0*(phi^(0:(n-1)))
z<-DHSimulate(n, tacvf)
#autocorrelations
sacf<-acf(z, plot=FALSE)$acf
#autocovariances
sacvf<-acf(z, plot=FALSE,type="covariance")$acf
tacf<-tacvf/tacvf[1]
tb<-matrix(c(tacvf[1:10],sacvf[1:10],tacf[1:10],sacf[1:10]),ncol=4)
dimnames(tb)<-list(0:9, c("Tacvf","Sacvf","Tacf","Sacf"))
tb

#Show the Davies-Harte condition sometimes hold and sometimes does not
#   in the case of fractionally differenced white noise
#
#Define autocovariance function for fractionally differenced white noise
`tacvfFdwn` <-
function(d, maxlag)
{
    x <- numeric(maxlag + 1)
    x[1] <- gamma(1 - 2 * d)/gamma(1 - d)^2
    for(i in 1:maxlag)
        x[i + 1] <- ((i - 1 + d)/(i - d)) * x[i]
    x
}
#Build table to show values of d for which condition is TRUE when n=5000
n<-5000
ds<-c(-0.45, -0.25, -0.05, 0.05, 0.25, 0.45)
tb<-logical(length(ds))
names(tb)<-ds
for (kd in 1:length(ds)){
    d<-ds[kd]
    r<-tacvfFdwn(d, n-1)
    tb[kd]<-DHSimulate(n, r, ReportTestOnly = TRUE)
    }
tb
```

DLAcfToAR *Autocorrelations to AR parameters*

Description

Given autocorrelations at lags 1,...,n the AR parameters corresponding to the AR coefficients, partial autocorrelations (pacf) and standarized minimum-mean-square predictor variance (sigsqk) are computed. Can also be used as a test for valid acf sequence.

Usage

```
DLAcfToAR(r, useC = TRUE, PDSequenceTestQ = FALSE)
```

Arguments

r	autocorrelations starting at lag 1
useC	TRUE, C-interface function used. Otherwise if FALSE calculations are done in R
PDSequenceTestQ	FALSE, an error message is given if the autocorrelation sequence in not pd otherwise test for pd

Details

This function is more general than the built-in acf2AR since it provides the pacf and standardized minimum-mean-square error predictors. The standardized minimum-mean-square error predictor variances are defined as the minimum-mean-square error predictor variance for an AR process with unit variance. So for a sufficiently high-order, an approximation to the innovation variance is obtained.

The pacf may be used as an alternative parameterization for the linear time series model (McLeod and Zhang, 2006).

Value

a matrix with 3 columns and length(r) rows is returned corresponding to the ar coefficients, pacf and sigsqk when PDSequenceTestQ = FALSE. Otherwise when PDSequenceTestQ = TRUE, the result is TRUE or FALSE according as the autocorrelation is a valid positive-definite sequence.

Author(s)

A.I. McLeod

References

McLeod, A.I. and Zhang, Y. (2006). Partial autocorrelation parameterization for subset autoregression. Journal of Time Series Analysis, 27, 599-612.

McLeod, A.I., Yu, Hao, Krougly, Zinovi L. (2007). Algorithms for Linear Time Series Analysis, Journal of Statistical Software.

See Also

acf2AR, ar

Examples

```
#Example 1:  Yule-Walker estimates
z<-log(lynx)
p<-11
r<-(acf(z, lag.max=p, plot=FALSE)$acf)[-1]
ans<-DLAcfToAR(r)
#compare with built-in ar
phiAR<-ar(z,aic=FALSE, order.max=p, method="yw")$ar
#yet another way is to use acf2AR
phi2<-(acf2AR(c(1,r)))[p,]
cbind(ans,phiAR,phi2)
#
#Example 2:  AR(1) illustration
#For AR(1) case compare useC = T and F
r<-0.9^(1:3)
DLAcfToAR(r, useC=TRUE)
DLAcfToAR(r, useC=FALSE)
DLAcfToAR(r, useC=TRUE, PDSequenceTestQ=TRUE)
DLAcfToAR(r, useC=FALSE, PDSequenceTestQ=TRUE)
#
#Example 3: test for valid tacf
r<-c(0.8, rep(0,99))
DLAcfToAR(r, PDSequenceTestQ=TRUE)
#
#Example 4: Fractional-difference example
#Hosking (1981), pacf, zeta[k]=d/(k-d)
#we compare this numerically with our procedure
`tacvfFdwn` <-
function(d, maxlag)
{
    x <- numeric(maxlag + 1)
    x[1] <- gamma(1 - 2 * d)/gamma(1 - d)^2
    for(i in 1:maxlag)
        x[i + 1] <- ((i - 1 + d)/(i - d)) * x[i]
    x
}
n<-10
d<-0.4
r<-tacvfFdwn(d, n)
r<-(r/r[1])[-1]
HoskingPacf<-d/(-d+(1:n))
cbind(DLAcfToAR(r),HoskingPacf)
#
# Example 5: Determining a suitable MA approximation
#Find MA approximation to hyperbolic decay series
N<-10^4  #pick N so large that mmse forecast error converged
r<-1/sqrt(1:N)
out<-DLAcfToAR(r[-1])
```

```
InnovationVariance<-out[nrow(out),3]
phi<-out[,1]
psi<-ARMAtoMA(ar=phi, lag.max=N)
Error<-r[1]-InnovationVariance*(1+sum(psi^2))
```

`DLLoglikelihood` *Durbin-Levinsion Loglikelihood*

Description

The Durbin-Levinsion algorithm is used for the computation of the exact loglikelihood function.

Usage

```
DLLoglikelihood(r, z, useC = TRUE)
```

Arguments

`r`	autocovariance or autocorrelation at lags 0,...,n-1, where n is length(z)
`z`	time series data
`useC`	TRUE, use compiled C, otherwise R

Details

The concentrated loglikelihood function may be written Lm(beta) = -(n/2)*log(S/n)-0.5*g, where beta is the parameter vector, n is the length of the time series, S=z'M z, z is the mean-corrected time series, M is the inverse of the covariance matrix setting the innovation variance to one and g=-log(det(M)). This method was given in Li (1981) for evaluating the loglikelihood function in the case of the fractionally differenced white noise.

Value

The loglikelihood concentrated over the parameter for the innovation variance is returned.

Note

The purpose of this function is to provide a check on the TrenchLoglikelihod function. Completely different algorithms are used in each case but the numerical values should agree.

Author(s)

A.I. McLeod

References

W.K. Li (1981). Topics in Time Series Analysis. Ph.D. Thesis, University of Western Ontario.

McLeod, A.I., Yu, Hao, Krougly, Zinovi L. (2007). Algorithms for Linear Time Series Analysis, Journal of Statistical Software.

See Also

TrenchLoglikelihood

Examples

```
#Example 1
#compute loglikelihood for white noise
z<-rnorm(100)
DLLoglikelihood(c(1,rep(0,length(z)-1)), z)

#Example 2
#simulate a time series and compute the concentrated loglikelihood using DLLoglikelihood and
#compare this with the value given by TrenchLoglikelihood.
phi<-0.8
n<-200
r<-phi^(0:(n-1))
z<-arima.sim(model=list(ar=phi), n=n)
LD<-DLLoglikelihood(r,z)
LT<-TrenchLoglikelihood(r,z)
ans<-c(LD,LT)
names(ans)<-c("DLLoglikelihood","TrenchLoglikelihood")

#Example 3
## Not run:
#Compare direct evaluation of AR(1) loglikelihood with DL method
#First define the exact concentrated loglikelihood function for AR(1)
AR1Loglikelihood <-function(phi,z){
n<-length(z)
S<-(z[1]^2)*(1-phi^2) + sum((z[-1]-phi*z[-n])^2)
0.5*log(1-phi^2)-(n/2)*log(S/n)
}
#Next run script to compare numerically the loglikelihoods.
#They should be identical.
phi<-0.8
n<-200
z<-arima.sim(list(ar=phi), n=n)
phis<-seq(0.1,0.95,0.05)
ansAR1<-ansDL<-numeric(length(phis))
for (i in 1:length(phis)) {
    ansAR1[i] <- AR1Loglikelihood(phis[i],z)
    r<-(1/(1-phis[i]^2))*phis[i]^(0:(n-1))
    ansDL[i] <- DLLoglikelihood(r,z,useC=FALSE)
}
ans<-matrix(c(ansDL,ansAR1),ncol=2)
dimnames(ans)<-list(phis, c("DL-method","AR1-method"))

## End(Not run)

#Example 4
## Not run:
#compare timings. See (McLeod, Yu, Krougly, Table 8).
 n<-5000
```

```
ds<-c(-0.45, -0.25, -0.05, 0.05, 0.25, 0.45)
tim<-matrix(numeric(3*length(ds)),ncol=3)
for (i in 1:length(ds)){
   d<-ds[i]
   alpha <- 1-2*d #equivalent hyperbolic autocorrelation
   r <- (1/(1:n))^alpha
   z<-DLSimulate(n,r)
   tim1a<-system.time(LL1<-DLLoglikelihood(r,z))[1]
   tim1b<-system.time(LL1<-DLLoglikelihood(r,z,useC=FALSE))[1]
   tim2<-system.time(LL2<-TrenchLoglikelihood(r,z))[1]
   tim[i,]<-c(tim1a,tim1b, tim2)
   }
dimnames(tim)<-list(ds, c("DL-C","DL-R","Trench"))
tim

## End(Not run)
```

DLResiduals *Prediction residuals*

Description

The Durbin-Levison algorithm is used to compute the one-step prediction residuals.

Usage

```
DLResiduals(r, z, useC = TRUE, StandardizedQ=TRUE)
```

Arguments

r	vector of length n containing the autocovariances or autocorrelations at lags 0,...,n-1
z	vector of length n, mean-corrected time series data
useC	if TRUE, the compiled C code is used, otherwise the computations are done entirely in R and much slower
StandardizedQ	TRUE, the residuals are divided by their standard deviation or FALSE, the raw prediction residuals are computed

Details

If the model is correct the standardized prediction residuals are approximately NID(0,1) and are asymptotically equivalent to the usual innovation residuals divided by the residual sd. This means that the usual diagnotic checks, such as the Ljung-Box test may be used.

Value

Vector of length n containing the residuals

Author(s)

A.I. McLeod

References

W.K. Li (1981). Topics in Time Series Analysis. Ph.D. Thesis, University of Western Ontario.

McLeod, A.I., Yu, Hao, Krougly, Zinovi L. (2007). Algorithms for Linear Time Series Analysis, Journal of Statistical Software.

See Also

DLLoglikelihood

Examples

```
# For the AR(1) the prediction residuals and innovation residuals are the same (except for
# t=1).  In this example we demonstrate the equality of these two types of residuals.
#
phi<-0.8
sde<-30
n<-30
z<-arima.sim(n=30,list(ar=phi),sd=sde)
r<-phi^(0:(n-1))/(1-phi^2)*sde^2
e<-DLResiduals(r,z)
a<-numeric(n)
for (i in 2:n)
    a[i]=z[i]-phi*z[i-1]
a<-a/sde
ERR<-sum(abs(e[-1]-a[-1]))
ERR
#
#Simulate AR(1) and compute the MLE for the innovation variance
phi <- 0.5
n <- 2000
sigsq <- 9
z<-arima.sim(model=list(ar=phi), n=n, sd=sqrt(sigsq))
g0 <- sigsq/(1-phi^2)
r <- g0*phi^(0:(n-1))
#comparison of estimate with actual
e<-DLResiduals(r,z,useC=FALSE, StandardizedQ=FALSE)
sigsqHat <- var(e)
ans<-c(sigsqHat,sigsq)
names(ans)<-c("estimate","theoretical")
ans
```

DLSimulate *Simulate linear time series*

Description

The Durbin-Levinsion recursions are used to simulate a stationary time series given an unit innovation sequence and given autocovariance function. Requires

$$O(n^2)$$

flops.

Usage

```
DLSimulate(n, r, useC = TRUE, rand.gen = rnorm, ...)
```

Arguments

n	length of time series to be generated
r	autocovariances, lags 0, ...,
useC	=TRUE, use C interface. Otherwise direct computation.
rand.gen	random number generator to use
...	optional arguments passed to rand.gen

Details

See Hipel and McLeod (1994) or McLeod, Yu and Krougly (2007).

Value

simulated time series of length n

Author(s)

A.I. McLeod

References

McLeod, A.I., Yu, Hao, Krougly, Zinovi L. (2007). Algorithms for Linear Time Series Analysis, Journal of Statistical Software.

See Also

DHSimulate, SimGLP, codearima.sim

Examples

```
#Simulate hyperbolic decay time series
#with Hurst coefficient, H=0.9
n<-2000
H<-0.9
alpha<-2*(1-H)  #hyperbolic decay parameter
r<-(1/(1:n))^alpha
z<-DLSimulate(n, r)
plot.ts(z)
#can use HurstK function in FGN library to estimate H
```

---

exactLoglikelihood *Exact log-likelihood and MLE for variance*

Description

Provides an exact log-likelihood that is exactly equal to the value of the probability density function with the random variables replaced by data and the parameters replaced by their estimated value. The corresponding estimate of the variance term is return.

Usage

```
exactLoglikelihood(r, z, innovationVarianceQ = TRUE)
```

Arguments

`r`	the portion of autocovariance function which when multiplied by the variance term equals the full autocovariance function.
`z`	the time series assumed to have mean zero
`innovationVarianceQ`	When TRUE, the variance term is the innovation variance and when FALSE it is the variance of the time series. For ARFIMA models, set to TRUE. But FGN requires setting innovationVarianceQ to FALSE since only the innovation variance is not known and so the likelihood has a slightly different form.

Details

This function uses the trench algorithm that is implememented in C. This function is provided to include all multiplicative constants. For many purposes, such as MLE, we only need to likelihood function up to a multiplicative constant. But for information criteria, we may need the constant terms so we can compare our results with other types of models or with other software such as arima(). The arima() function also computes the exact log-likelihood and uses it in the computation of the AIC and BIC.

Value

LL	exact log-likelihood
sigmaSq	MLE for the variance term. If innovationVarianceQ is TRUE, is the an estimate of the residual variance otherwise it is an estimate of the variance of the time series.

Author(s)

A. I. McLeod, aimcleod@uwo.ca

See Also

TrenchLoglikelihood, DLLoglikelihood

Examples

```
set.seed(7773311)
n <- 200
z <- arima.sim(model=list(ar=0.9, ma=-0.6), n=n, n.start=10^4)
out <- arima(z, order=c(1,0,1), include.mean=FALSE)
out
#note
#sigma^2 estimated as 0.9558:  log likelihood = -279.66,  aic = 565.31
r <- tacvfARMA(phi=coef(out)[1], theta=-coef(out)[2], maxLag=n-1)
exactLoglikelihood(r, z, innovationVarianceQ = TRUE)
#agrees!
```

---

innovationVariance *Nonparametric estimate of the innovation variance*

Description

The innovation variance is estimated using a high order AR approximation determined by the AIC or by using Kolmogoroff's formula with a smoothed periodogram. Default is AR.

Usage

```
innovationVariance(z, method = c("AR", "Kolmogoroff"), ...)
```

Arguments

z	time series
method	Default "AR". Set to "Kolmogoroff" for non-parametric periodogram estimate.
...	optional arguments that are passed to spec.pgram()

Value

the innovation variance

Author(s)

A. I. McLeod

See Also

exactLoglikelihood, PredictionVariance,

Examples

```
z<-sunspot.year
#fitting high-order AR
innovationVariance(z)
#using periodogram
innovationVariance(z, method="Kolmogoroff")
#using smoothed periodogram
innovationVariance(z, method="Kolmogoroff", span=c(3, 3))
#the plot argument for spec.pgram() works too
innovationVariance(z, method="Kolmogoroff", span=c(3, 3), plot=TRUE)
```

---

is.toeplitz *test if argument is a symmetric Toeplitz matrix*

Description

Auxilary function, used to validate the input of TrenchInverse

Usage

```
is.toeplitz(x)
```

Arguments

x	value to be tested

Details

A symmetric Toeplitz matrix of order n has (i,j)-entry of the form g[abs(1+i-j)], where g is a vector of length n.

Value

returns True or False according to whether x is or is not a symmetric Toeplitz matrix

Author(s)

A.I. McLeod

See Also

TrenchInverse, toeplitz

Examples

```
is.toeplitz(toeplitz(1:5))
is.toeplitz(5)
```

PredictionVariance *Prediction variance*

Description

The prediction variance of the forecast for lead times l=1,...,maxLead is computed given theoretical autocovariances.

Usage

```
PredictionVariance(r, maxLead = 1, DLQ = TRUE)
```

Arguments

r	the autocovariances at lags 0, 1, 2, ...
maxLead	maximum lead time of forecast
DLQ	Using Durbin-Levinson if TRUE. Otherwise Trench algorithm used.

Details

Two algorithms are available which are described in detail in McLeod, Yu and Krougly (2007). The default method, DLQ=TRUE, uses the autocovariances provided in r to determine the optimal linear mean-square error predictor of order length(r)-1. The mean-square error of this predictor is the lead-one error variance. The moving-average expansion of this model is used to compute any remaining variances (McLeod, Yu and Krougly, 2007). With the other Trench algorithm, when DLQ=FALSE, a direct matrix representation of the forecast variances is used (McLeod, Yu and Krougly, 2007). The Trench method is exact. Provided the length of r is large enough, the two methods will agree.

Value

vector of length maxLead containing the variances

Author(s)

A.I. McLeod

References

McLeod, A.I., Yu, Hao, Krougly, Zinovi L. (2007). Algorithms for Linear Time Series Analysis, Journal of Statistical Software.

See Also

predict.Arima, TrenchForecast, exactLoglikelihood

Examples

```
#Example 1. Compare using DL method or Trench method
va<-PredictionVariance(0.9^(0:10), maxLead=10)
vb<-PredictionVariance(0.9^(0:10), maxLead=10, DLQ=FALSE)
cbind(va,vb)
#
#Example 2. Compare with predict.Arima
#general script, just change z, p, q, ML
z<-sqrt(sunspot.year)
n<-length(z)
p<-9
q<-0
ML<-10
#for different data/model just reset above
out<-arima(z, order=c(p,0,q))
sda<-as.vector(predict(out, n.ahead=ML)$se)
#
phi<-theta<-numeric(0)
if (p>0) phi<-coef(out)[1:p]
if (q>0) theta<-coef(out)[(p+1):(p+q)]
zm<-coef(out)[p+q+1]
sigma2<-out$sigma2
r<-sigma2*tacvfARMA(phi, theta, maxLag=n+ML-1)
sdb<-sqrt(PredictionVariance(r, maxLead=ML))
cbind(sda,sdb)
#
#
#Example 3. DL and Trench method can give different results
#  when the acvf is slowly decaying. Trench is always
#  exact based on a finite-sample.
L<-5
r<-1/sqrt(1:(L+1))
va<-PredictionVariance(r, maxLead=L)
vb<-PredictionVariance(r, maxLead=L, DLQ=FALSE)
cbind(va,vb) #results are slightly different
r<-1/sqrt(1:(1000)) #larger number of autocovariances
va<-PredictionVariance(r, maxLead=L)
vb<-PredictionVariance(r, maxLead=L, DLQ=FALSE)
cbind(va,vb) #results now agree
```

SimGLP *Simulate GLP given innovations*

Description

Simulates a General Linear Time Series that can have nonGaussian innovations. It uses the FFT so it is O(N log(N)) flops where N=length(a) and N is assumed to be a power of 2. The R function `convolve` is used which implements the FFT.

Usage

```
SimGLP(psi, a)
```

Arguments

`psi`	vector, length Q, of MA coefficients starting with 1.
`a`	vector, length Q+n, of innovations, where n is the length of time series to be generated.

Details

$$z_t = \sum_{k=0}^{Q} psi_k a_{t-k}$$

where $t = 1, \ldots, n$ and the innovations $a_t, t=1-Q, \ldots, 0, 1, \ldots, n$ are given in the input vector a.

Since `convolve` uses the FFT this is faster than direct computation.

Value

vector of length n, where n=length(a)-length(psi)

Author(s)

A.I. McLeod

See Also

`convolve`, `arima.sim`

Examples

```
#Simulate an AR(1) process with parameter phi=0.8 of length n=100 with
#  innovations from a t-distribution with 5 df and plot it.
#
phi<-0.8
psi<-phi^(0:127)
n<-100
Q<-length(psi)-1
a<-rt(n+Q,5)
z<-SimGLP(psi,a)
z<-ts(z)
plot(z)
```

tacvfARMA *theoretical autocovariance function (acvf) of ARMA*

Description

The theoretical autocovariance function of ARMA(p,q) process is computed. This is more useful in some situations than the built-in R function ARMAacf. See Details.

Usage

```
tacvfARMA(phi = numeric(0), theta = numeric(0), maxLag = 1, sigma2 = 1)
```

Arguments

phi	ar parameters
theta	ma parameters
maxLag	acvf is computed at lags 0, ..., maxLag
sigma2	innovation variance

Details

The details of the autocovariance computation are given in McLeod (1975).

In addition to this computation, we also test if the model is stationary-causal or not. The test, which is included directly in the function, uses the Durbin-Levison recursion to transform from the phi parameters to the pacf. See McLeod and Zhang (2006, eqn. (1)) for more details. Formally, the stationary-causal condition requires that all roots of the polynomial equation,

$$1 - phi[1] * B - ... - phi[p] * B^p = 0$$

must lie outside the unit circle (Brockwell and Davis, 1991, Section 3.3).

This function is included because it is necessary to demonstrate that in the case of ARMA models, TrenchInverse and the built-in R function predict.Arima produce equivalent results. See Example 1 in the documentation for TrenchForecast and the example discussed in McLeod, Yu and Krougly (2007, 3.2).

Value

Vector of length maxLag containing the autocovariances at lags 0, ..., maxLag. But see Warning below.

Note

An error is returned if the model is not stationary-causal.

Author(s)

A.I. McLeod

References

P.J. Brockwell and R.A. Davis (1991) Time Series: Theory and Methods. Springer.

A.I. McLeod (1975) Derivation of the theoretical autocovariance function of autoregressive-moving average models, Applied Statistics 24, 255-256.

A.I. McLeod and Zhang, Y. (2006) Partial autocorrelation parameterizations for subset autoregression, Journal of Time Series Analysis,

McLeod, A.I., Yu, Hao, Krougly, Zinovi L. (2007). Algorithms for Linear Time Series Analysis, Journal of Statistical Software.

See Also

ARMAacf

Examples

```
#Example 1.  Estimate the acvf of a fitted ARMA model
#There are two methods but they give slighly different results,
#general script, just change z, p, q, ML
z<-sqrt(sunspot.year)
n<-length(z)
p<-9
q<-0
ML<-5
#for different data/model just reset above
out<-arima(z, order=c(p,0,q))
phi<-theta<-numeric(0)
if (p>0) phi<-coef(out)[1:p]
if (q>0) theta<-coef(out)[(p+1):(p+q)]
zm<-coef(out)[p+q+1]
sigma2<-out$sigma2
rA<-tacvfARMA(phi, theta, maxLag=n+ML-1, sigma2=sigma2)
rB<-var(z)*ARMAacf(ar=phi, ma=theta, lag.max=n+ML-1)
#rA and rB are slighly different
cbind(rA[1:5],rB[1:5])
#
#Example 2. Compute Rsq for fitted ARMA model
#Rsq = 1 - (series variance / innovation variance)
#Again there are two methods but only the first method is guaranteed to
#produce an Rsq which is non-negative!
#Run last example and then evaluate the script below:
RsqA <- 1 - rA/sigma2
RsqB <- 1 - rB/sigma2
#
#Example 3. Test if model is stationary-causal or not.
StationaryQ <- function(phi) tryCatch(is.vector(tacvfARMA(phi=phi)),error=function(e) FALSE )
StationaryQ(1.1) #AR(1) with phi=1.1 is not stationary-causal.
#try with parameters from Example 1 above
StationaryQ(phi)
```

`ToeplitzInverseUpdate` *Inverse of Toeplitz matrix of order n+1 given inverse of order n*

Description

Let G be a Toeplitz matrix of order n and with (i,j)-element, r[Abs[i-j]]. So the first row of G may be written (r[0],...,r[n-1]). Suppose the next element in the sequence is r[n]. Then the inverse of the Toeplitz matrix whose first row is (r[0],...,r[n]) may be obtained either using ToeplitzInverseUpdate or directly using TrenchInverse. ToeplitzInverseUpdate is somewhat faster.

Usage

```
ToeplitzInverseUpdate(GI, r, rnew)
```

Arguments

`GI`	inverse of Toeplitz matrix G of order n
`r`	first row of G , ie r[0],...,r[n-1]
`rnew`	next element, r[n]

Details

Although this update requires $O(n^2)$ flops, the same as TrenchInverse, it is somewhat faster in practice.

Value

inverse matrix of order n+1

Author(s)

A.I. McLeod

References

Graybill, F.A. (1983). Matrices with Applications in Statistics.

McLeod, A.I., Yu, Hao, Krougly, Zinovi L. (2007). Algorithms for Linear Time Series Analysis, Journal of Statistical Software.

See Also

`TrenchInverse`

Examples

```
#In this example we compute the update inverse directly and using ToeplitzInverseUpdate and
#compare the result.
phi<-0.8
sde<-30
n<-30
r<-arima.sim(n=30,list(ar=phi),sd=sde)
r<-phi^(0:(n-1))/(1-phi^2)*sde^2
n1<-25
G<-toeplitz(r[1:n1])
GI<-solve(G) #could also use TrenchInverse
GIupdate<-ToeplitzInverseUpdate(GI,r[1:n1],r[n1+1])
GIdirect<-solve(toeplitz(r[1:(n1+1)]))
ERR<-sum(abs(GIupdate-GIdirect))
ERR
```

`TrenchForecast` *Minimum Mean Square Forecast*

Description

Given time series of length n+m, the forecasts for lead times k=1,...,L are computed starting with forecast origin at time t=n and continuing up to t=n+m. The input time series is of length n+m. For purely out-of-sample forecasts we may take n=length(z). Note that the parameter m is inferred using the fact that m=length(z)-n.

Usage

```
TrenchForecast(z, r, zm, n, maxLead, UpdateAlgorithmQ = TRUE)
```

Arguments

`z`	time series data, length n+m
`r`	autocovariances of length(z)+L-1 or until damped out
`zm`	mean parameter in model
`n`	forecast origin, n
`maxLead`	=L, the maximum lead time
`UpdateAlgorithmQ`	= TRUE, use efficient update method, otherwise if UpdateAlgorithmQ=FALSE, the direct inverse matrix is computed each time

Details

The minimum mean-square error forecast of z[N+k] given time series data z[1],...,z[N] is denoted by $z_N(k)$, where N is called the forecast origin and k is the lead time. This algorithm computes a table for $z_N(k), N = n, \ldots, n+m; k = 1, \ldots, m$ The minimum mean-square error forecast is simply the conditional expectation of z_{N+k} given the time series up to including time $t = N$. This conditional expectation works out to the same thing as the conditional expectation in an appropriate multivariate normal distribution – even if no normality assumption is made. See McLeod, Yu, Krougly (2007, eqn. 8). Similar remarks hold for the variance of the forecast. An error message is given if length(r) < n + L -1.

Value

A list with components

`Forecasts`	matrix with m+1 rows and maxLead columns with the forecasts
`SDForecasts`	matrix with m+1 rows and maxLead columns with the sd of the forecasts

Note

An error message is given if r is not a pd sequence, that is, the Toeplitz matrix of r must be pd. This could occur if you were to approximate a GLP which is near the stationary boundary by a MA(Q) with Q not large enough. In the bootstrap simulation experiment reported in our paper McLeod, Yu and Krougly (2007) we initially approximated the FGN autocorrelations by setting them to zero after lag 553 but in this case the ARMA(2,1) forecasts were always better. When we used all required lags of the acvf then the FGN forecasts were better as we expected. From this experience, we don't recommend setting high-order acf lags to zero unless the values are in fact very small.

Author(s)

A.I. McLeod

References

McLeod, A.I., Yu, Hao, Krougly, Zinovi L. (2007). Algorithms for Linear Time Series Analysis, Journal of Statistical Software.

See Also

`TrenchInverse`

Examples

```
#Example 1. Compare TrenchForecast and predict.Arima
#general script, just change z, p, q, ML
z<-sqrt(sunspot.year)
n<-length(z)
p<-9
q<-0
ML<-10
#for different data/model just reset above
```

```
out<-arima(z, order=c(p,0,q))
Fp<-predict(out, n.ahead=ML)

phi<-theta<-numeric(0)
if (p>0) phi<-coef(out)[1:p]
if (q>0) theta<-coef(out)[(p+1):(p+q)]
zm<-coef(out)[p+q+1]
sigma2<-out$sigma2
#r<-var(z)*ARMAacf(ar=phi, ma=theta, lag.max=n+ML-1)
#When r is computed as above, it is not identical to below
r<-sigma2*tacvfARMA(phi, theta, maxLag=n+ML-1)
F<-TrenchForecast(z, r, zm, n, maxLead=ML)
#the forecasts are identical using tacvfARMA
#
#Example 2. Compare AR(1) Forecasts.  Show how
#Forecasts from AR(1) are easily calculated directly.
#We compare AR(1) forecasts and their sd's.
#Define a function for the AR(1) case
AR1Forecast <- function(z,phi,n,maxLead){
        nz<-length(z)
        m<-nz-n
        zf<-vf<-matrix(numeric(maxLead*m),ncol=maxLead)
        zorigin<-z[n:nz]
        zf<-outer(zorigin,phi^(1:maxLead))
        vf<-matrix(rep(1-phi^(2*(1:maxLead)),m+1),byrow=TRUE,ncol=maxLead)/(1-phi^2)
        list(zf=zf,sdf=sqrt(vf))
        }
#generate AR(1) series and compare the forecasts
phi<-0.9
n<-200
m<-5
N<-n+m
z<-arima.sim(list(ar=phi), n=N)
maxLead<-3
nr<-N+maxLead-1
r<-(1/(1-phi^2))*phi^(0:nr)
ansT1<-TrenchForecast(z,r,0,n,maxLead)
ansT2<-TrenchForecast(z,r,0,n,maxLead,UpdateAlgorithmQ=FALSE)
ansAR1<-AR1Forecast(z,phi,n,maxLead)
```

`TrenchInverse` *compute the matrix inverse of a positive-definite Toepliz matrix*

Description

The Trench algorithm (Golub and Vanload, 1983) is implemented in C and interfaced to R. This provides an expedient method for obtaining the matrix inverse of the covariance matrix of n successive observations from a stationary time series. Some applications of this are discussed by McLeod and Krougly (2005).

Usage

```
TrenchInverse(G)
```

Arguments

G	a positive definite Toeplitz matrix

Value

the matrix inverse of G is computed

Warning

You should test the input x using is.toeplitz(x) if you are not sure if x is a symmetric Toeplitz matix.

Note

TrenchInverse(x) assumes that x is a symmetric Toeplitz matrix but it does not specifically test for this. Instead it merely takes the first row of x and passes this directly to the C code program which uses this more compact storage format. The C code program then computes the inverse. An error message is given if the C code algorithm encounters a non-positive definite input.

Author(s)

A.I. McLeod

References

Golub, G. and Van Loan (1983). Matrix Computations, 2nd Ed. John Hoptkins University Press, Baltimore. Algorithm 5.7-3.

McLeod, A.I., Yu, Hao, Krougly, Zinovi L. (2007). Algorithms for Linear Time Series Analysis, Journal of Statistical Software.

See Also

TrenchLoglikelihood, is.toeplitz, DLLoglikelihood, TrenchMean, solve

Examples

```
#compute inverse of matrix and compare with result from solve
data(LakeHuron)
r<-acf(LakeHuron, plot=FALSE, lag.max=4)$acf
R<-toeplitz(c(r))
Ri<-TrenchInverse(R)
Ri2<-solve(R)
Ri
Ri2

#invert a matrix of order n and compute the maximum absolute error
# in the product of this inverse with the original matrix
```

```
n<-5
r<-0.8^(0:(n-1))
G<-toeplitz(r)
Gi<-TrenchInverse(G)
GGi<-crossprod(t(G),Gi)
id<-matrix(0, nrow=n, ncol=n)
diag(id)<-1
err<-max(abs(id-GGi))
err
```

`TrenchLoglikelihood` *Loglikelihood function of stationary time series using Trench algorithm*

Description

The Trench matrix inversion algorithm is used to compute the exact concentrated loglikelihood function.

Usage

```
TrenchLoglikelihood(r, z)
```

Arguments

`r`	autocovariance or autocorrelation at lags 0,...,n-1, where n is length(z)
`z`	time series data

Details

The concentrated loglikelihood function may be written Lm(beta) = -(n/2)*log(S/n)-0.5*g, where beta is the parameter vector, n is the length of the time series, S=z'M z, z is the mean-corrected time series, M is the inverse of the covariance matrix setting the innovation variance to one and g=-log(det(M)).

Value

The loglikelihood concentrated over the parameter for the innovation variance is returned.

Author(s)

A.I. McLeod

References

McLeod, A.I., Yu, Hao, Krougly, Zinovi L. (2007). Algorithms for Linear Time Series Analysis, Journal of Statistical Software.

See Also

DLLoglikelihood

Examples

```
#compute loglikelihood for white noise
z<-rnorm(100)
TrenchLoglikelihood(c(1,rep(0,length(z)-1)), z)


#simulate a time series and compute the concentrated loglikelihood using DLLoglikelihood and
#compare this with the value given by TrenchLoglikelihood.
phi<-0.8
n<-200
r<-phi^(0:(n-1))
z<-arima.sim(model=list(ar=phi), n=n)
LD<-DLLoglikelihood(r,z)
LT<-TrenchLoglikelihood(r,z)
ans<-c(LD,LT)
names(ans)<-c("DLLoglikelihood","TrenchLoglikelihood")
```

TrenchMean *Exact MLE for mean given the autocorrelation function*

Description

Sometimes this is also referred to as the BLUE.

Usage

```
TrenchMean(r, z)
```

Arguments

r	vector of autocorrelations or autocovariances of length n
z	time series data vector of length n

Value

the estimate of the mean

Note

An error is given if r is not a postive-definite sequence or if the lengths of r and z are not equal.

Author(s)

A.I. McLeod

References

McLeod, A.I., Yu, Hao, Krougly, Zinovi L. (2007). Algorithms for Linear Time Series Analysis, Journal of Statistical Software.

See Also

TrenchInverse

Examples

```
#compare BLUE and sample mean
phi<- -0.9
a<-rnorm(100)
z<-numeric(length(a))
phi<- -0.9
n<-100
a<-rnorm(n)
z<-numeric(n)
mu<-100
sig<-10
z[1]<-a[1]*sig/sqrt(1-phi^2)
for (i in 2:n)
 z[i]<-phi*z[i-1]+a[i]*sig
z<-z+mu
r<-phi^(0:(n-1))
meanMLE<-TrenchMean(r,z)
meanBLUE<-mean(z)
ans<-c(meanMLE, meanBLUE)
names(ans)<-c("BLUE", "MLE")
ans
```

Index