I principali comandi di R per l'analisi delle serie storiche

versione 4.0

Francesco Lisi

Dipartimento di Scienze Statistiche - Università di Padova

Corso di Serie Storiche

Settembre 2026

Introduzione

- R è un linguaggio di programmazione orientato all'analisi statistica dei dati ed è disponibile, gratuitamente, per chiunque lo intenda utilizzare secondo la licenza GPL (Gnu Public Licence), al seguente indirizzo:
  https://cran.r-project.org/
- I nomi per gli oggetti possono essere formati dalle lettere dell'alfabeto, dalle cifre 0-9, e dal carattere punto: ".''. Un nome non può iniziare con una cifra (*9ma* non è
- ammesso ma *m9a* lo è) né con un punto seguito solo da cifre (*.90* non è ammesso ma *.9a* lo è).
- E' importante sottolineare che in R usare lettere maiuscole piuttosto che minuscole fa differenza e dunque, ad esempio, l'oggetto *ma* è diverso dall'oggetto *mA*.

1. Comandi generali

- L'assegnazione di un valore ad un variabile avviene tramite l'operatore <-
  a<-12
- Il concatenamento di oggetti si ottiene con il comando *c*
  x<-c(2,3,4)
  y<-c("stringa1", "stringa2", "stringa3")
- È possibile ottenere sequenze numeriche con il comando *seq*
  a<-seq(minimo, massimo, incremento)
- Per ottenere vettori di costanti o con andamenti regolari si usa il comando *rep*
  rep(costante, n. di volte)
  rep(min:max, n. di volte)
  rep(c(0,6), n. di volte)
- Per creare una matrice
  matrix(dati, nrow, ncol, byrow=TRUE)
- Prodotto matriciale (riga per colonna) tra due matrici A e B
  A %*% B
- Numero righe e colonne in una matrice x

```
nrow(x)
ncol(x)
```

- Per concatenare matrici o per aggiungere righe o colonne ad una matrice

```
rbind(mat1, mat2)        # concatena per riga
cbind(mat1, mat2)        # concatena per colonna
```

- Caricamento di un programma

```
source("c:/sse/sse")
```

- Caricamento di una libreria

```
library(ts)
```

- Per vedere gli oggetti correntemente disponibili

```
objects()
```

- Per rimuove l'oggetto di nome pippo

```
rm(pippo)
```

- Per rimuovere (quasi) ogni cosa dall'ambiente di lavoro

```
rm(list=ls())
```

- Per aprire una finestra di aiuto su un comando, ad es. il comando *mean*

```
?mean
```

2. Caricamento e salvataggio dei dati

- Si supponga di avere una serie storica in formato ASCII contenuta nel file *c:\sse\pippo.dat*. Per assegnare questa serie storica ad una variabile *x* di tipo array

```
x<-scan("c:/sse/pippo.dat")
```

Si noti l'uso della barra / al posto della barra rovesciata \ tipica del DOS.

- Nel caso di serie storiche è possibile creare degli oggetti di classe *ts* mediante la funzione *ts*. Per esempio, se il file contenesse 36 osservazioni a frequenza annuali partendo dal 1985 si avrebbe

```
x<-ts(scan("c:/sse/pippo.dat"),start=1985,frequency=1)
```

Con dati mensili (periodo 12), per il periodo gennaio 2015 - dicembre 2017, si dovrebbe scrivere

```
x<-ts(scan("c:/sse/pippo.dat"),start=c(2015,1),
end=c(2017,12), frequency=12)
```

Con dati trimestrali (periodo 4), partendo dal secondo trimestre del 2011 e arrivando fino al terzo trimestre del 2019, si dovrebbe scrivere

```
x=ts(scan("c:/sse/pippo.dat", start=c(2011,2),
end=c(2019,3), frequency=4)
```

- Con dati in formato CSV si può utilizzare il comando read.csv

  `x=read.csv(file="c:/sse/pippo.csv"), header=T, sep=",")`

  dove `header=T` indica che la prima riga va considerate come intestazione e `sep=","` che il separatore per le colonne è la virgola.

- Con dati in formato Excel (xlsx o xls) si può utilizzare il comando *read_excel* della la libreria *readxl*

  ```
  library(readxl)
  x <- read_excel="c:/sse/pippo.xlsx", skip = 10)
  ```

  dove `skip=10` indica il numero di righe che devono essere saltate prima della lettura dei dati

- Per visualizzare i dati si può utilizzare il comando

  `View(x)`

- Se si utilizza *R studio* è possibile importare dati direttamente dalla finestra *Environment* e quindi scegliere *Import Dataset from* (text o Excel)

  ATTENZIONE: un file può contenere più di una serie storica, quindi bisogna scegliere quella corretta

- Per salvare il valore di una variabile *y* in un file *c:\sse\results.out* si deve usare il comando

  `write(y,"c:/sse/reults.out")`

3. Grafici

- E' possibile creare il grafico di una serie di dati con il comando *plot*
  `plot(x, parametri)`
- Per creare un diagramma a dispersione tra i dati in *x* e quelli in *y*
  `plot(x,y, parametri)`
- Creazione di un istogramma
  `hist(x,breaks=15)`
- Alcune opzioni per il parametro *type* nel comando *plot*

Opzioni	Stile
`Type="p"`	Punti (default)
`Type="l"`	Linee
`Type="b"`	Linee e punti
`Type="h"`	Istogrammi

- Alcune opzioni per le funzioni del comando *plot*

Opzioni	Stile
`Main="`*titolo*`"`	Stringa del titolo
`Sub="`*sottotitolo*`"`	Stringa del sottotitolo
`Xlab="`*intestazione*`"`	Intestazione asse delle x
`Ylab="`*intestazione*`"`	Intestazione asse delle y
`Xlim=c(`*xmin,xmax)*	Scala dell'asse x
`Ylim=c(`*ymin,ymax)*	Scala dell'asse y
`Lwd=1`	Grossezza linea (1,2,..)
`Lty=1`	Tipo di linea (1=continua, 2..)

- Comandi che aggiungono elementi a grafici esistenti

Funzione	Descrizione
`Abline(a,b)`	Aggiunge al grafico corrente una linea con intercetta *a* e coefficiente angolare *b*
`Abline(`*h=c*`)`	Aggiunge una linea orizzontale in *y=c*
`Abline(`*v=c*`)`	Aggiunge una linea verticale in x=*c*
`Text(`*xpos,ypos,testo)*	Aggiunge *testo* nella posizione specificata

- Grafici di sottoserie

```
plot(window(x,n1,n2))        # sottoserie n1-n2 #
```

- Grafici multipli

```
win.graph(width=6, height=9)
par(mfrow=c(3,1))
plot(serie1)
plot(serie2)
plot(serie3)
```

- Per rappresentare diverse serie sullo stesso grafico è disponibile il comando *ts.plot* appartenente alla libreria *ts*

```
ts.plot(serie1,serie2,…,gpars=list(col=c(2,3,…)))
```

- Box-plot stagionali

  Per fare il box-plot stagionale di una serie mensile **y** che comprende tutti i mesi dei 5 anni 2021-2025

```
ymat=matrix(y, nrow=5,ncol=12, byrow=T)
boxplot(ymat, col=rainbow(12)) # boxplot distribuzioni
                               # dei 12 mesi
```

Se la serie **y** non copre interi periodi stagionali, ma solo dei sotto-periodi, ad esempio marzo 2021 – novembre 2025, allora bisogna inserire degli **NA** in corrispondenza dei mesi mancanti:

```
ymat=matrix(c(NA, NA, y,NA),nrow=5,ncol=12, byrow=T)
boxplot(ymat, col=rainbow(12))
```

4. Analisi preliminari

- Lunghezza di una serie
  ```
  length(x)
  ```
- Varie statistiche descrittive
  ```
  summary(x)
  ```
- Media
  ```
  mean(x)
  ```
- Mediana
  ```
  median(x)
  ```
- Varianza
  ```
  var(x)
  ```
- Quantili
  ```
  quantile(x, c(0.3,0.8))
  ```
- Trasformazione logaritmica
  ```
  log(x)
  ```
- Differenziazione
  ```
  diff(x,lag=1,differences=1)
  ```
- Campionamento da una serie (una osservazione ogni 4)
  ```
  x.sub<-x[seq(1,length(x),by=4)] # a partire dalla prima
  x.sub<-x[seq(0,length(x),by=4)] # a partire dalla quarta
  ```
- Indice di asimmetria (libreria *sse*)
  ```
  asimm(x)
  ```
- Indice di curtosi (libreria *sse*)
  ```
  curtosi(x)
  ```

5. Stima delle componenti mediante trend polinomiali e variabili dummy

- La "libreria" *sse* è una raccolta di funzioni creata a scopi didattici
  `source("sse.txt")`

- Stimare un trend polinomiale del tempo, ad esempio un trend cubico per una serie *y* di lunghezza *n*

```
t=1:n
t2=t^2
t3=t^3
mod=lm(y ~ t+t2+t3)
summary(mod)
```

  Il comando *lm* produce in output
  `mod$coefficients =` parametri stimati
  `mod$fitted.values=` valori stimati dal modello
  `mod$residuals` = residui del modello

- Per generare delle variabili dummy per una serie storica *y* con una componente periodica di periodo *freq* la cui prima osservazione si riferisce al periodo *start* si può usare il comando *make.dummy* della libreria *sse.* Per esempio se la frequenza è trimestrale:

```
dm <- make.dummy(length(y),start=1, freq=4)
```

  con
  `start=` period da cui ha inizio la serie
  `freq =` lunghezza del periodo
  Il comando *make.dummy* produce una matrice con le dummy nelle colonne.

- Per stimare la componente stagionale con variabili dummy, e in relazione al comando appenda descritto:

```
mod=lm(y ~ -1+dm[,1]+ dm[,2]+ dm[,3]+ dm[,4])
```

6. Stima di un trend non lineare (curve di crescita)

Per stimare una curva di crescita, ad esempio del tipo esponenziale, con i minimi quadrati non lineari si può utilizzare il comando *nls*

```
nlmod=nls(y ~ a0 *exp(a1*tempo), start = list(a0 = 10, a1 = 0.5))
summary(nlmod)
```

Stime dei coefficienti del modello, dei residui e dei valori del trend

`coef(nlmod) =` stime dei coefficienti

`resid(nlmod)=` residui del modello

`fitted(nlmod) =` valori stimati dal modello `#y-res`

7. Medie mobili

- Per calcolare una media mobile ponderata e centrata di ordine *order* (essendo centrata deve essere dispari), con vettore dei pesi *wts*, per la serie storica *x* si può utilizzare il comando *wma* della libreria *sse*

```
wma(x,order=5,wts=c(1/8,1/4,1/4,1/4,1/8),centre=TRUE,
plot=TRUE)
```

Se *centre=FALSE* la funzione calcola una media mobile ponderata degli ultimi *order* termini della serie. In questo caso la media mobile non è più centrata.

In alternativa, si può usare il comando *filter,* che è un comando di base di R

```
filter(x, filter= c(1/8,1/4,1/4,1/4,1/8))
```

dove `filter=` è il vettore dei pesi da assegnare alle osservazioni.

- Si possono calcolare i box-plot dei coefficienti stagionali specifici per ogni "mese" usando il comando *CS.matrice* della libreria *sse*

```
CS.mat=CS.matrice(CS, n.periodi=19, periodo=12, na.inizio=0,
na.fine=4)
```

dove

`CS` = è il vettore dei coefficienti stagionali specifici

`n.periodi` = è il numero di periodi interi, ad es. gli anni per dati mensili

`periodo` = è la lunghezza del periodo

`na.inizio` = da che "mese" ha inizio la serie

`na.fine` = a che "mese" finisce la serie

```
boxplot(CS.mat,col=rainbow(12))
```

8. Generazione di una serie da un modello ARIMA

- La generazione di *n* numeri casuali normali può avvenire mediante il comando *rnorm*

  `wn<-rnorm(200)`

- La generazione di *n* osservazioni da un modello ARIMA avviene mediante il comando *arima.sim*. Per esempio, per un ARIMA(1,1,1)

  `ts.sim <- arima.sim(n=200, model=list(order = c(1,1,1), ar=0.7, ma=0.3), sd = sqrt(0.18))`

  Per un ARMA(2,1)

  `y <-arima.sim(n,model=list(order=c(2,0,1),ar=c(0.5,0.3), ma=c(0.4)), sd = 1)`

9. Generazione di una serie da un modello SARIMA

- **`Library(sarima)`**

- La generazione di *n* osservazioni da un modello ARMA avviene mediante il comando *sim_sarima* che fa parte della libreria *sarima*. Per esempio, per generare dati da un modello SARIMA(2,1,1)(1,1,1)_12 si dovrà scrivere

```
x <- sim_sarima(n=144, model=list(ar=c(1.2, -0.8),
ma=0.4, sar=0.3, sma=0.7, iorder=1, siorder=1,
nseasons=12, sigma2 = 1))
```

dove *iorder* e *siorder* corrispondono a *d* e *D*, mentre *nseason* è il periodo.

10. Funzioni di autocovarianza e autocorrelazione

- Il calcolo dell'autocorrelazione teorica di un processo ARMA avviene tramite il comando *ARMAacf*. Ad es. per calcolare la funzione di autocorrelazione globale teorica per un modello ARMA(2,1) per i primi 15 ritardi si dovrà scrivere:

  ```
  ARMAacf(ar = c(1,-0.25), ma =c(0.3), lag.max =15, pacf =
  FALSE)
  ```

  Ponendo *pacf=TRUE* si ottiene la corrispondente funzione di autocorrelazione parziale teorica.

- Il calcolo della funzione di autocorrelazione campionaria avviene tramite il comando *acf*

  ```
  acf(x,lag.max=36,type=c("correlation","covariance",
  "partial"), plot=TRUE)
  ```

  `lag.max` = massimo ritardo per il quale calcolare l'autocorrelazione
  `type` = permette di calcolare la correlazione (default), la covarianza o la correlazione parziale.
  `plot` = se posto uguale a FALSE non viene rappresentato il grafico

- Il calcolo della funzione campionaria di autocorrelazione parziale può avvenire anche tramite il comando *pacf*

  ```
  pacf(x,lag.max=36,plot=TRUE)
  ```

- Il calcolo della funzione di autocorrelazione campionaria incrociata avviene tramite il comando *ccf* (libreria ts)

  `ccf(x,y,lag.max=36,type=c("correlation",covariance"), plot=TRUE)`

11. Stima di un modello (S)ARIMA

- La stima di un modello lineare autoregressivo può avvenire tramite il comando *ar.ols*.

  `fit.x<-ar.ols(x,aic=TRUE,order.max=10,intercept=T)`

  Questo comando adatta alla serie *x* un modello AR(p) stimando i parametri con le equazioni di Yule-Walker e selezionandone l'ordine mediante il criterio AIC nell'insieme dei modelli AR di ordine $p \in (0,1,2,\ldots,10)$

  Se invece si pone *`aic=FALSE`* `ad esempio`

  `fit.x<-ar.ols(x,aic=FALSE,order.max=3,intercept=F)`

  La stima avviene come sopra, ma ora viene stimato solo un modello di ordine 3, in questo caso senza costante.

  I valori del comando *ar* sono

  `fit.x$ar` fornisce le stime dei parametri

  `fit.x$asy.var.coef` fornisce, basandosi sulla teoria asintotica, la matrice di varianza-covarianza, dei coefficienti stimati

  `fit.x$order` fornisce informazioni sull'ordine del modello

  `fit.x$resid` fornisce la serie dei residui del modello stimato

  `fit.x$aic` fornisce informazioni sui valori di AIC

- Una procedura preferibile viene fornita dal comando *Arima* della libreria *forecast* il quale stima un modello ARIMA(p,d,q) mediante la verosimiglianza

```
library(forecast)
fit.x <- Arima(x,order=c(p,d,q),include.mean=T,
include.drift=F)
```

Se `Include.mean=T` viene stimato un modello ARMA con costante. Si noti che per modelli integrati (`d=1`) la costante viene comunque tralasciata.
Se `Include.drift=T` viene stimato un modello ARIMA con drift. Si noti che per modelli con `d>1` non viene calcolata né la costante sulla parte ARMA né il drift

I valori del comando *Arima* sono:

`fit.x`	primo riassunto dei risultati
`fit.x$coef`	fornisce i valori delle stime dei coefficienti
`fit.x$var.coef`	fornisce la varianza degli stimatori dei coefficienti
`fit.x$arma`	fornisce informazioni sugli ordini del modello
`fit.x$loglik`	fornisce il valore delle log-verosimiglianza massimizzata
`fit.x$aic`	fornisce il valore di AIC corrispondente alla log-verosimiglianza
`fit.x$aicc`	fornisce il valore di AICc corrispondente alla log-verosimiglianza
`fit.x$bic`	fornisce il valore di BIC corrispondente alla log-verosimiglianza
`fit.x$code`	1 o 2 se OK (maggiori dettagli sull'*help* della funzione *nlm*)
`fit.x$resid`	fornisce i residui del modello stimato
`fit.x$sigma2`	fornisce la stima MV della varianza dell'errore

N.B. *– R utilizza la parametrizzazione ARMA in cui gli epsilon hanno segno positivo per cui la parte MA del modello dovrà essere scritta con il segno che si legge nell'output.*

- La funzione *confint* restituisce, in relazione al modello stimato *fit.x*, gli estremi degli intervalli di confidenza per le stime dei parametri al livello *level*

```
confint(fit.x, level=0.95)
```

- È possibile applicare il modello ad un serie alla quale è stata applicata la trasformata di Box-Cox, ad es.

```
fit.x <- Arima(BoxCox(x,lambda=0),order=c(0,1,1),
          seasonal=list(order=c(0,1,0), period=4))
```

- Per la stima di un modello stagionale SARIMA(p,d,q)x(P,D,Q) si usa il comando *Arima* con l'opzione *seasonal*

```
fit.x <- Arima(x,order=c(p,d,q),
               seasonal=list(order=c(P,D,Q),period=12),
               include.mean=T, include.drift=T)
```

I valori sono gli stessi di quelli visti precedentemente. In questo caso, per modelli con `d>1` o `D>1` o `(d=1` e `D=1)` non viene calcolata né la costante sulla parte ARMA né il drift

12. Verifica della bontà del modello stimato

- Stazionarietà ed invertibilità: per il calcolo del modulo delle radici di un polinomio si usa il comando *polyroot*. Ad es. per trovare le radici di $1-0.5x-0.3x^2=0$

```
polyroot(c(1,-0.5,-0.3))
```

Per determinarne il modulo

```
Mod(polyroot(c(1,-0.5,-0.3)))
```

o

```
abs(polyroot(c(1,-0.5,-0.3)))
```

N.B. *– Dato che R utilizza la parametrizzazione ARMA in cui gli epsilon hanno*

segno positivo per cui la parte MA, con R per la costruzione del polinomio caratteristico della parte MA bisogna utilizzare i segni come vengono prodotti da R. Ad esempio, se le stime di theta_1 e theta_2 sono 0.5 e 0.3 si deve utilizzare l'espressione `polyroot(c(1,0.5,0.3))`.

- Analisi dei residui (liberia forecast)

  `checkresiduals(fit.mod, lag, plot = TRUE)`

  lag lag massimo da utilizzare per il test di Ljung-Box

- Per effettuare il test di Box-Ljung si usa il comando *Ljung.Box* appartenente alla libreria *sse*

  `Bt <- Ljung.Box.2(x, lag=20, par=2, all=T)`

  lag è il valore del ritardo per cui si calcola il test, *par* vale 0 se il test viene effettuato sulla serie originale e $p+q$ se viene applicato ai residui di un modello ARMA(p,q). Se *all=T* il test viene calcolato per tutti i ritardi fino al ritardo *lag*.

`bt$statistics`	valore della statistica
`bt$parameter`	g.d.l. della distribuzione chi quadro appropriato
`bt$p.value`	p-value

  Il comando *Ljung.Box(...)* senza alcun assegnamento produce un riassunto contenente il valore della statistica test, i g.d.l. e il p-value per il particolare ritardo considerato.

- Per effettuare il test di Jarque e Bera si usa il comando *jarque.bera.test* facente parte della libreria *tseries*

  `jb<-jarque.bera.test(x)`

`jb$statistics`	valore della statistica test
`jb$parameter`	g.d.l. della distribuzione chi quadro

`jb$p.value` p-value

Il comando `jarque.bera.test(x)` senza alcun assegnamento produce un output simile a quello descritto per il test di Box.

- Per effettuare il test di Pierce sulla stagionalità residua si usa il comando *pierce.test* facente parte della libreria *sse*

  `pt<-pierce.test(x,m,s,para)`

  m numero di ritardi stagionali da considerare

  s periodo stagionale

  para numero di parametri stimati. Se il test viene fatto sulla serie originale para=0, altrimenti corrisponde all'ordine del modello ARMA.

  `pt$statistics` valore della statistica test

  `pt$p.value` p-value

- Per effettuare il test BDS di indipendenza si usa il comando *bds.test* facente parte della libreria *tseries*

  `bdsx<-bds.test(x,d=4,eps=c(eps1,eps2,eps3,…))`

  d indica il massimo valore di *d* per il quale il test viene calcolato. Ad es. *d=4* significa che il test viene calcolato per *d=2,3,4*

  eps vettore numerico di valori per *epsilon*. Il test viene calcolato per ogni valore di *d* e per ogni valore di *epsilon*. Generalmente *epsilon* viene espresso come multiplo della deviazione standard dei dati, ad es. *eps1=0.5*sd(x)*.

  `bdsx$statistics` matrice dei valori della statistica test corrispondenti a ciascuna coppia (*d*,*epsilon*)

  `bdsx$p.value` matrice dei valori dei p-value corrispondenti alla statistica test per ogni coppia (*d*,*epsilon*)

13. Previsione

- La previsione k passi in avanti si ottiene usando la funzione *forecast* della libreria omonima

```
library(forecast)
prev <- forecast(object=fit.mod, h=8, level=c(80,95),
                 fan=TRUE, bootstrap=FALSE)
```

object indica l'oggetto (prodotto da Arima) che contiene il modello stimato
h indica l'orizzonte predittivo
level indica un vettore con i livelli di confidenza degli intervalli predittivi
bootstrap: se TRUE vengono calcolati intervalli bootstrap

I suoi valori sono

`prev$mean` previsione puntuale (media)
`prev$lower` estremo inferiore della previsione intervallare
`prev$upper` estremo superiore della previsione intervallare
`prev$x` la serie originaria dei dati
`prev$fitted` valori "previsti" in-sample

- Se è stata applicata la trasformata di Box-Cox il comando *forecast* farà la previsione in relazione ai valori della variabile originale *x*

- Grafico della previsione

```
plot(prev)
```

14. Misure e test di accuratezza predittiva

- Misure di accuratezza predittiva (libreria forecast)

```
accuracy(prev)
```

- Test di Diebold e Mariano (libreria forecast)

```
dm.test(err1, err2,
        alternative= c("two.sided", "less","greater"),
        h=2, power=2, varestimator=c("acf", "bartlett"))
```

dove

`err1, err2` = errori di previsione del previsore 1 e del previsore 2

`alternative =` ipotesi alternative del test

`h =` orizzonte di previsione

`power =` potenza a cui elevare gli errori

`varestimator =` metodo di stima della varianza dello stimatore