ANALISI DELLE SERIE STORICHE CON R

Versione 0.4 -21 febbraio 2005

Vito Ricci
vito_ricci@yahoo.com

INDICE

1. Introduzione

L'ambiente statistico R[1] è dotato di una pluralità di comandi e funzioni molto utili nel campo dell'analisi delle serie storiche mettendo a disposizione dei pacchetti specifici come `tseries`, `ast`, `fSeries`, `its` e `dse`, oltre che ad un insieme di strumenti di base disponibili nel package `stats`.
Scopo di questo lavoro è illustrare alcuni di questi comandi attraverso un'applicazione pratica nelle varie fasi dell'analisi delle serie temporali. Si presuppone una conoscenza di base di R (si consiglia la lettura di "*An Introduction to R*" scaricabile da *http://cran.r-project.org/doc/manuals/R-intro.pdf*) e dell'analisi delle serie storiche, anche se verranno fatti alcuni brevi richiami teorici nel corso dei vari paragrafi. Chiaramente questo lavoro non pretende di fare una trattazione esaustiva della materia, ma solo di soffermarsi su quelli che sono gli aspetti principali e maggiormente ricorrenti nelle tecniche di analisi delle serie storiche.
L'esempio oggetto di analisi è una serie storica delle ore lavorate mensilmente in un'azienda da gennaio 1998 a marzo 2004 contenuti in file di testo denominato: *mydata.txt*, ubicato nella directory C:\, lavorando con R sotto un sistema operativo di tipo MS Windows.

---

[1] Per una breve descrizione dell'ambiente R si veda: Vito Ricci, *R : un ambiente opensource per l'analisi statistica dei dati*, Economia e Commercio, (1):69-82, 2004

2. Caricamento dei dati

Prima di poter iniziare l'analisi della serie storica occorre in primo luogo caricare i dati da una sorgente esterna (nel nostro caso un file di testo) e creare l'oggetto ts in R. Esistono due possibili modi per operare in tal senso:

a) usare la funzione `read.ts()` del package `tseries` che consente di leggere i dati dal file esterno e creare contestualmente la serie storica;
b) usare la funzione `read.table()` per leggere i dati dal file creando un dataframe, successivamente si impiega il comando `ts()` per ottenere la serie storica come oggetto di R.

2.1 Funzione `read.ts()`

La funzione `read.ts()` si trova nel package `tseries`[2] che occorre scaricare dal sito del CRAN in quanto non fa parte della configurazione base di R. Una volta scaricato e installato sulla propria macchina tale pacchetto addizionale è necessario caricarlo in memoria con il comando:

```
library(tseries)
```

di seguito, con tale comando si crea la serie storica denominata `ore` leggendo i dati dal file *mydata.txt*:

```
ore<-read.ts("C:/mydata.txt", header=TRUE, start=1998, frequency=12)

ore
       Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
1998 12508 12231 13195 12663 12498 12562 11812  6371 13658 14360 13537 12934
1999 13341 14403 16105 14616 15388 15045 14841  8626 15852 16250 16165 16164
2000 15532 16190 18077 15130 17737 16833 14988 10147 18286 18705 18079 16920
2001 19870 18028 20326 17804 20436 18552 16674 12154 19042 21728 20502 18461
2002 20749 18895 19845 18700 19802 18133 18811 11962 20416 22582 20759 20355
2003 20851 19786 21127 19540 24664 20602 20712 13016 23024 24845 22556 22049
2004 22874 22590 25268
```

Come si può vedere i parametri principali della funzione `read.ts()` risultano essere quattro. Occorre specificare il nome e la collocazione del file da cui leggere i dati; di seguito viene indicato (parametro `header`) se nel file vi è o meno l'indicazione di nome della variabile. In caso affermativo `header=TRUE`, in caso contrario `header=FALSE`, che è anche l'impostazione di default. I parametri `start` e `frequency` sono relativi alla serie storica e indicano rispettivamente il tempo della prima osservazione e la frequenza, ossia il numero di osservazioni per unità di tempo. Nel nostro caso, essendo la serie a cadenza mensile, si è impostato il parametro `frequency=12` e, iniziando a gennaio 1998, il valore di `start=1998`. A seconda dei casi `frequency` può assumere il valore 7 quando i dati sono rilevati giornalmente e il periodo di tempo di riferimento è la settimana, 12 quando i dati sono mensili, 4 quando sono trimestrali e 3 quando sono con frequenza quadrimestrale e il periodo di riferimento è l'anno. Il parametro `start` può essere un singolo numero o un vettore di due interi. Nel caso di serie con frequenza mensile gli elementi del vettore indicano rispettivamente l'anno e il mese a cui si riferisce la prima osservazione della serie. Se si indica solo il primo elemento, questo verrà considerato come l'anno iniziale e il mese pari a gennaio. Qualora si avesse una serie con dati trimestrali il primo elemento del vettore indica sempre l'anno, mentre il secondo il trimestre.
Nel caso in cui la nostra serie di dati relativi alle ore lavorate fosse iniziata ad aprile 1998 anziché a gennaio 1998, avremmo dovuto scrivere:

```
ore2<-read.ts("C:/mydata.txt", header=TRUE, start=c(1998,4), frequency=12)
ore2
       Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
1998                   12508 12231 13195 12663 12498 12562 11812  6371 13658
1999 14360 13537 12934 13341 14403 16105 14616 15388 15045 14841  8626 15852
2000 16250 16165 16164 15532 16190 18077 15130 17737 16833 14988 10147 18286
2001 18705 18079 16920 19870 18028 20326 17804 20436 18552 16674 12154 19042
2002 21728 20502 18461 20749 18895 19845 18700 19802 18133 18811 11962 20416
2003 22582 20759 20355 20851 19786 21127 19540 24664 20602 20712 13016 23024
```

[2] http://cran.r-project.org/src/contrib/Descriptions/tseries.html

```
2004 24845 22556 22049 22874 22590 25268
```

Analogamente nel caso di dati trimestrali con inizio al secondo trimestre del 1998 si avrebbe:

```
ore3<-read.ts("C:/mydata.txt", header=TRUE, start=c(1998,2), frequency=4)
ore3
      Qtr1  Qtr2  Qtr3  Qtr4
1998       12508 12231 13195
1999 12663 12498 12562 11812
2000  6371 13658 14360 13537
2001 12934 13341 14403 16105
2002 14616 15388 15045 14841
2003  8626 15852 16250 16165
2004 16164 15532 16190 18077
.....
2015 13016 23024 24845 22556
2016 22049 22874 22590 25268
```

2.2 Funzione `ts()`

Un modo alternativo e più tradizionale per creare una serie storica senza dover ricorrere a packages aggiuntivi è quello di usare il comando `ts()` presente nella versione base di R. In primo luogo occorre importare i dati dal file di testo *mydata.txt* e questo può essere fatto o con `scan()`, memorizzando i dati in un vettore, o con `read.table()`, nel qual caso vengono inseriti in un dataframe.

```
dati<-scan("C:/mydata.txt",skip=1)
Read 75 items

is.vector(dati)
[1] TRUE

dati
 [1] 12508 12231 13195 12663 12498 12562 11812  6371 13658 14360 13537 12934
[13] 13341 14403 16105 14616 15388 15045 14841  8626 15852 16250 16165 16164
[25] 15532 16190 18077 15130 17737 16833 14988 10147 18286 18705 18079 16920
[37] 19870 18028 20326 17804 20436 18552 16674 12154 19042 21728 20502 18461
[49] 20749 18895 19845 18700 19802 18133 18811 11962 20416 22582 20759 20355
[61] 20851 19786 21127 19540 24664 20602 20712 13016 23024 24845 22556 22049
[73] 22874 22590 25268
```

Si è inserito il parametro `skip=1` per saltare, in lettura, la prima riga del file di testo che contiene una etichetta (nome della variabile).

```
dati<-read.table("C:/mydata.txt", header=TRUE)

dati
   OreLavorate
1        12508
2        12231
3        13195
.....
73       22874
74       22590
75       25268

is.data.frame(dati)
[1] TRUE
```

Nel primo caso l'oggetto `dati` è un vettore, mentre nel secondo è un dataframe, tuttavia in entrambe le circostanze esso contiene i dati a partire dai quali bisogna creare la serie storica:

`ore<-ts(dati,start=1998,frequency=12) ##` se `dati` è un vettore

```
ore<-ts(dati$OreLavorate,start=1998,frequency=12)  ## se dati è un dataframe
ore
       Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
1998 12508 12231 13195 12663 12498 12562 11812  6371 13658 14360 13537 12934
1999 13341 14403 16105 14616 15388 15045 14841  8626 15852 16250 16165 16164
2000 15532 16190 18077 15130 17737 16833 14988 10147 18286 18705 18079 16920
2001 19870 18028 20326 17804 20436 18552 16674 12154 19042 21728 20502 18461
2002 20749 18895 19845 18700 19802 18133 18811 11962 20416 22582 20759 20355
2003 20851 19786 21127 19540 24664 20602 20712 13016 23024 24845 22556 22049
2004 22874 22590 25268
```

Il comando `ts()` presenta alcune analogie con `read.ts()`: il primo parametro indica la sorgente (vettore, matrice o dataframe) da cui vanno presi i dati, mentre gli altri due hanno il medesimo significato illustrato precedentemente.
Per conoscere il tempo della prima e dell'ultima osservazione di una serie storica si utilizzano i comandi `start()` ed `end()`, mentre `frequency()` restituisce il numero di osservazioni per unità di tempo:

```
start(ore)
[1] 1998    1

end(ore)
[1] 2004    3

frequency(ore)
[1] 12
```

La funzione `time()` restituisce sotto forma di oggetto ts l'insieme degli istanti di tempo della serie storica mentre `window()` consente di estrarre una serie derivata (un sottoinsieme di dati) da quella originale indicando l'inizio e la fine:

```
window(ore,start=c(1999,5),end=c(2001,5))

       Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
1999                         15388 15045 14841  8626 15852 16250 16165 16164
2000 15532 16190 18077 15130 17737 16833 14988 10147 18286 18705 18079 16920
2001 19870 18028 20326 17804 20436
```

3. Decomposizione di una serie storica

Uno degli scopi fondamentali dell'analisi classica delle serie temporali è quello di scomporre la serie nelle sue componenti, isolandole per poterle studiare meglio. Inoltre, per poter applicare l'approccio stocastico (modelli AR, MA, ARIMA) alle serie storiche è quasi sempre necessario eliminare il trend e la stagionalità al fine di avere un processo stazionario.
Le componenti di una serie storica di solito sono le seguenti: trend, stagionalità, ciclo, residua o erratica. Esse possono essere legate tra loro in modo additivo:

$$Y_t=T_t + C_t + S_t + E_t$$

oppure in modo moltiplicativo:

$$Y_t=T_t * C_t * S_t * E_t$$

Un modello di tipo moltiplicativo può essere facilmente trasformato in un modello additivo usando l'operatore logaritmo:

$$\log(Y_t)=\log(T_t)+ \log(C_t)+ \log(S_t)+ \log(E_t)$$

La componente stagionale è presente nel caso di serie storiche infrannuali (mensili, trimestrali, etc.), mentre il ciclo è tipico delle serie storiche che descrivono dei fenomeni economici in un periodo di osservazione

piuttosto lungo. Spesso, quando non è particolarmente evidente, la componente ciclica viene considerata all'interno della componente di fondo.
Per la stima del trend si ricorre di solito a funzioni tipiche come la retta (trend lineare), la parabola (trend parabolico), un polinomio di grado k, l'esponenziale (trend esponenziale), la logistica e via discorrendo. Tali funzioni sono quasi tutte lineari o comunque linearizzabili attraverso opportune trasformazioni.

3.1 Stima delle componenti

3.1.1 Medie mobili, differenze dei termini

Oltre al metodo analitico per la stima del trend, ci sono metodi più elementari anche se meno raffinati per detrendizzare una serie temporale: la perequazione meccanica con medie mobili e l'applicazione dell'operatore differenza. Nel caso delle medie mobili si pone il problema dell'esatta determinazione del numero dei termini da usare. In genere il trend può essere stimato con un'opportuna ponderazione dei valori della serie:

$$T_t = \frac{1}{2a+1}\sum_{-a}^{a} X_t$$

Il package *ast*[3] mette a disposizione la funzione `sfilter()`:

```
library(ast)
ore.filt<-sfilter(ore)
ore.filt
          Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
1998       NA       NA       NA       NA       NA       NA 12395.46 12520.67
1999 13590.37 13810.54 13995.92 14166.08 14354.33 14598.42 14824.29 14990.04
2000 15622.54 15692.04 15856.83 16060.54 16242.58 16353.83 16566.08 16823.42
2001 17748.67 17902.54 18017.67 18175.13 18402.04 18567.21 18668.04 18740.79
2002 18812.79 18893.83 18943.08 19035.92 19082.21 19171.83 19255.00 19296.38
2003 20200.46 20323.58 20476.17 20679.13 20848.29 20993.75 21148.63 21349.75
2004       NA       NA       NA
          Sep      Oct      Nov      Dec
1998 12732.42 12935.04 13136.83 13360.71
1999 15146.67 15250.25 15369.54 15541.92
2000 16993.71 17198.83 17422.71 17606.79
2001 18756.88 18774.17 18785.08 18741.21
2002 19386.92 19475.33 19712.92 20018.37
2003 21639.12       NA       NA       NA
```

Si è applicata ai dati originari una media mobile ponderata a 13 termini per eliminare la stagionalità e mettere in risalto solo la componente di fondo. Per ottenere una visualizzazione grafica Graf.1/b):

```
plot(ore.filt, main="Trend stimato con la media mobile")
```

In alternativa si può usare la funzione `filter()` presente nel package `stats`, fissando il parametro `filter` al valore corrispondente ad una serie mensile (1/25 per 25 volte per un valore di a=12 come dalla formula di sopra dei filtri lineari).

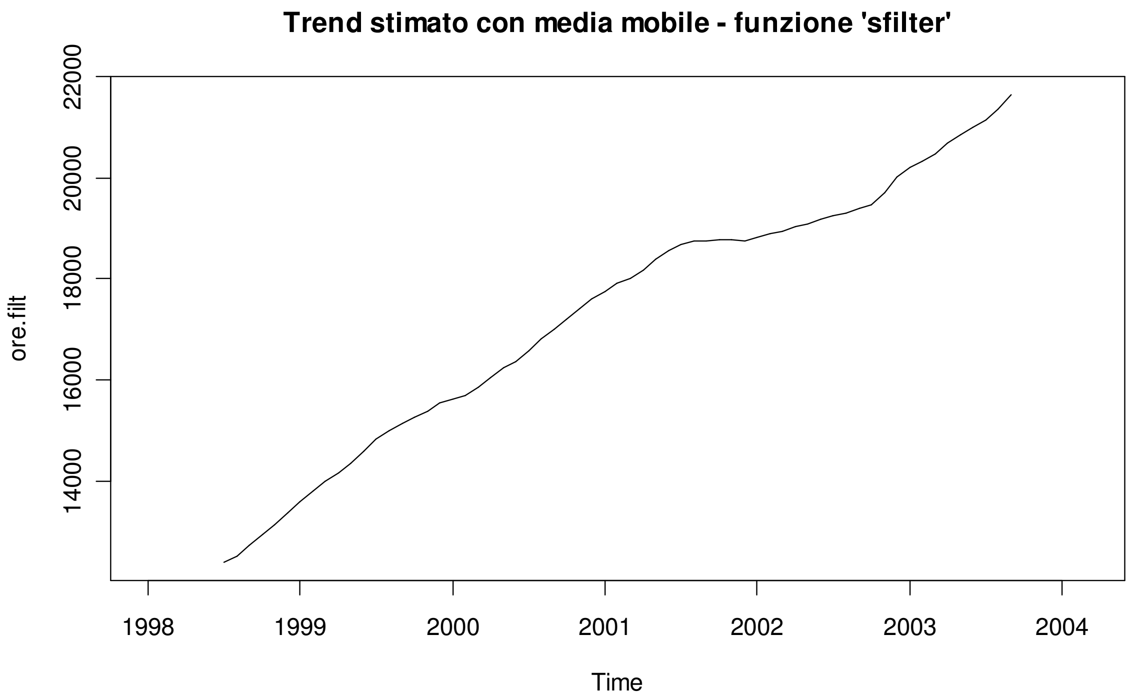
```
ore.fil2<-filter(ore, filter=rep(1/25,25))
ore.fil2
          Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
1998       NA       NA       NA       NA       NA       NA       NA       NA
1999 13626.28 13773.56 14007.40 14084.80 14287.76 14461.16 14558.20 14491.60
2000 15731.60 15919.08 16156.00 16223.96 16456.76 16583.32 16648.48 16541.00
2001 17638.00 17772.52 17918.72 17943.64 18130.52 18146.36 18225.48 18104.44
2002 19017.48 19014.12 19138.08 19106.64 19381.04 19387.68 19474.08 19327.76
2003 20266.20 20339.84 20594.76       NA       NA       NA       NA       NA
2004       NA       NA       NA
          Sep      Oct      Nov      Dec
```

[3] http://sirio.stat.unipd.it/index.php?id=libast

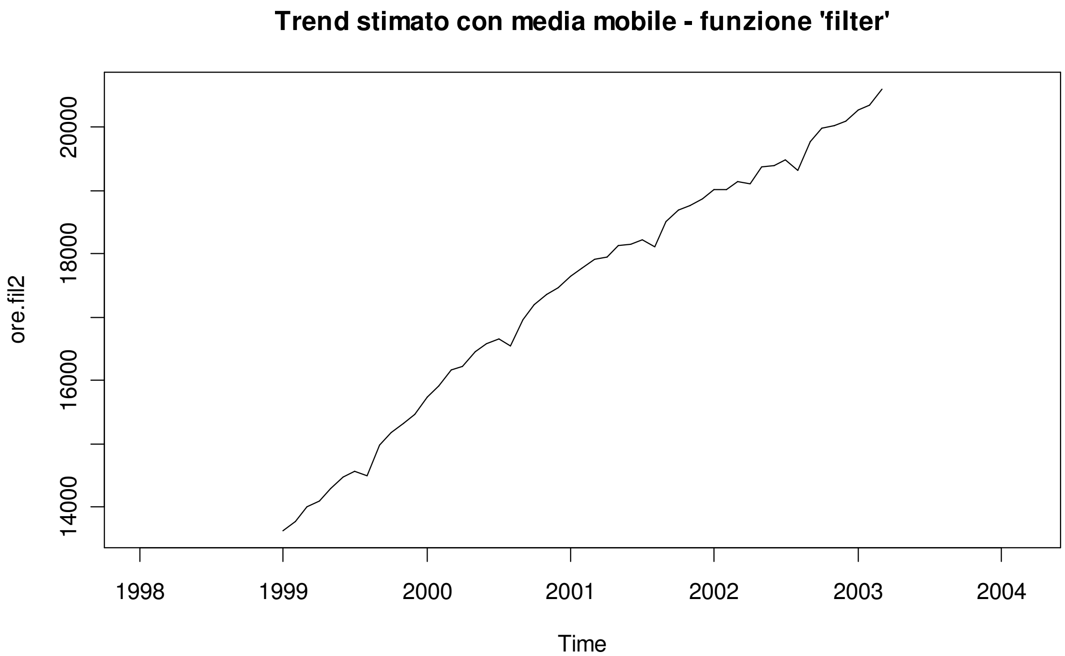
```
1998       NA       NA       NA       NA
1999 14968.20 15170.08 15318.84 15454.16
2000 16957.64 17192.68 17362.76 17454.60
2001 18515.20 18687.04 18769.20 18860.24
2002 19762.56 19994.68 20027.80 20089.68
2003       NA       NA       NA       NA
2004
```

e ne tracciamo il grafico (Graf. 1/b):

```
plot(ore.fil2, main="Trend stimato con media mobile - funzione 'filter'")
```



Graf. 1/a



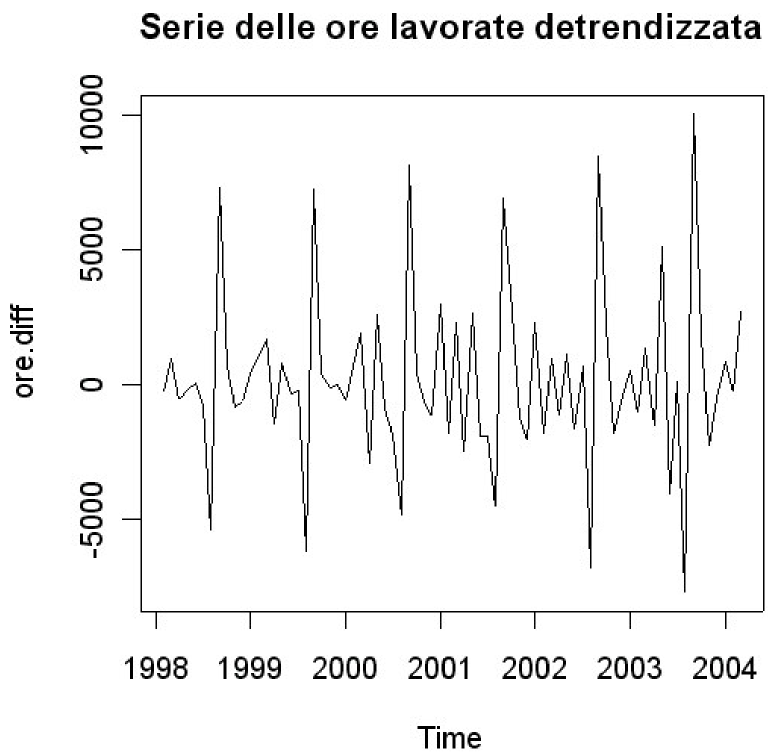
Graf. 1/b

Un ulteriore metodo per eliminare il trend è quello di operare sulle differenze tra i termini (o i logaritmi dei termini in caso di modello moltiplicativo) della serie storica: le differenze del primo ordine rimuovono un trend lineare, quelle del secondo ordine un trend parabolico, quelle di ordine k rimuovono un trend polinomiale di grado k:

$$\Delta_t = Y_{t+1} - Y_t$$

Il comando di R per applicare tale metodo è: `diff.ts()` che consente di operare le differenze del primo ordine e in generale, usato ripetitivamente, le differenze di qualsiasi ordine.

```
ore.diff<-diff.ts(ore)
ore.diff
       Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
1998        -277   964  -532  -165    64  -750 -5441  7287   702  -823  -603
1999   407  1062  1702 -1489   772  -343  -204 -6215  7226   398   -85    -1
2000  -632   658  1887 -2947  2607  -904 -1845 -4841  8139   419  -626 -1159
2001  2950 -1842  2298 -2522  2632 -1884 -1878 -4520  6888  2686 -1226 -2041
2002  2288 -1854   950 -1145  1102 -1669   678 -6849  8454  2166 -1823  -404
2003   496 -1065  1341 -1587  5124 -4062   110 -7696 10008  1821 -2289  -507
2004   825  -284  2678

plot(ore.diff,main="Serie delle ore lavorate detrendizzata")
```



Graf.2

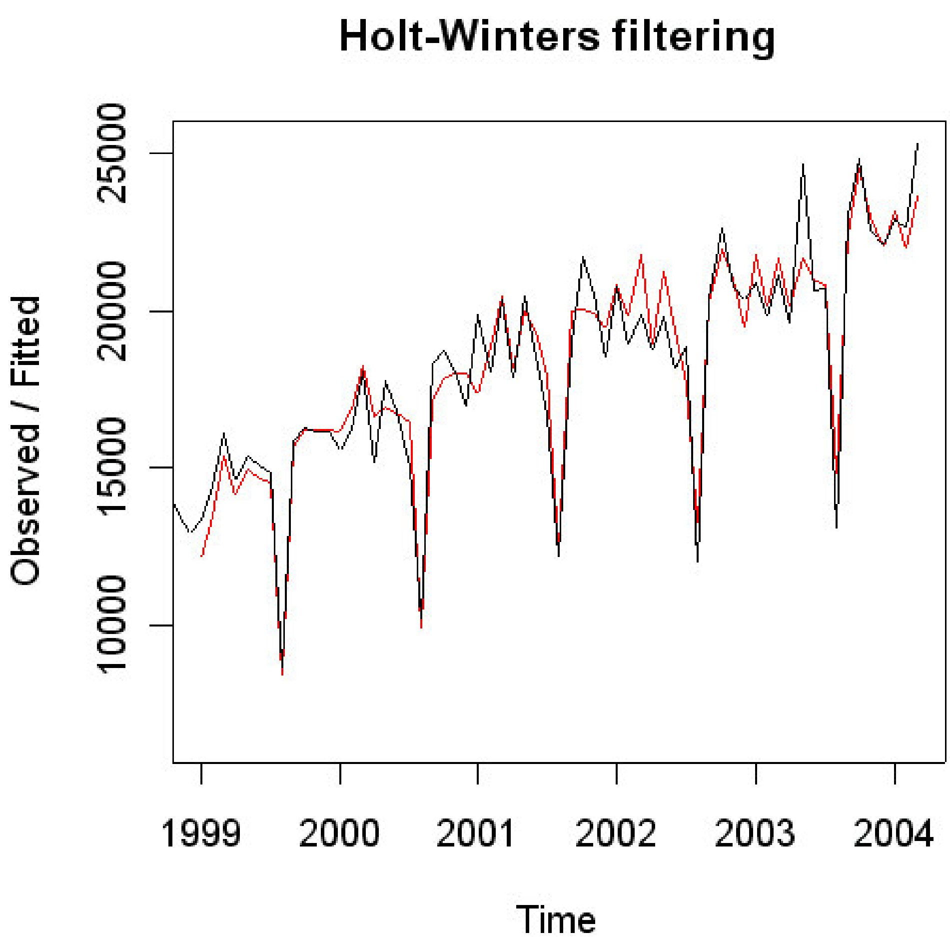
3.1.2 Livellamento esponenziale con il metodo di Holt-Winters

Il livellamento esponenziale[4] è un metodo che può aiutare a descrivere l'andamento di una serie storica e soprattutto è un utile strumento per effettuare delle previsioni. Il package `stats` mette a disposizione la funzione `HoltWinters()`. Usiamo tale funzione per la serie delle ore lavorate:

```
ore.hw<-HoltWinters(ore, seasonal="additive")
ore.hw

Holt-Winters exponential smoothing with trend and additive seasonal component.

Call:
 HoltWinters(x = ore, seasonal = "additive")

Smoothing parameters:
 alpha:  0.2489612
 beta :  0.0008647188
 gamma:  0.7306181

Coefficients:
           [,1]
a   23161.33721
b     174.25604
s1   -755.11091
s2   2721.81148
s3   -717.60356
s4   -824.29143
s5  -7931.82482
```

[4] Per un approfondimento si veda il seguente link: http://www.dss.uniud.it/utenti/proietti/StAnEc/Lezione3.pdf [consultato in data 24/11/04]

```
s6    1275.52658
s7    2889.53623
s8     692.07964
s9     -90.49324
s10    679.87066
s11   -118.12374
s12   1774.14192

plot(ore.hw)
```



Graf.3

Se vogliamo effettuare la previsione delle ore lavorate nei successivi 6 mesi impieghiamo il metodo `predict()` specificando il numero di periodi per i quali si vuole effettuare la previsione:

```
prev<-predict(ore.hw, n.ahead=6)

prev
          Apr      May      Jun      Jul      Aug      Sep
2004 22580.48 26231.66 22966.50 23034.07 16100.79 25482.40
```

3.1.3 Metodo analitico

L'approccio più semplice della decomposizione classica, utile a titolo introduttivo, è basato sul modello:

$$Y_t=f(t) + \varepsilon_t$$

in cui f(t) è una funzione del tempo che descrive trend e stagionalità in modo semplice. In particolare nel caso di un modello di tipo additivo, come nella nostra esemplificazione pratica:

$$Y_t= T_t + S_t + \varepsilon_t$$

con $\varepsilon_t \sim NID\ (0,\sigma^2)$, ovvero supponendo che gli errori siano distribuiti normalmente con media zero e varianza costante (omoschedasticità) e siano tra loro indipendenti. Sono le ipotesi di base della regressione lineare che verranno verificate tramite appositi tests di specificazione del modello.
R consente di determinare e stimare le tre componenti con ben tre diverse funzioni: `decompose()` e `stl()` nel package stat, `tsr()` nel package ast.
La funzione `decompose()` decompone la serie storica in stagionalità, trend e componente erratica usando il metodo della media mobile e può essere applicata sia al modello additivo che a quello moltiplicativo (occorre specificarlo nel parametro `type = c("additive", "multiplicative")`.

```
dec.fit<-decompose(ore, type="additive")
```

la funzione restituisce un oggetto della classe '"decomposed.ts"' con le seguenti componenti:

seasonal: un vettore con la stagionalità
figure: le stime della componente stagionale per i dodici mesi dell'anno
trend: un vettore con il trend della serie
random: un vettore con gli errori
type: "additive" o "moltiplicative" a seconda di quanto specificato nel parametro `type`

Memorizziamo in alcuni vettori le tre componenti ottenute dalla decomposizione.

```
stag.dec<-dec.fit$seasonal
stag.dec
            Jan         Feb         Mar         Apr         May         Jun
1998   896.82899   271.68316  1852.29774  -329.49392  1287.88108   -64.65017
1999   896.82899   271.68316  1852.29774  -329.49392  1287.88108   -64.65017
2000   896.82899   271.68316  1852.29774  -329.49392  1287.88108   -64.65017
2001   896.82899   271.68316  1852.29774  -329.49392  1287.88108   -64.65017
2002   896.82899   271.68316  1852.29774  -329.49392  1287.88108   -64.65017
2003   896.82899   271.68316  1852.29774  -329.49392  1287.88108   -64.65017
            Jul         Aug         Sep         Oct         Nov         Dec
1998 -1032.43142 -6772.98351   795.38108  2109.02691  1021.11024   -34.65017
1999 -1032.43142 -6772.98351   795.38108  2109.02691  1021.11024   -34.65017
2000 -1032.43142 -6772.98351   795.38108  2109.02691  1021.11024   -34.65017
2001 -1032.43142 -6772.98351   795.38108  2109.02691  1021.11024   -34.65017
2002 -1032.43142 -6772.98351   795.38108  2109.02691  1021.11024   -34.65017
2003 -1032.43142 -6772.98351   795.38108  2109.02691  1021.11024   -34.65017

trend.dec<-dec.fit$trend
trend.dec
          Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
1998       NA       NA       NA       NA       NA       NA 12395.46 12520.67
1999 13590.37 13810.54 13995.92 14166.08 14354.33 14598.42 14824.29 14990.04
2000 15622.54 15692.04 15856.83 16060.54 16242.58 16353.83 16566.08 16823.42
2001 17748.67 17902.54 18017.67 18175.13 18402.04 18567.21 18668.04 18740.79
2002 18812.79 18893.83 18943.08 19035.92 19082.21 19171.83 19255.00 19296.38
2003 20200.46 20323.58 20476.17 20679.13 20848.29 20993.75 21148.63 21349.75
2004       NA       NA       NA
          Sep      Oct      Nov      Dec
1998 12732.42 12935.04 13136.83 13360.71
1999 15146.67 15250.25 15369.54 15541.92
2000 16993.71 17198.83 17422.71 17606.79
2001 18756.88 18774.17 18785.08 18741.21
2002 19386.92 19475.33 19712.92 20018.37
```

```
2003 21639.12       NA       NA       NA
2004

res.dec<-dec.fit$random
res.dec
             Jan          Feb          Mar          Apr          May
1998          NA           NA           NA           NA           NA
1999 -1146.203993   320.775174   256.785590   779.410590  -254.214410
2000  -987.370660   226.275174   367.868924  -601.047743   206.535590
2001  1224.504340  -146.224826   456.035590   -41.631076   746.077257
2002  1039.379340  -270.516493  -950.381076    -6.422743  -568.089410
2003  -246.287326  -809.266493 -1201.464410  -809.631076  2527.827257
             Jun          Jul          Aug          Sep          Oct
1998          NA   448.973090   623.316840   130.202257  -684.068576
1999   511.233507  1049.139757   408.941840   -90.047743 -1109.276910
2000   543.816840  -545.651910    96.566840   496.910590  -602.860243
2001    49.441840  -961.610243   186.191840  -510.256076   844.806424
2002  -974.183160   588.431424  -561.391493   233.702257   997.639757
2003  -327.099826   595.806424 -1560.766493   589.493924           NA
             Nov          Dec
1998  -620.943576  -392.058160
1999  -225.651910   656.733507
2000  -364.818576  -652.141493
2001   695.806424  -245.558160
2002    24.973090   371.275174
2003          NA           NA
```

Come si può vedere vanno persi i dati relativi ad alcuni termini a causa della perequazione con la media mobile. Per ottenere il grafico della decomposizione della serie :

```
plot(dec.fit)
```

Un modo più raffinato che fornisce delle stime migliori è quello ottenuto con la funzione `stl()` che per la decomposizione usa il metodo ‘loess’.

```
stl.fit<-stl(ore,s.window="periodic")
```

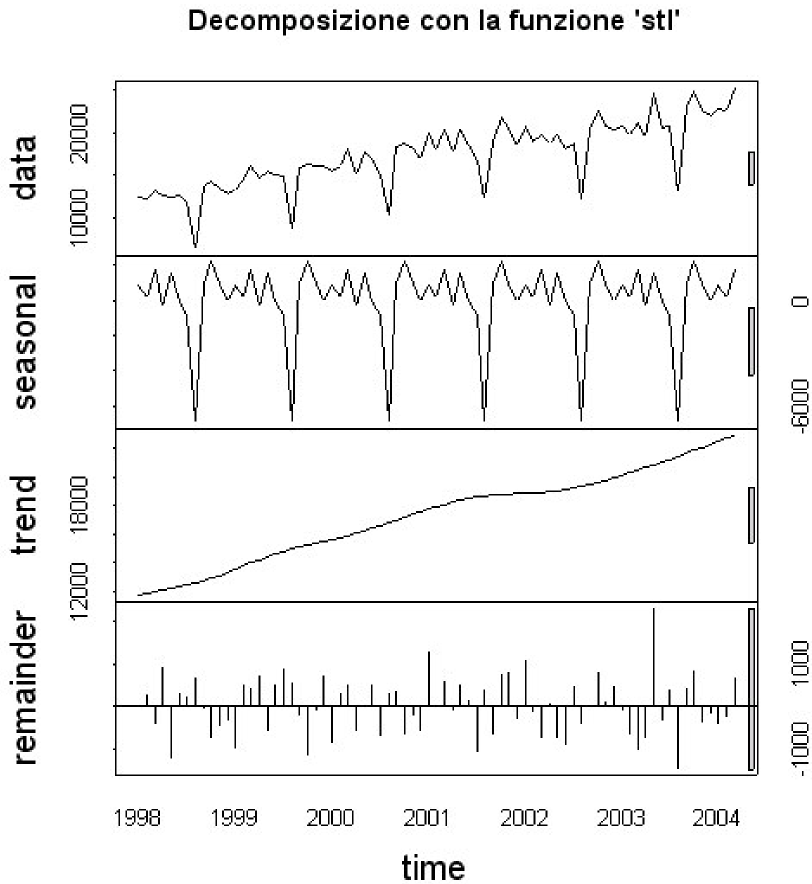
Tale funzione ha una pluralità di parametri (si veda l’help), ma ci soffermeremo solo su `s.window`. che serve per determinare il parametro per la stima della stagionalità basata su una perequazione di tipo loess, può assumere valori numerici oppure il valore stringa “periodic”. La funzione restituisce un oggetto della classe “stl” del quale i nomi delle componenti possono essere visualizzati digitando questo comando:

```
attributes(stl.fit)

$names
[1] "time.series" "weights"     "call"        "win"         "deg"
[6] "jump"        "inner"       "outer"

$class
[1] "stl"
```

in particolare `time.series` è una serie multipla le cui colonne contengono la stagionalità, il trend e i residui.

```
trend.stl<-stl.fit$time.series[,2]

trend.stl
          Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
1998 11696.49 11822.11 11947.73 12076.31 12204.88 12337.58 12470.28 12610.99
1999 13548.88 13774.23 13999.58 14220.23 14440.88 14631.89 14822.91 14974.05
2000 15601.13 15742.54 15883.95 16052.36 16220.78 16399.11 16577.43 16779.56
2001 17766.49 17923.88 18081.28 18232.29 18383.30 18507.33 18631.36 18694.74
```

```
2002 18853.16 18889.86 18926.55 18981.61 19036.66 19122.45 19208.23 19317.45
2003 20145.47 20315.01 20484.55 20652.54 20820.52 21011.95 21203.39 21417.00
2004 22494.01 22707.93 22921.85
          Sep      Oct      Nov      Dec
1998 12751.70 12927.89 13104.08 13326.48
1999 15125.20 15244.80 15364.40 15482.76
2000 16981.69 17191.13 17400.57 17583.53
2001 18758.11 18784.51 18810.90 18832.03
2002 19426.67 19593.63 19760.59 19953.03
2003 21630.61 21847.20 22063.80 22278.90
2004

stag.stl<-stl.fit$time.series[,1]
stag.stl
            Jan         Feb         Mar         Apr         May         Jun
1998   814.88473   143.90236  1676.06339  -328.88502  1532.99444   -71.78582
1999   814.88473   143.90236  1676.06339  -328.88502  1532.99444   -71.78582
2000   814.88473   143.90236  1676.06339  -328.88502  1532.99444   -71.78582
2001   814.88473   143.90236  1676.06339  -328.88502  1532.99444   -71.78582
2002   814.88473   143.90236  1676.06339  -328.88502  1532.99444   -71.78582
2003   814.88473   143.90236  1676.06339  -328.88502  1532.99444   -71.78582
2004   814.88473   143.90236  1676.06339
            Jul         Aug         Sep         Oct         Nov         Dec
1998  -858.40025 -6912.40024   960.93189  2186.29233   900.98628   -44.58393
1999  -858.40025 -6912.40024   960.93189  2186.29233   900.98628   -44.58393
2000  -858.40025 -6912.40024   960.93189  2186.29233   900.98628   -44.58393
2001  -858.40025 -6912.40024   960.93189  2186.29233   900.98628   -44.58393
2002  -858.40025 -6912.40024   960.93189  2186.29233   900.98628   -44.58393
2003  -858.40025 -6912.40024   960.93189  2186.29233   900.98628   -44.58393
2004

res.stl<-stl.fit$time.series[,3]
res.stl
             Jan          Feb          Mar          Apr          May
1998    -3.377557   264.984109  -428.797628   915.577446 -1239.875351
1999 -1022.766258   484.865535   429.353926   724.654077  -585.873643
2000  -884.012448   303.560205   516.989456  -593.479941   -16.777210
2001  1288.626502   -39.785812   568.658471   -99.401932   519.709793
2002  1080.956608  -138.758911  -757.617833    47.276834  -767.656370
2003  -109.355769  -672.915215 -1033.618064  -783.653012  2310.484169
2004  -434.889864  -261.829261   670.087939
             Jun          Jul          Aug          Sep          Oct
1998   296.204967   200.119461   672.408676   -54.634236  -754.183502
1999   484.893319   876.494456   564.346810  -234.132962 -1181.091996
2000   505.677335  -731.033945   279.839648   343.381114  -672.418986
2001   116.457711 -1098.960197   371.663988  -677.043953   757.201642
2002  -917.660378   461.169788  -443.050925    28.396236   802.076176
2003  -338.168488   367.013032 -1488.596230   432.462381   811.505412
2004
             Nov          Dec
1998  -468.066283  -347.896836
1999  -100.384543   725.820938
2000  -222.552600  -618.943615
2001   790.113723  -326.445400
2002    97.422602   446.552850
2003  -408.785071  -185.318033
2004

plot(stl.fit,main="Decomposizione con la funzione 'stl'")
```



Graf. 4

La libreria *ast* mette a disposizione un'ulteriore funzione per la decomposizione di una serie temporale: si tratta del comando `tsr()` che permette di stimare una serie di modelli del tipo:

(serie osservata) = f(trend, stagionalità, residuo)

che può essere adoperata sia modelli addittivi che per quelli moltiplicativi, inoltre prevede l'utilizzo di diversi stimatori per calcolare la componente di trend. Gli stimatori, detti *smoothers* (sovente tradotto in italiano con "lisciatori"), riconosciuti sono:

- *constant*: la serie "lisciata" assume valore uguale alla media della serie osservata per ogni istante di tempo;
- *poly(r)*: la serie osservata viene interpolata con un polinomio di grado r;
- *loess(r,g)*: la serie osservata è "lisciata" utilizzando una regressione locale di tipo loess; r è il grado del polinomio utilizzato; g è (approssimativamente) il numero di parametri equivalenti desiderati;
- *gauss(r,g)*: la serie osservata è "lisciata" utilizzando una regressione locale con pesi gaussiani (ovvero la funzione peso è la densità di una normale di media nulla); r è il grado del polinomio utilizzato; g è (approssimativamente) il numero di parametri equivalenti desiderati
- *spline(g)*: la serie osservata è "lisciata" utilizzando una spline con (approssimativamente) g parametri equivalenti.

Tale funzione risulta essere alquanto flessibile per le diverse possibilità offerta. La scelta dello stimatore migliore può essere fatta con l'ausilio dei grafici.

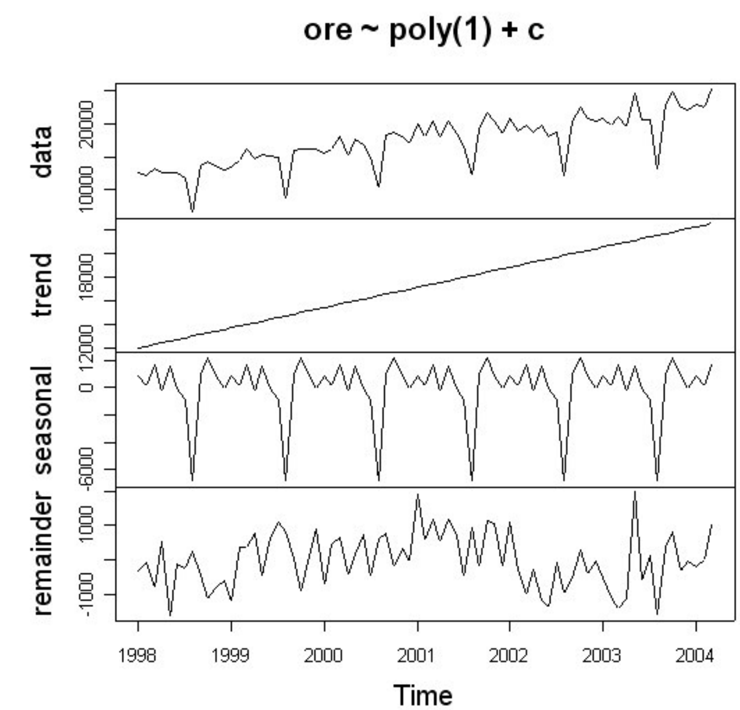
Utilizziamo la funzione `tsr()` per stimare le componenti della nostra serie di ore lavorate ipotizzando un trend lineare con stagionalità costante:

```
library(ast)
tsr.fit<-tsr(ore~poly(1)+c)
```

```
tsr.fit
Call=tsr(f = ore ~ poly(1) + c)
          data    trend    seasonal   remainder
Jan 1998 12508 12028.34   807.28037  -327.62112
Feb 1998 12231 12170.70   150.34524   -90.04969
Mar 1998 13195 12313.07  1696.55297  -814.62112
Apr 1998 12663 12455.43  -317.50949   525.07764
May 1998 12498 12597.80  1552.12681 -1651.92236
Jun 1998 12562 12740.16   -56.57022  -121.58903
….
Nov 2003 22556 21993.80   876.77795  -314.57764
Dec 2003 22049 22136.16   -51.41908   -35.74431
Jan 2004 22874 22278.53   807.28037  -211.80745
Feb 2004 22590 22420.89   150.34524    18.76398
Mar 2004 25268 22563.25  1696.55297  1008.19255
```

per aver una visualizzazione grafica del modello stimato (Graf. 5)

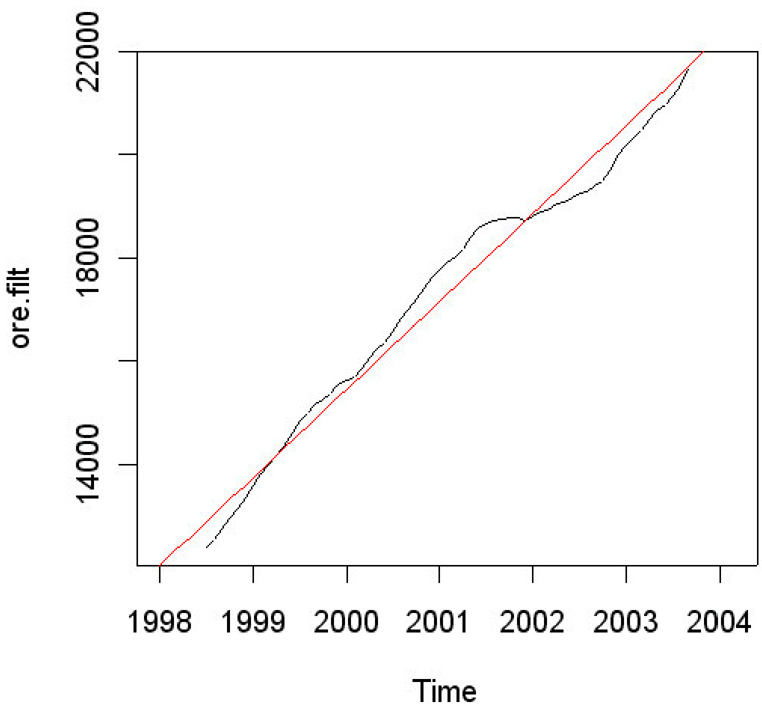
```
plot(tsr.fit)
```



Graf. 5

Proviamo a vedere il grado di adattamento di questo tipo trend tracciando il grafico della serie filtrata (`ore.filt`) sovrapposto al trend stimato con `tsr()` (Graf. 6)

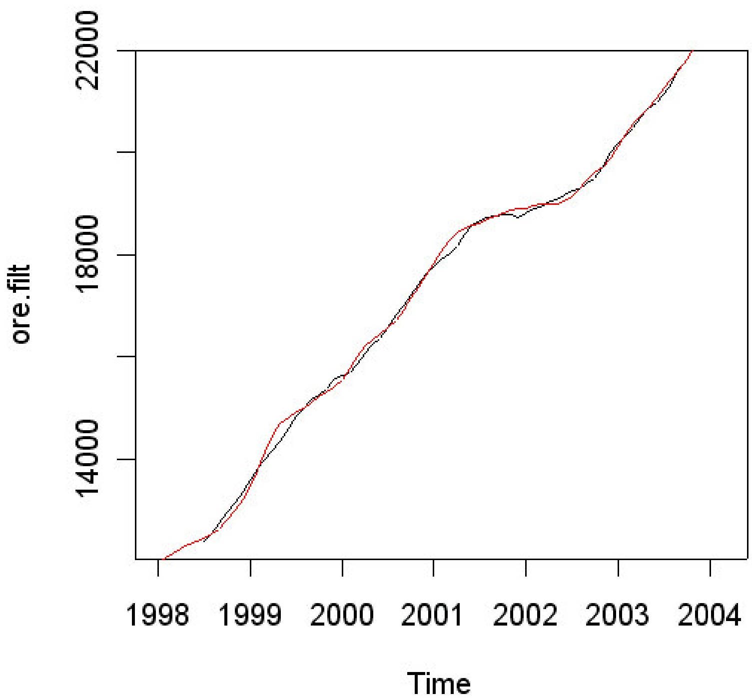
```
plot(ore.filt)
lines(trend(tsr.fit),col="red")
```



Graf. 6

Come si può facilmente vedere l'adattamento non è particolarmente buono. Ciò ci induce ad utilizzare un altro tipo di "lisciatore", ad esempio di tipo `loess(r,g)`.

```
tsr2.fit<-tsr(ore~loess(1,10))

plot(ore.filt)
lines(trend(tsr2.fit),col="red")
```

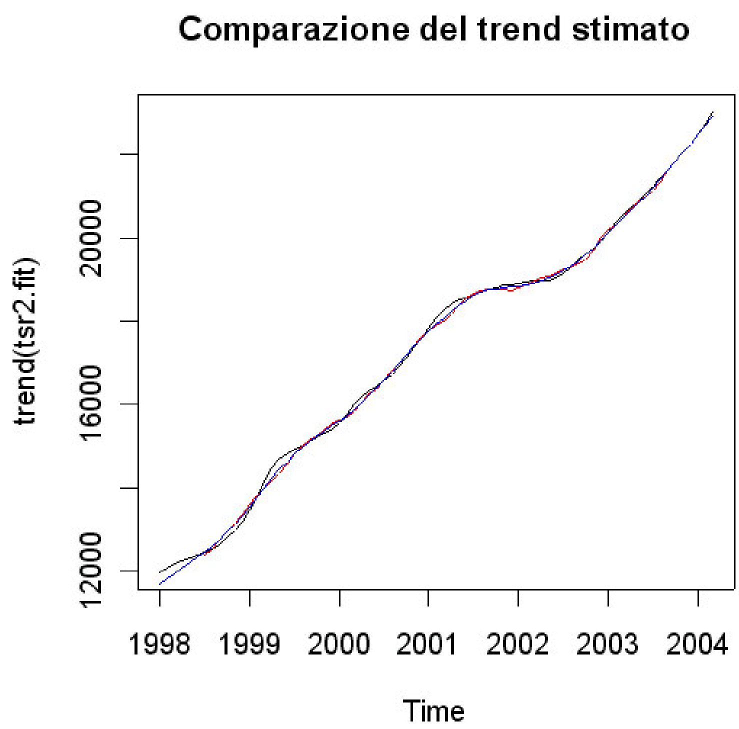


Graf. 7

Il risultato è decisamente migliore (Graf. 7). Per determinare il numero dei parametri equivalenti spesso si deve procedere per tentativi: in genere aumentando il numero dei parametri equivalenti il grado di adattamento migliora.
Confrontiamo i trend stimati con i tre metodi (`decompose()` in rosso, `stl()` in blu, `tsr()` in nero):

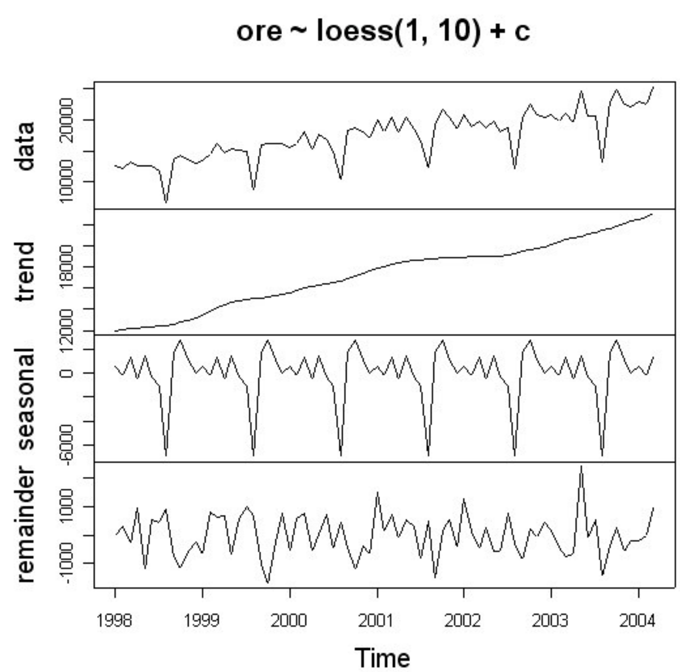
```
plot(trend(tsr2.fit),main="Comparazione del trend stimato")
lines(ore.filt, col="red")
lines(trend.stl, col="blue")
```

Come si vede I tre trend sono all'incirca tutti equali e nel grafico si sovrappongono abbondantemente (Graf. 7)



Graf. 8

```
plot(tsr2.fit)
```



Graf. 9

Ad un oggetto ottenuto come risultato della funzione `tsr()` si possono applicare una pluralità di metodi per estrarre le singole componenti:

`trend()`: per ottenere la stima del trend
`seasonal()`: per ottenere la stima della componente stagionale
`remainder()` o `residuals()`: per la stima della componente irregolare
`deseasonal()`: per ottenere la serie destagionalizzata
`detrend()`: per ottenere la serie detrendizzata
`fitted.tsr()`: per ottenere i valori "previsti" dal modello

```
trend.tsr<-trend(tsr2.fit)
trend.tsr
          Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
1998 11982.60 12073.72 12160.84 12242.49 12317.96 12385.39 12449.30 12514.56
1999 13453.69 13777.96 14147.02 14468.86 14685.00 14818.64 14916.62 14998.33
2000 15548.94 15761.10 15999.92 16201.12 16335.51 16440.96 16555.96 16708.39
2001 17830.82 18063.68 18278.36 18429.67 18515.23 18569.69 18617.39 18690.78
2002 18917.97 18948.40 18977.19 18980.72 18993.31 19040.89 19139.20 19294.69
2003 20123.59 20366.20 20568.04 20722.13 20876.81 21058.70 21263.31 21459.71
2004 22512.90 22758.23 23013.72
          Sep      Oct      Nov      Dec
1998 12628.87 12793.52 12994.38 13201.92
1999 15095.57 15204.71 15299.75 15396.44
2000 16910.34 17141.14 17364.16 17591.35
2001 18767.35 18823.36 18863.02 18893.86
2002 19461.96 19602.98 19735.18 19906.34
2003 21623.74 21826.77 22047.22 22276.14
2004

stag.tsr<-seasonal(tsr2.fit)
stag.tsr
             Jan         Feb         Mar         Apr         May         Jun
1998   544.51198  -167.02613  1322.14557  -521.07497  1390.87471  -344.77808
1999   544.51198  -167.02613  1322.14557  -521.07497  1390.87471  -344.77808
2000   544.51198  -167.02613  1322.14557  -521.07497  1390.87471  -344.77808
2001   544.51198  -167.02613  1322.14557  -521.07497  1390.87471  -344.77808
2002   544.51198  -167.02613  1322.14557  -521.07497  1390.87471  -344.77808
2003   544.51198  -167.02613  1322.14557  -521.07497  1390.87471  -344.77808
2004   544.51198  -167.02613  1322.14557
             Jul         Aug         Sep         Oct         Nov         Dec
1998 -1092.22988 -7025.17424  1782.98989  2751.69359  1088.11716   -13.32151
1999 -1092.22988 -7025.17424  1782.98989  2751.69359  1088.11716   -13.32151
2000 -1092.22988 -7025.17424  1782.98989  2751.69359  1088.11716   -13.32151
2001 -1092.22988 -7025.17424  1782.98989  2751.69359  1088.11716   -13.32151
2002 -1092.22988 -7025.17424  1782.98989  2751.69359  1088.11716   -13.32151
2003 -1092.22988 -7025.17424  1782.98989  2751.69359  1088.11716   -13.32151

res.tsr<-residuals(tsr2.fit)
res.tsr
              Jan          Feb          Mar          Apr          May
1998   -19.116912   324.304481  -287.987918   941.582317 -1210.830982
1999  -657.206086   792.067322   635.834776   668.210001  -687.875415
2000  -561.449053   595.922946   754.935820  -550.048037    10.614041
2001  1494.665685   131.343896   725.492656  -104.598211   529.891435
2002  1286.514556   113.630831  -454.334987   240.354292  -582.180431
2003   182.901060  -413.174951  -763.182078  -661.052598  2396.317977
2004  -183.416903    -1.202178   932.134079
              Jun          Jul          Aug          Sep          Oct
1998   521.386590   454.929991   881.617824  -753.860542 -1185.216653
1999   571.138676  1016.607644   652.845521 -1026.555877 -1706.407913
2000   736.816826  -475.735014   463.780515  -407.327009 -1187.837707
2001   327.089643  -851.160127   488.393134 -1508.343098   152.941439
2002  -563.115037   764.028715  -307.513698  -828.951464   227.328319
2003  -111.917372   540.916698 -1418.533172  -382.725466   266.540313
2004
```

```
   Nov          Dec
1998  -545.494838  -254.602238
1999  -222.868454   780.884574
2000  -373.272402  -658.033319
2001   550.864377  -419.542097
2002   -64.297317   461.985774
2003  -579.339935  -213.819674
2004
```

Un altro metodo per stimare trend e stagionalità può essere effettuato mediante la regressione con utilizzo della funzione `lm()`. Il modello da stimare è il seguente ed esprime il trend come funzione del tempo e la stagionalità come somma di funzioni trigonometriche:

$$Y_t = f(t) + \alpha \cos(\frac{2\pi t}{12}) + \beta \sin(\frac{2\pi t}{12}) + e_t$$

trasformiamo la serie storica in un vettore di dati:

```
ore.vec<-as.vector(ore)
```

creiamo il parametro tempo:

```
t<-seq(1,length(ore.vec))
t
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
[51] 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
```

creiamo le componenti seno e coseno

```
cos.t <- cos(2*pi*t/12)
sin.t <- sin(2*pi*t/12)
```

stimiamo il modello di regressione (ipotizzando un trend lineare)

```
gfit<-lm(ore.vec~t+cos.t+sin.t)

summary(gfit)

Call:
lm(formula = ore.vec ~ t + cos.t + sin.t)

Residuals:
    Min      1Q  Median      3Q     Max
-7251.4  -981.4   176.3  1501.0  3834.0

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 11865.31     519.22  22.852   <2e-16 ***
t             142.60      11.88  12.004   <2e-16 ***
cos.t         910.70     366.03   2.488   0.0152 *
sin.t         969.05     360.74   2.686   0.0090 **
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

Residual standard error: 2219 on 71 degrees of freedom
Multiple R-Squared: 0.6935,     Adjusted R-squared: 0.6805
F-statistic: 53.55 on 3 and 71 DF,  p-value: < 2.2e-16
```

Come si evince tutti i coefficienti di regressione risultano significativi.

```
ore.fit<-fitted(gfit)## sono i valori stimati con il modello di cui sopra
ore.fit.ts<-ts(ore.fit,start=1998,freq=12)## che ritrasformiamo in serie storica
```

e tracciamo il grafico del trend e della stagionità stimati e i dati originali (Graf. 10):

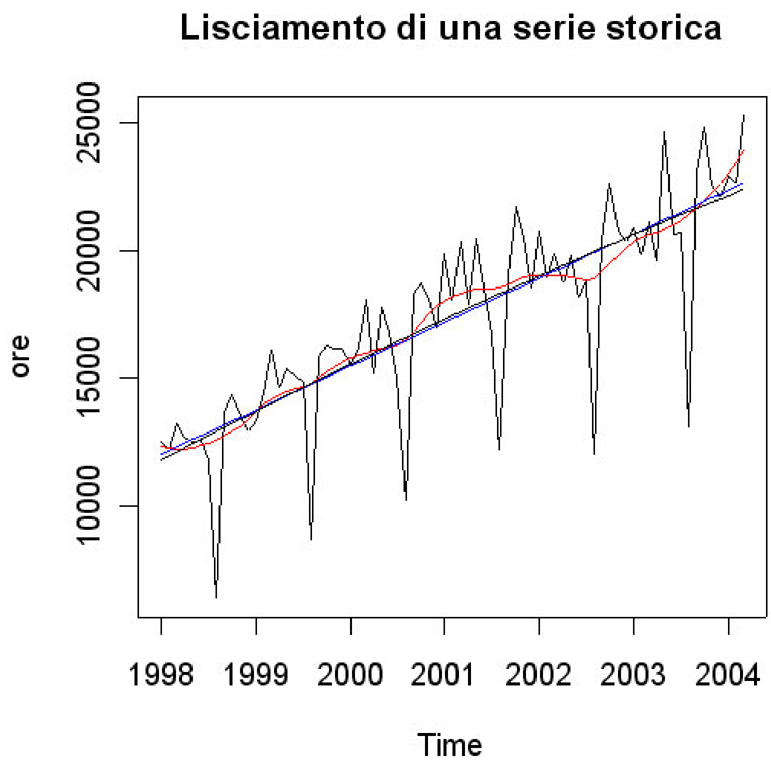
```
ts.plot(ore, ore.fit.ts, main="Stima trend e stagionalità con funzioni sin e cos")
```

Graf. 10

3.2 Lisciamento di una serie storica

La funzione `smoothts()` del package *ast* permette di "lisciare" una serie temporale in una varietà di modi attraverso l'impiego degli stessi identici stimatori visti per la funzione `tsr()`. Di seguito è riportato il grafico (Graf. 11) una serie di "lisciamenti" della serie storica delle ore lavorate.

```
library(ast)
plot(ore,main="Lisciamento di una serie storica") ## serie dei dati
lines(smoothts(ore~lo(2,10)),col="red")           ## lisciamento con loess
lines(smoothts(ore~s(2)),col="blue")              ## lisciamento con splines
lines(smoothts(ore~p(2)),col="black")             ## lisciamento con polinomio
```

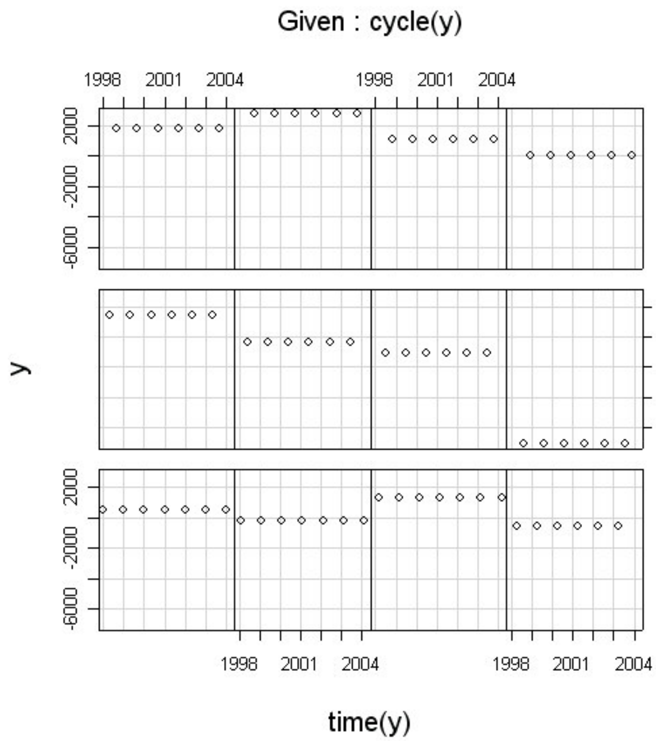


Graf. 11

3.3 Verifica dell'esistenza di un trend nella stagionalità

Una volta stimata la componente stagionale è necessario e opportuno verificare che la stagionalità nel corso degli anni non presenti un qualche trend, si parla in tal caso di trend-stagionalità. Un modo semplice ed immediato è quello di ricorrere alla funzione `seaplot()` del package *ast*. Nel nostro caso la stagionalità è costante in tutti i mesi, infatti (Graf. 12):

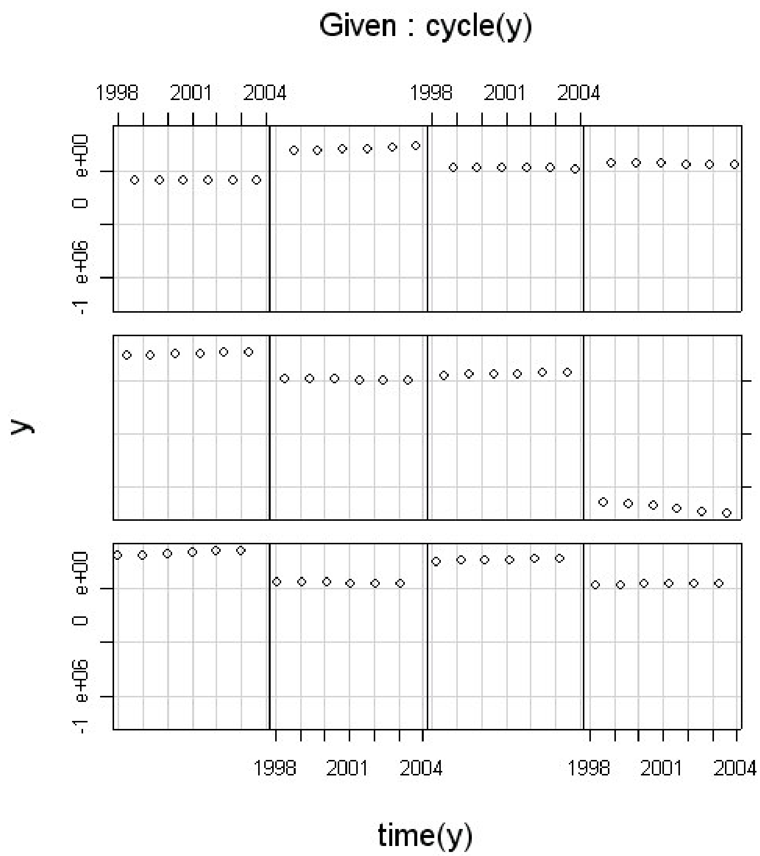
```
library(ast)
seaplot(stag.tsr)
```



Graf. 12

Consideriamo un altro caso caratterizzato dalla presenza di un trend nella componente stagionale. Abbiamo la serie storica della stagionalità dei ricavi mensili di un'azienda dal 1998 al 2003 stimata con il metodo `stl()`.

```
stag
             Jan          Feb          Mar          Apr          May
1998  305832.570    63988.695   259085.086    37671.141   226074.600
1999  314800.098    60130.203   261460.076    39102.193   230976.552
2000  323749.413    56208.390   263726.634    40360.695   235641.837
2001  335663.789    50584.322   272944.330    41407.662   244563.024
2002  347754.788    45185.859   282436.614    42771.344   253843.053
2003  356338.076    41234.904   285596.842    44596.976   258671.501
             Jun          Jul          Aug          Sep          Oct
1998   13503.620    40751.165 -1144710.631   -87685.750   185101.419
1999    9695.510    44226.226 -1161989.741   -88651.193   192097.697
2000    5574.878    47312.910 -1179725.480   -90141.517   198519.716
2001   -1936.937    52851.884 -1205743.760   -91989.080   207644.442
2002   -9055.056    58819.407 -1231306.213   -93353.535   217267.815
2003  -12956.536    62999.936 -1246429.933   -94717.882   223265.175
             Nov          Dec
1998   30926.172    64196.148
1999   29100.309    63889.942
2000   26650.806    62941.085
2001   22539.190    61143.968
2002   18941.762    59852.618
2003   17051.650    58990.911

library(ast)
seaplot(stag)
```



Graf. 13

Dal Graf. 13 emerge come in questo caso la stagionalità non è costante da un anno all'altro ma presenta un trend all'incirca lineare per quasi tutti i mesi dell'anno. In tale circostanza è necessario stimare questo trend-stagionalità attraverso la regressione lineare (modello di stagionalità variabile). Se la stagionalità oscilla intorno ad un dato valore costante, ed è probabile che tali oscillazioni sia di natura casuale, una stima della stagionalità di quel mese può essere ottenuta con la media delle componenti stagionali che quel mese ha presentato nel corso dei vari anni (modello di stagionalità costante).
Utilizzeremo delle funzioni costruite ad hoc, sfruttando la flessibilità di R, per stimare il trend della stagionalità. La prima funzione è la seguente:

```
estrai.stagionalita<-function(x.ts){
                   x.ts.lim<-window(x.ts,start=c(1998,1), end=c(2003,12))
                   y<-as.vector(x.ts.lim)
                   n.anni<-6
                   anni<-1998:2003
                   z<-matrix(y,ncol=n.anni,nrow=12)
                   trasp.z<-t(z)
                   df.z<-as.data.frame(trasp.z)
                   df.z<-cbind(anni,df.z)
                   colnames(df.z)<-
                  c(“anno”,"gen","feb","mar","apr","mag","giu","lug","ago","set"
                  ,"ott","nov","dic")
                   return(df.z)
                  }
```

Questa funzione consente di creare un dataframe partendo dalla serie storica della stagionalità le cui colonne sono i mesi e le righe gli anni.

```
st.orig<-estrai.stagionalita(stag) ## questa è la stagionalita originale
st.orig
  anno      gen      feb      mar      apr      mag        giu      lug
1 1998 305832.6 63988.69 259085.1 37671.14 226074.6  13503.620 40751.16
```

```
2 1999 314800.1 60130.20 261460.1 39102.19 230976.6   9695.510 44226.23
3 2000 323749.4 56208.39 263726.6 40360.69 235641.8   5574.878 47312.91
4 2001 335663.8 50584.32 272944.3 41407.66 244563.0  -1936.937 52851.88
5 2002 347754.8 45185.86 282436.6 42771.34 253843.1  -9055.056 58819.41
6 2003 356338.1 41234.90 285596.8 44596.98 258671.5 -12956.536 62999.94
       ago       set      ott      nov      dic
1 -1144711 -87685.75 185101.4 30926.17 64196.15
2 -1161990 -88651.19 192097.7 29100.31 63889.94
3 -1179725 -90141.52 198519.7 26650.81 62941.09
4 -1205744 -91989.08 207644.4 22539.19 61143.97
5 -1231306 -93353.54 217267.8 18941.76 59852.62
6 -1246430 -94717.88 223265.2 17051.65 58990.91
```

La seconda funzione consente di stimare le regressioni delle stagionalità dei 12 mesi sui vari anni e restituisce una lista con i coefficienti per le 12 regressioni calcolate. Come argomento di input viene passato il dataframe della stagionalità ottenuto in precedenza:

```
calcola.reg.stag<-function(st.df){
                                    attach(st.df)
                                    g1<-lm(gen~anno)
                                    g2<-lm(feb~anno)
                                    g3<-lm(mar~anno)
                                    g4<-lm(apr~anno)
                                    g5<-lm(mag~anno)
                                    g6<-lm(giu~anno)
                                    g7<-lm(lug~anno)
                                    g8<-lm(ago~anno)
                                    g9<-lm(set~anno)
                                    g10<-lm(ott~anno)
                                    g11<-lm(nov~anno)
                                    g12<-lm(dic~anno)
                              g<-list(g1,g2,g3,g4,g5,g6,g7,g8,g9,g10,g11,g12)
                              return(g)
                                    }

g<-calcola.reg.stag(st.orig)
g
[[1]]

Call:
lm(formula = gen ~ anno)

Coefficients:
(Intercept)         anno
  -20434842        10380


[[2]]

Call:
lm(formula = feb ~ anno)

Coefficients:
(Intercept)         anno
    9439581        -4692

…

[[12]]

Call:
lm(formula = dic ~ anno)
```

```
Coefficients:
(Intercept)         anno
    2344422        -1141
```

Se si vogliono estrarre i coefficienti delle 12 regressioni stimate si può usare questa funzione passando come argomento di input `g`:

```
estrai.coef<-function(reg){cf<-list()
                           for (i in 1:12)
                           cf[[i]]<-coef(reg[[i]])
                           return(cf)

                          }
coeff<-estrai.coef(g)
coeff
[[1]]
 (Intercept)         anno
-20434841.69     10380.17

[[2]]
(Intercept)        anno
9439580.840   -4692.173
```

…

```
[[12]]
(Intercept)        anno
2344422.025   -1141.008
```

Per ottenere un dataframe che contiene i coefficienti di regressione si può usare questa funzione:

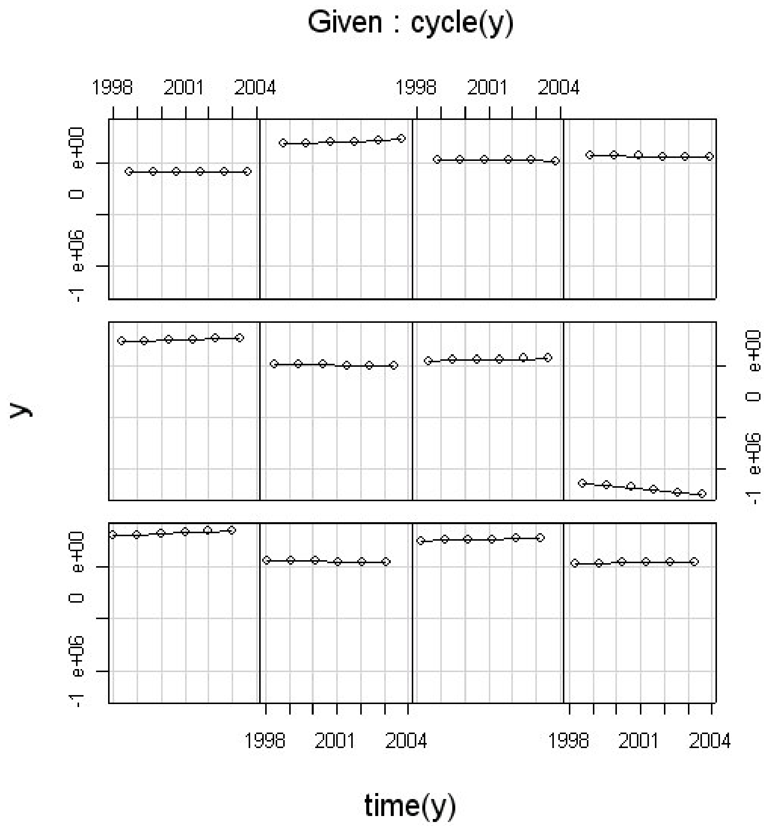
```
coef.df<-function(coeff){as.data.frame(do.call("rbind",coeff))}
```

La seguente funzione permette di ottenere la serie storica dei valori del trend della stagionalità stimato con le regressioni:

```
stima.stag.ts<-function(reg){ft<-matrix(ncol=12,nrow=6)
                                    st<-1998
                                    for (i in 1:12)
                                    ft[,i]<-fitted(reg[[i]])
                                    ft<-as.vector(t(ft))
                                    ft.ts<-ts(ft,start=c(st,1), frequency=12)
                                    return(ft.ts)
                              }
stag.tr<-stima.stag.ts(g)
stag.tr
            Jan          Feb          Mar          Apr          May
1998 304739.362    64619.161   256253.066    37650.459   224449.485
1999 315119.533    59926.988   262101.812    38984.276   231321.062
2000 325499.704    55234.815   267950.558    40318.093   238192.639
2001 335879.874    50542.642   273799.303    41651.910   245064.216
2002 346260.045    45850.469   279648.049    42985.727   251935.794
2003 356640.216    41158.296   285496.794    44319.544   258807.371
            Jun          Jul          Aug          Sep          Oct
1998  14808.839    39691.514 -1141943.992   -87438.737   184307.435
1999   9207.002    44279.010 -1163160.112   -88899.173   192177.545
2000   3605.165    48866.506 -1184376.233   -90359.609   200047.655
2001  -1996.672    53454.003 -1205592.353   -91820.044   207917.766
2002  -7598.509    58041.499 -1226808.473   -93280.480   215787.876
2003 -13200.346    62628.996 -1248024.593   -94740.916   223657.986
            Nov          Dec
```

```
1998    31627.353    64688.298
1999    28657.071    63547.290
2000    25686.789    62406.282
2001    22716.507    61265.275
2002    19746.225    60124.267
2003    16775.943    58983.259
```

```
seaplot(stag,fitted=stag.tr)
```



Graf. 14

Come si può vedere dal grafico (Graf. 14) il trend stagionalità può essere ben stimato con una retta. Un modo più generale per "interpolare" ciascuna sottoserie mensile con un opportuno stimatore, a seconda del tipo di trend, è quello di usare la funzione `smoothts()`.
Con l'ultima funzione che esaminiamo si ottiene una stima della stagionalità dei dodici mesi di un anno specificato come argomento assieme ai coefficienti delle 12 regressioni stimate:

```
prev.stagionalita<-function(coeff,anno){ prev<-vector()
                                          for (i in 1:12)
                                          prev[i]<-coeff[[i]][1]+
                                          coeff[[i]][2]*anno
                                          prev.ts<-
                                          ts(prev,start=c(anno,1),frequency=12)
                                          return(prev.ts)
                                          }
```

```
prev.stagionalita(coeff,2004)
            Jan         Feb         Mar         Apr         May         Jun
2004  367020.39    36466.12   291345.54    45653.36   265678.95   -18802.18
            Jul         Aug         Sep         Oct         Nov         Dec
2004   67216.49 -1269240.71   -96201.35   231528.10    13805.66    57842.25
```

Ci viene fornita in tal modo una stima della componente stagionale relativa ai dodici mesi del 2004.

4. Tests di specificazione

Nel paragrafo precedente si è supposto che la componente erratica fosse distribuita normalmente con media pari a zero, varianza costante (omoschedasticità) e che non vi fosse autocorrelazione. Tali ipotesi alla base del modello vanno opportunamente verificate con opportuni test statistici detti test di specificazione del modello. Il venire meno di una di queste ipotesi potrebbe inficiare la validità del modello adottato e far propendere per altri modelli più complessi oppure di intervenire sulla serie con delle trasformazioni atte, ad esempio, a stabilizzare la varianza oppure ad eliminare l'autocorrelazione. Tali tests risultano necessari anche nell'ambito dell'approccio stocastico delle serie storiche.

4.1 Verifica sul valore della media degli errori

In primo luogo occorre verificare che la media dei residui non sia significativamente diversa da zero. Occorre effettuare il test t. Se n è il numero di osservazioni della serie allora la media degli errori osservati è:

$$\overline{e} = \frac{\sum_t e_t}{n}$$

e la varianza campionaria corretta:

$$s^2 = \frac{\sum_t (e_t - \overline{e})^2}{n-1}$$

il test da effettuare è:

$$t = \frac{\overline{e}}{s/\sqrt{n}}$$

che si distribuisce come una t di Student con n-1 gradi di libertà.

Per la nostra esemplificazione possiamo prendere i residui stimati con uno dei tre metodi di decomposizione visti precedentemente. Consideriamo quelli ottenuti con la funzione `stl()`:`res.stl`.
Calcoliamo la media dei residui:

```
media.residui<-mean(res.stl)
media.residui
[1] -7.824039
```

calcoliano il numero delle osservazioni presenti nella serie:

```
n<-length(res.stl)
n
[1] 75
```

e la varianza campionaria corretta e lo scarto quadratico medio:

```
var.residui<-(n/(n-1))*var(res.stl)
var.residui
[1] 474884.8

s<-sqrt(var.residui)
```

Ora possiamo determinare il valore del test t:

```
test.t<-(media.residui/(s/sqrt(n)))
test.t
[1] -0.0983258
```

il cui p-value è pari a:

```
pt(test.t,n-1,lower.tail=F)
[1] 0.5390303
```

oppure considerando il valore soglia del test per un livello di significatività al 99%:

```
qt(0.99,n-1)
[1] 2.377802
```

che ci consente di concludere che la media degli errori non è significativamente diversa da zero.

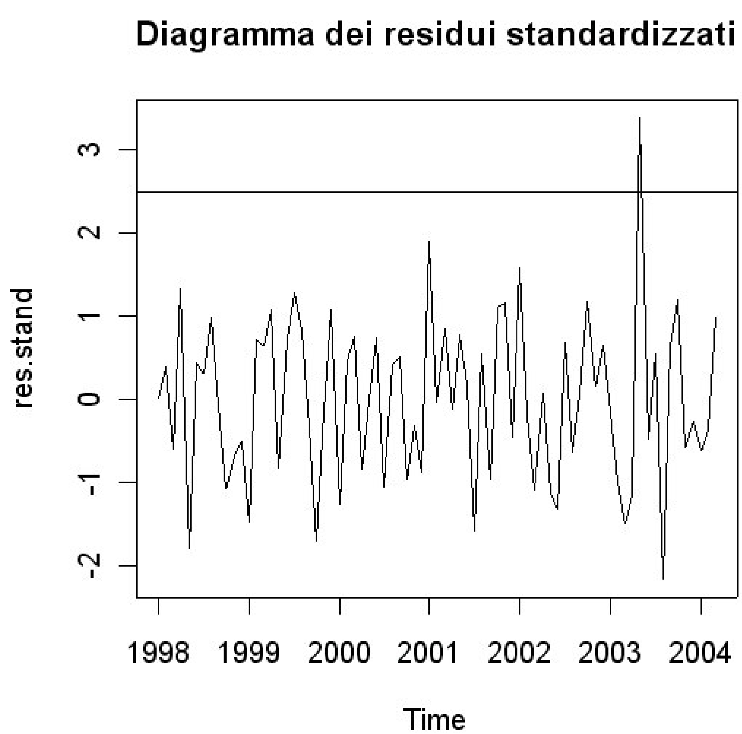
4.2 Test di normalità degli errori

Prima di procedere a verificare che i residui si distribuiscano secondo una variabile aleatoria normale, andiamo ad esaminare i residui al fine di individuare dei valori anomali. È preferibile operare sui residui standardizzati per avere a che fare con numeri puri. A tal scopo andiamo a scrivere una semplice funzione che standardizza il vettore dei dati in input:

```
stand<-function(x){m=mean(x)
+ s=(var(x)^0.5)
+ z=(x-m)/s
+ return(z)}
```

Creiamo il vettore dei residui standardizzati e ne tracciamo il grafico (Graf. 15):

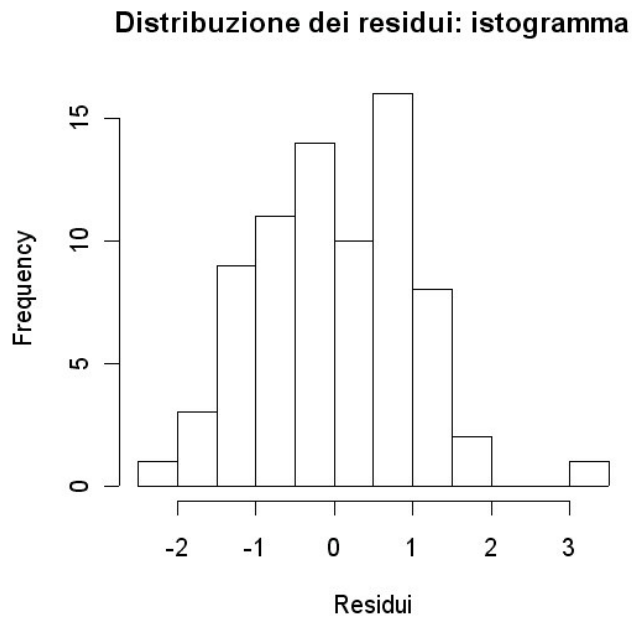
```
res.stand<-stand(res.stl)
plot(res.stand,main="Diagramma dei residui standardizzati")
abline(h=2.5)
```



Graf. 15

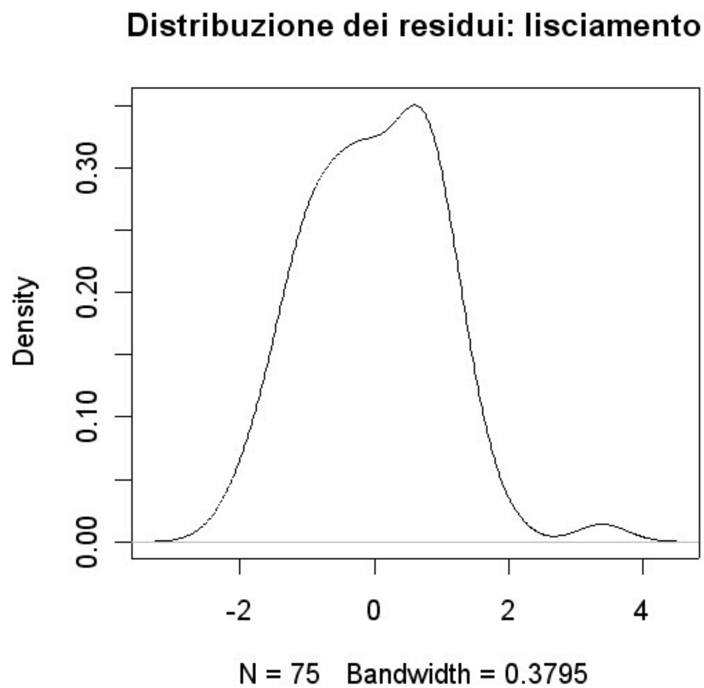
Dal diagramma emerge che una sola osservazione (maggio 2003) risulta essere anomala poiché si trova al di là della banda di confidenza del 99% (banda compresa tra –2,5 e +2,5). È necessario analizzare questo outlier e comprendere quali motivazioni lo hanno determinato, ed esempio potrebbe essere stato causato da un fatto episodico o comunque circoscritto ed individuabile, ed in caso estremo potrebbe essere necessario eliminarlo. Dal grafico può supporsi che agli errori sottenda un processo stocastico di tipo white noise.
Un modo abbastanza semplice ed intuitivo per verificare la normalità della distribuzione degli errori è quello di ricorre all'ausilio grafico con un istogramma (Graf. 16 e 17) e con un QQ-plot (Graf. 18).

```
hist(res.stand,main="Distribuzione dei residui: istogramma",xlab="Residui")
```



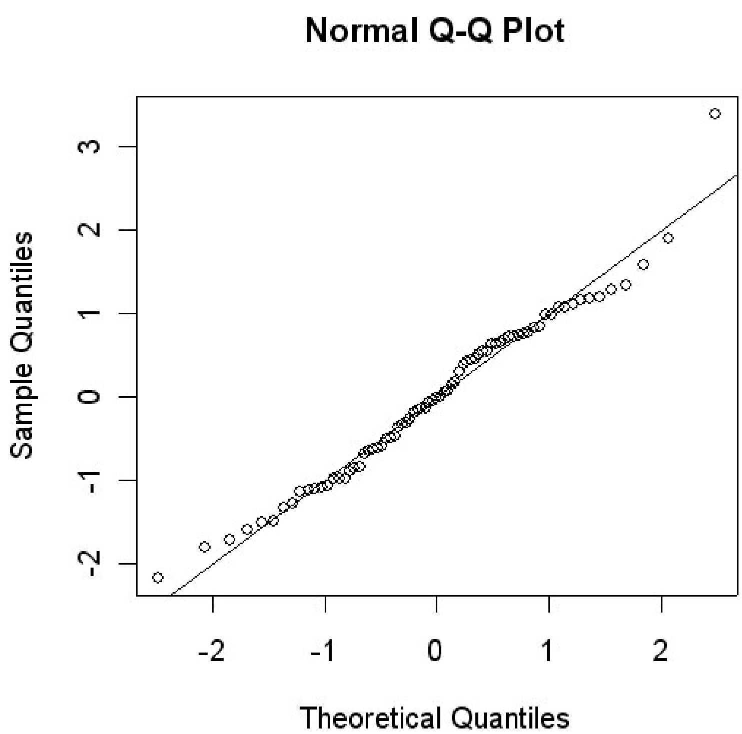
Graf. 16

```
plot(density(res.stand,kernel="gaussian"),main="Distribuzione    dei    residui: 
lisciamento")
```



Graf. 17

```
qqnorm(res.stand)
abline(0,1)
```



Graf. 18

Tutti e tre i grafici ci danno una buona indicazione per una probabile distribuzione pressoché normale dei residui. Per avere un risultato statisticamente più affidabile bisogna effettuare dei test di normalità. Ne effettueremo due: quello di Wilk – Shapiro e quello di Jarque – Bera. Per completezza informiamo che il package *nortest* fornisce ulteriori test statistici per verificare la normalità.

4.2.1 Test di Wilk – Shapiro

Il test di Shapiro-Wilk è considerato in letteratura uno dei test più potenti per la verifica della normalità, soprattutto per piccoli campioni.
La verifica della normalità avviene confrontando due stimatori alternativi della varianza σ^2: uno stimatore non parametrico basato sulla combinazione lineare ottimale della statistica d'ordine di una variabile aleatoria normale al numeratore, e il consueto stimatore parametrico, ossia la varianza campionaria, al denominatore. I pesi per la combinazione lineare sono disponibili su apposite tavole. La statistica W può essere interpretata come il quadrato del coefficiente di correlazione in un diagramma quantile-quantile. Il comando per effettuare il test di normalità in questione in ambiente R è `shapiro.test()` presente nel package stats. Esso restituisce come risultato il valore della statistica W e il relativo p-value:

```
shapiro.test(res.stand)

        Shapiro-Wilk normality test

data:  res.stand
W = 0.9785, p-value = 0.2299
```

Il p-value è decisamente elevato rispetto ai livelli di significatività a cui di solito si fa riferimento: ciò ci fa propendere per l'ipotesi nulla ovvero la normalità della distribuzione degli errori.

4.2.2 Test di Jarque- Bera

Il test di Jarque – Bera è impiegato molto spesso per la verifica dell'ipotesi di normalità in campo econometrico. Esso si basa sulla misura dell'asimmetria e della curtosi di una distribuzione. Si considera in particolare la distribuzione asintotica di una combinazione dei noti coefficienti b3 e b4 (o gamma3 e gamma4) che è di tipo chi-quadro.
In R tale test è presente nel package *tseries* ed è richiamabile tramite il comando `jarque.bera.test()` che restituisce il valore della statistica, i gradi di libertà e il p-value:

```
jarque.bera.test(res.stand)

        Jarque Bera Test

data:  res.stand
X-squared = 1.5368, df = 2, p-value = 0.4638
```

Anche in questa circostanza il p-value è sufficientemente elevato per impedirci di rifiutare l'ipotesi di normalità della distribuzione dei residui.

4.3 Test di Breusch – Pagan per omoschedasticità

Il test di Breusch-Pagan, largamente utilizzato in econometria per verificare l'ipotesi di omoschedasticità, applica ai residui gli stessi concetti della regressione lineare. Esso è valido per grandi campioni, assume che gli errori siano indipendenti e normalmente distribuiti e che la loro varianza (σ^2_t) sia funzione lineare del tempo t secondo:

$$\ln((\sigma^2_t) = a+bt$$

ciò implica che la varianza aumenti o diminuisca al variare di t, a seconda del segno di b. Se si ha l'omoschedasticità, si realizza l'ipotesi nulla:

$$H_0 : b = 0$$

Contro l'ipotesi alternativa bidirezionale:

$H_0 : b \neq 0$

Per la sua verifica, si calcola una regressione lineare, a partire da un diagramma di dispersione (Graf. 15) che:

- sull'asse delle ascisse riporta il tempi t
- sull'asse delle ordinate il valore dei residui corrispondente

Si ottiene una retta di regressione, la cui devianza totale (SQR) è in rapporto alla devianza d'errore precedente (SQE) calcolata con i dati originari secondo una relazione di tipo quadratico che, se è vera l'ipotesi nulla, al crescere del numero delle osservazioni si distribuisce secondo una variabile casuale chi-quadro con un grado di libertà.

Per effettuare il test di Breusch Pagan in R è necessario caricare il package *lmtest* ove è contenuta la funzione `bptest()`.

```
library(lmtest)
```

Possiamo effettuare la verifica in tre modi diversi:

a) facendo regredire i valori delle ore lavorare rispetto al trend e alla stagionalità stimati secondo un modello del tipo: ore ~ -1+trend+stagionalita, senza considerare l'intercetta:

```
bp.test<- bptest(formula(ore~-1+trend.stl+stag.stl))
bp.test

        studentized Breusch-Pagan test

data:  formula(ore ~ -1 + trend.stl + stag.stl)
BP = 0.1473, df = 1, p-value = 0.7011
```

b) facendo regredire gli errori su una costante:

```
bp.test2<-bptest(res.stl~1, varformula=~-1+trend.stl+stag.stl)
bp.test2

        studentized Breusch-Pagan test

data:  res.stl ~ 1
BP = 0.1416, df = 1, p-value = 0.7067
```

c) per verificare se la varianza dei residui è monotona crescente o decrescente nel tempo si può procedere in questo modo:

```
Y<- as.numeric(res.stl)
X<-1:nrow(stl.fit$time.series)
bptest(Y~X)

        studentized Breusch-Pagan test

data:  Y ~ X
BP = 1.2403, df = 1, p-value = 0.2654
```

L'output fornisce il valore della statistica del test (BP), il numero dei gradi di libertà (df) e il p-value corrispondente. Poiché il numero delle osservazioni è abbastanza elevato il test risulta applicabile e, in tutti i modelli esaminati, risulta non essere significativo (il p-value è di gran lunga superiore al livelli significatività di solito utilizzati nell'analisi statistica) e quindi di accetta l'ipotesi nulla, ossia di varianza dei residui costante nel tempo.

4.4 Test di autocorrelazione

Si può avere un fenomeno di autocorrelazione temporale, a causa dell'inerzia o stabilità dei valori osservati, per cui ogni valore è influenzato da quello precedente e determina in parte rilevante quello successivo.
Esistono diversi test statistici per saggiare la presenza di una correlazione seriale dei residui di una serie storica. In questa sede si farà riferimento ai test di Box-Pierce, Ljung-Box Tests e Durbin-Watson.

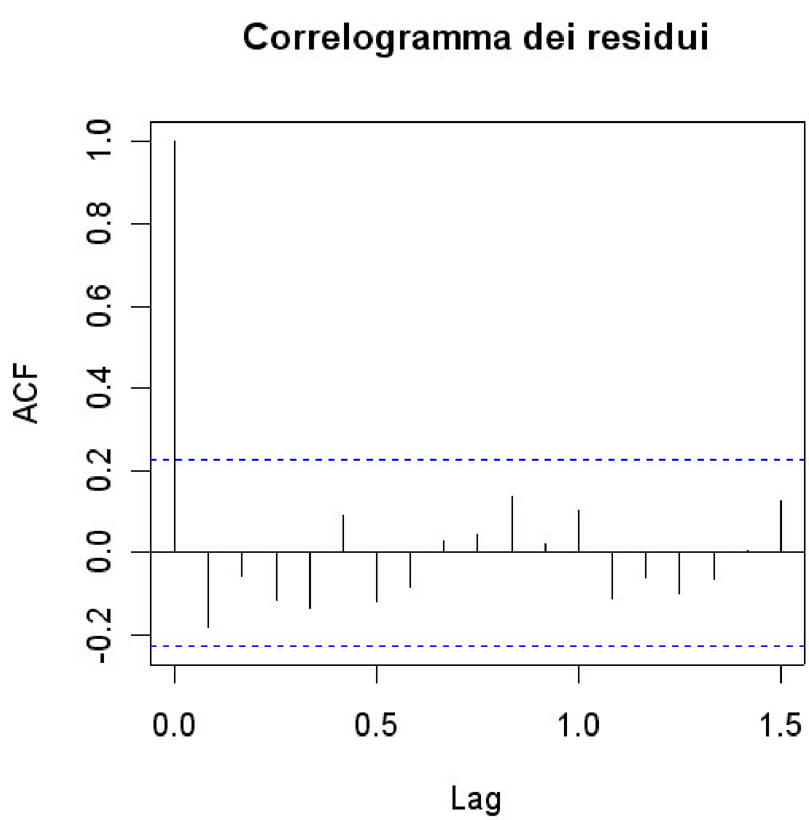
4.4.1 Uso del correlogramma

Un modo abbastanza semplice per vedere se una serie presenza autocorrelazione è quella di tracciarne il correlogramma con la funzione `acf()`. In caso di assenza di autocorrelazione la distribuzione asintotica della stima del coefficiente di autocorrelazione è di tipo normale ed avremo una banda di confidenza del tipo:

$$\left[-z_{1-\alpha/2}/\sqrt{n};z_{1-\alpha/2}/\sqrt{n}\right]$$

valori esterni a tale intervallo indicano la presenza di autocorrelazione significativa. Tracciamo quindi il correlogramma dei residui standardizzati (Graf. 19):

```
acf(res.stand,main="Correlogramma dei residui")
```



Graf. 19

Le linee tratteggiate di colore azzurro indicano la banda di confidenza ad un livello del 95%. Al variare del lag temporale i coefficienti di autocorrelazione dei residui risultano essere tutti interni alla banda di confidenza, indicando quindi assenza di correlazione serie.

4.4.2 Test di Ljung-Box e di Box-Pierce

Una statistica che può essere utilizzata per verificare l’assenza di autocorrelazione è una opportuna combinazione lineare dei coefficienti di autocorrelazione dei residui r(t):

$$LB = n(n+2)\sum_{t=1}^{k}\frac{r^2(t)}{n-t}$$

dove k è un intero prescelto. Se è vera l'ipotesi nulla (assenza di autocorrelazione) la statistica LB si asintoticamente secondo una variabile casuale chi-quadro con k gradi di libertà. Un test analogo a quello di Ljung – Box è quello che si basa sulla statistica proposta da Box – Pierce:

$$BP = n\sum_{t=1}^{k} r^2(t)$$

Le due statistiche differiscono semplicemente per il diverso sistema di ponderazione adoperato, ma asintoticamente hanno la medesima distribuzione. Si dimostra, tuttavia, che LB ha una convergenza più rapida alla distribuzione asintotica e, per tale motivo, risulta preferibile al test di Box-Pierce.
Verifichiamo che nella nostra serie di residui standardizzati non vi sia correlazione seriale prima con il test di Ljiung-box e poi con quello di Box-Pierce:

```
lb<-Box.test(res.stand, lag = 12, type ="Ljung-Box") ## test di Ljung-Box
lb

        Box-Ljung test

data:  res.stand
X-squared = 11.1595, df = 12, p-value = 0.5153

bp<-Box.test(res.stand, lag = 12, type ="Box-Pierce") ## test di Box-Pierce
bp

        Box-Pierce test

data:  res.stand
X-squared = 10.1069, df = 12, p-value = 0.6066
```

Come output dei risultati calcolati da `Box.test()` possiamo vedere la statistica utilizzata (Box-Pierce oppure Ljung-Box), il relativo valore (X-squared), i corrispondenti gradi di libertà (df) e il livello di significatività osservato (p-value). Si noti in particolare che il secondo argomento della funzione è costituito dal numero dei coefficienti di autocorrelazione che si vogliono considerare nella statistica e che diventano i gradi di libertà della statistica stessa. Il terzo argomento serve per specificare il tipo di test che si vuole effettuare. In entrambi i casi il risultato del test non ci consente di rifiutare l'ipotesi nulla, concludiamo per tanto sostenendo che non i residui della nostra serie storica non sono autocorrelati.

4.4.3 Test di di Durbin-Watson

Un ulteriore modalità per valutare l'esistenza di autocorrelazione tra i residui è rappresentata dal test di Durbin-Watson, che saggia l'ipotesi nulla di assenza di autocorrelazione tra i residui. È un test di solito adoperato per la verifica sui residui della regressione, ma è anche applicabile ai residui di una serie storica. La statistica test è:

$$d = \frac{\sum_{t=1}^{n}(e_t - e_{t-1})^2}{\sum_{t=1}^{n}(e_t)^2}$$

Nell'ambiente R è disponibile il comando `dwtest()` presente nel package *lmtest*. Effettuiamo il test sui residui standardizzati seguendo due strade:

1) facendo regredire i valori osservati della serie storica della ore lavorate sul trend e la stagionalità, senza considerare l'intercetta ed effettuare il test di Durbin e Watson sui residui di tale regressione:

```
dw.test<- dwtest(formula(ore~-1+trend.stl+stag.stl), alternative="greater")
dw.test

        Durbin-Watson test

data:  formula(ore ~ -1 + trend.stl + stag.stl)
DW = 2.3563, p-value = 0.9397
alternative hypothesis: true autocorrelation is greater than 0
```

2) facendo regredire i residui su una costante :

```
dw.test2<-dwtest(res.stand~1,alternative="two.sided" )
dw.test2

        Durbin-Watson test

data:  res.stand ~ 1
DW = 2.3563, p-value = 0.1185
alternative hypothesis: true autocorelation is not 0
```

La funzione richiede come argomento una formula che costituisce il modello del quale si vuole verificare l'assenza di autocorrelazione dei residui e il tipo di ipotesi alternativa specificata nel parametro `alternative` che può assumere i valori: "`greater`" nel caso di ipotesi alternativa unidirezionale destra, "`two.sided`"in caso di ipotesi alternativa bidirezionale e "`less`" in caso di ipotesi unidirezionale sinistra. L'output del test ci fornisce il modello testato, il valore della statistica, il livello significatività e l'ipotesi alternativa scelta. In ambo i casi esaminati si conferma l'assenza di autocorrelazione dei residui.
Concludendo possiamo affermare che la componente erratica della nostra serie storica risulta soddisfare tutti i requisiti richiesti: normalità, omoschedasticità e assenza di correlazione seriale. Essa può quindi assimilarsi ad un processo stocastico detto *white noise* (rumore bianco) gausssiano.
Per completezza sui tests di specificazione si ritiene opportuno segnalare che nel package *fSeries* sono contenuti ulteriori tests statistici che vanno al di là della portata di questo manuale. Per visualizzare l'aiuto in linea di tali funzioni digitare dalla linea di comando:

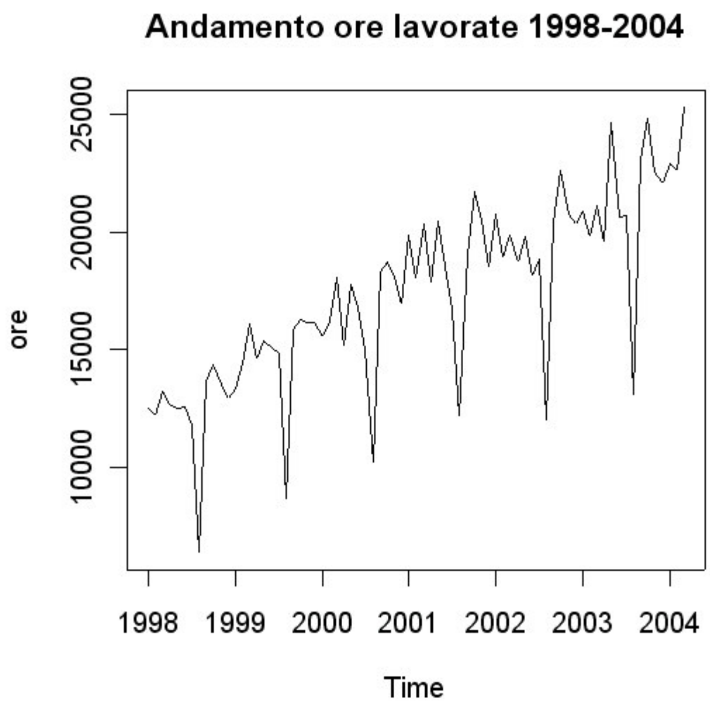
```
library (fSeries)
? TseriesTests
? RegressionTests
```

Naturalmente occorre aver prima scaricato dal CRAN il package in oggetto.

5 Grafici

Nei precedenti paragrafi abbiamo avuto già modo di far ricorso in diverse circostanze a dei grafici, strumenti assai utili nell'analisi delle serie storiche.
Per poter ottenere il grafico di una singola serie temporale (Graf. 20) si utilizza il comando `plot()` presente nel package "*graphics*":

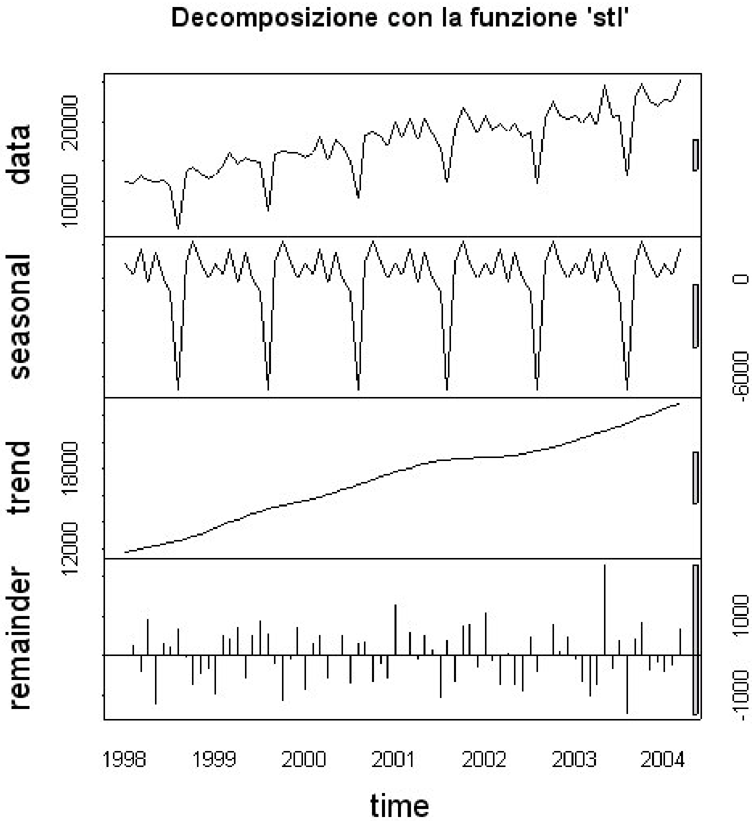
```
plot(ore,main="Andamento ore lavorate 1998-2004")
```



Graf. 20

Questa funzione consente di ottenere i grafici di qualsiasi oggetto di classe ts (Graf. 21):

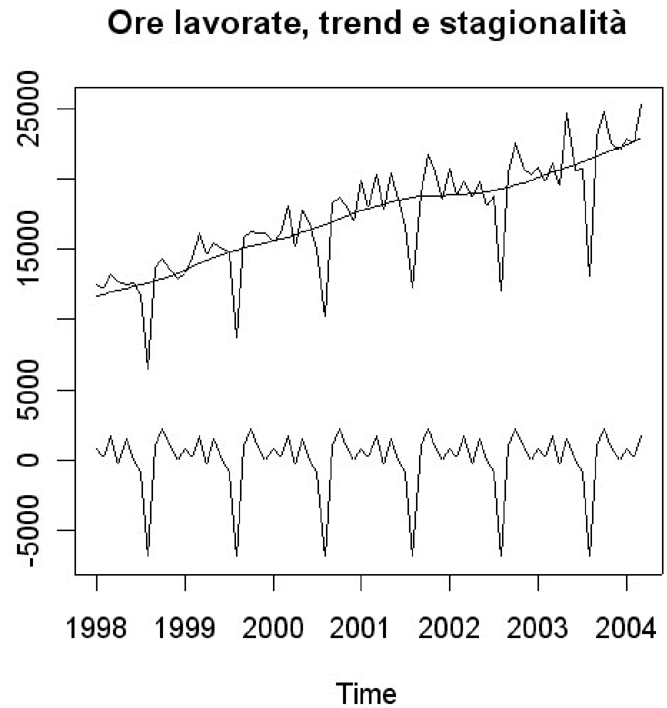
```
plot(stl.fit,main="Decomposizione con la funzione 'stl'")
```



Graf. 21

Per rappresentare in un unico grafico più serie storiche riferite anche ad archi temporali differenti, ma tutte con la stessa periodicità (ad esempio le serie devono essere tutte annuali, oppure tutte mensili, oppure tutte trimestrali, ma possono avere inizio e/o fine diversi da una serie all'altra) si usa il comando `ts.plot()` (Graf. 22):

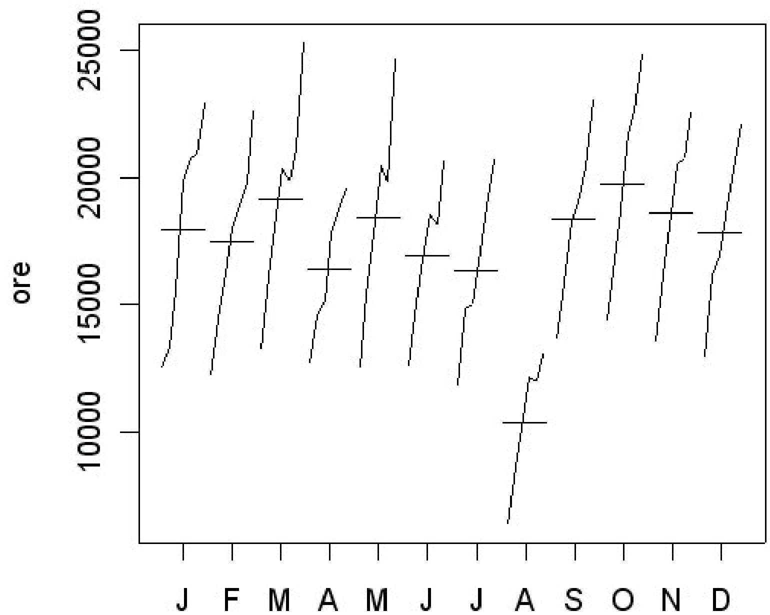
```
ts.plot(ore,trend.stl,stag.stl,main="Ore lavorate, trend e stagionalità")
```



Graf. 22

Nel package "*tseries*" esiste il comando `seqplot.ts()` che consente di tracciare nel medesimo grafico due serie storiche con inizio e/o fine differenti, ma con la stessa periodicità temporale.
La funzione `monthplot()` permette di creare il grafico della serie storica (Graf. 23) o di una sua componente stimata (trend, stagionalità, residui) (Graf. 24) considerando l'evoluzione del fenomeno nei 12 mesi (o altre frazioni) dell'anno.

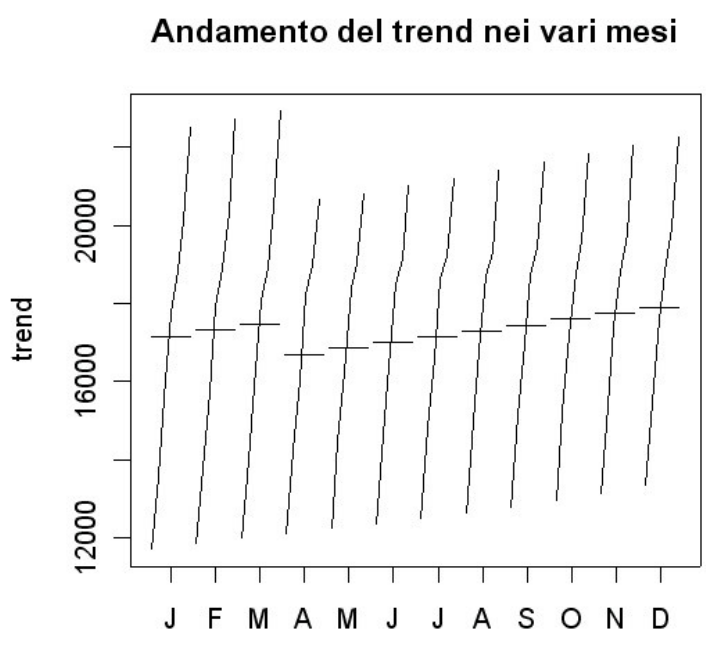
```
monthplot(ore)
```



Graf. 23

oppure considerando la stima del trend con il comando `stl()` calcolato in precedenza e memorizzato nell'oggetto `stl.fit`:

```
monthplot(stl.fit,choice="trend",main="Andamento del trend nei vari mesi")
```

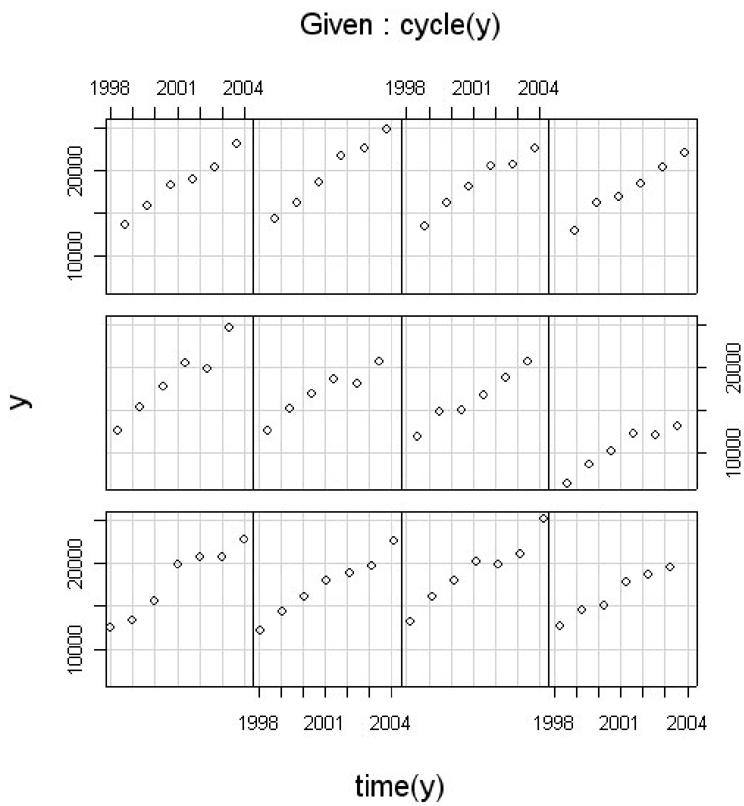


Graf. 24

per visualizzare la stagionalità occorre settare `choice=”seasonal”`, mentre per rappresentare i residui `choice=”remainder”`.

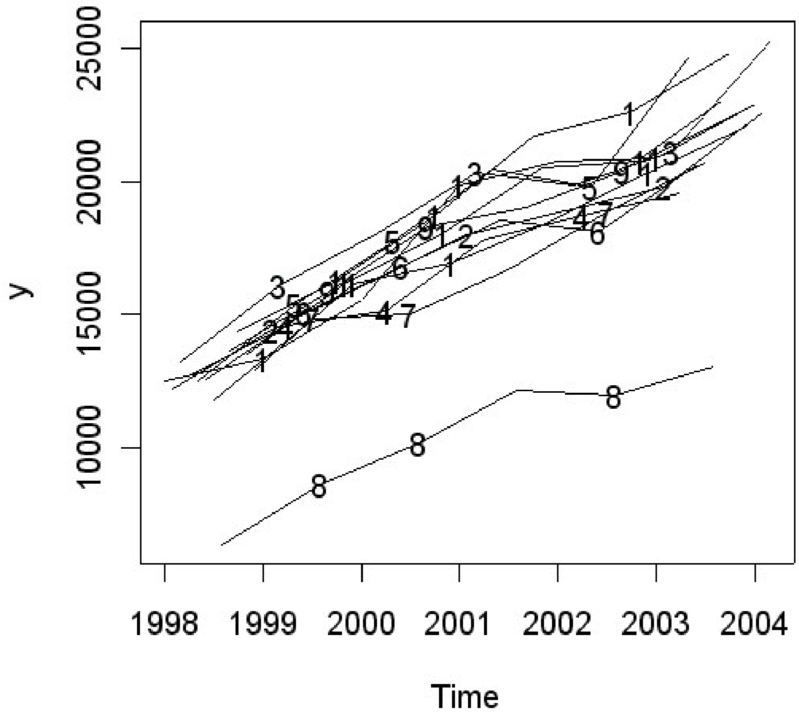
Un comando analogo al `monthplot()` è `seaplot()` fornito dal package “*ast*” che consente di ottenere il grafico delle serie mensili dei dati. Essa ha due parametri, oltre alla serie temporale della quale si sta tracciando il grafico: `type` che può assumere valore `“multiple”` e ciascuna serie mensile viene rappresentata in un proprio grafico (Graf. 25), oppure `“single”` e tutte le serie mensili sono rappresentate in un unico grafico (Graf. 26), oppure `“profile”` e in questa circostanza è tracciato, per ciascun anno, il grafico del profilo stagionale (Graf. 27); `fitted` è usato per tracciare il grafico di valori stimati (Graf. 28).

```
library(ast)
seaplot(ore,type="multiple")
```



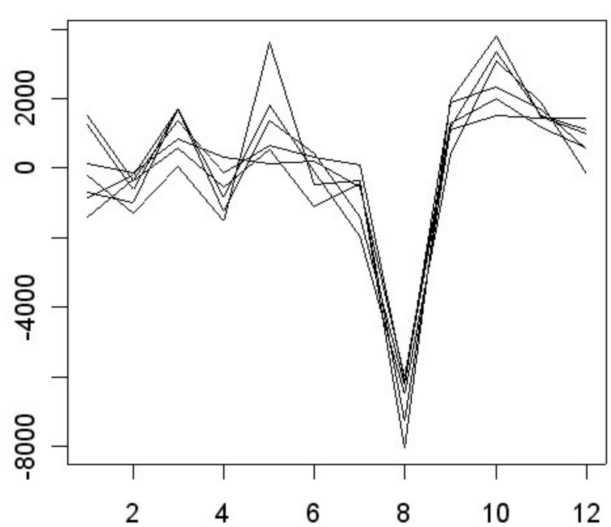
Graf. 25

```
seaplot(ore,type="single")
```



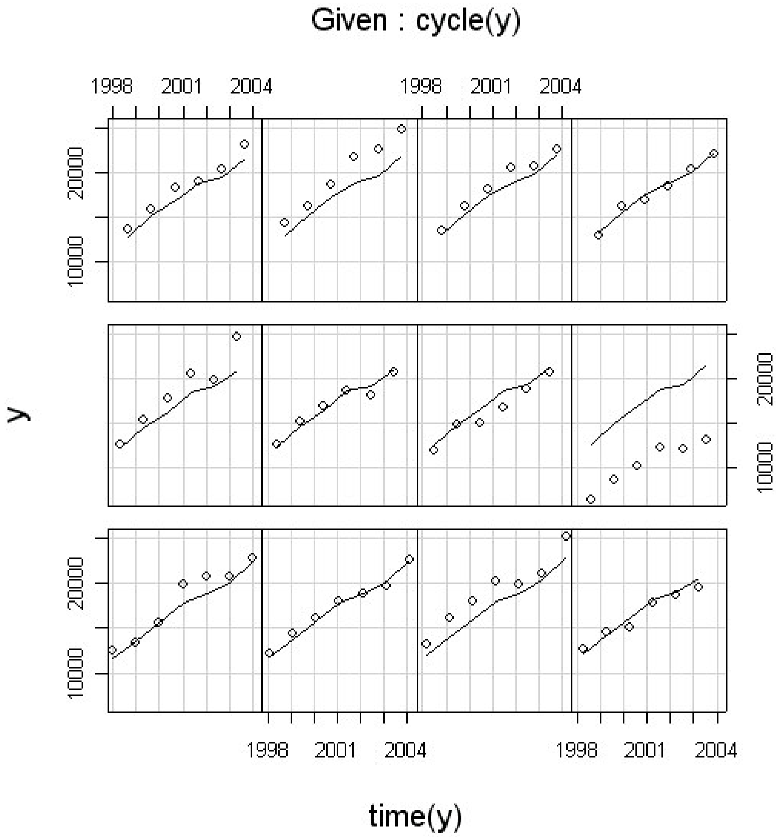
Graf. 26

```
seaplot(ore,type="profile")
```



Graf. 27

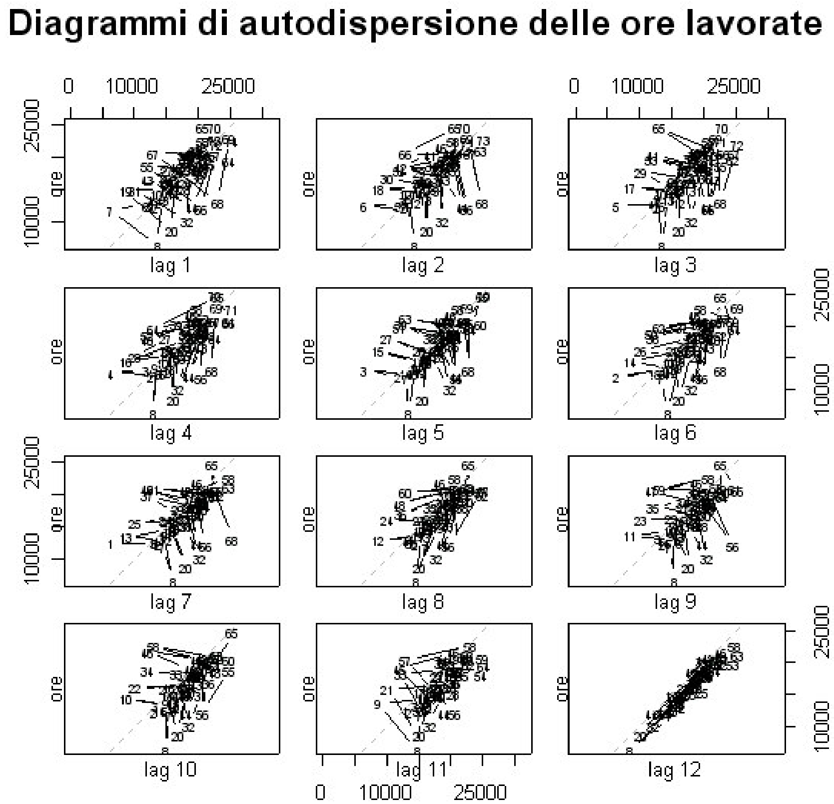
```
seaplot(ore,type="multiple",fitted=trend.stl)
```



Graf. 28

Un metodo grafico per verificare se una serie storica presenta autocorrelazione è quello di tracciare il grafico di dispersione tra la serie originaria e la stessa serie ritardata di un certo lag. Tale grafico (Graf. 29), detto spesso di autodispersione, si può ottenere nell'ambiente R con il comando `lag.plot()`:

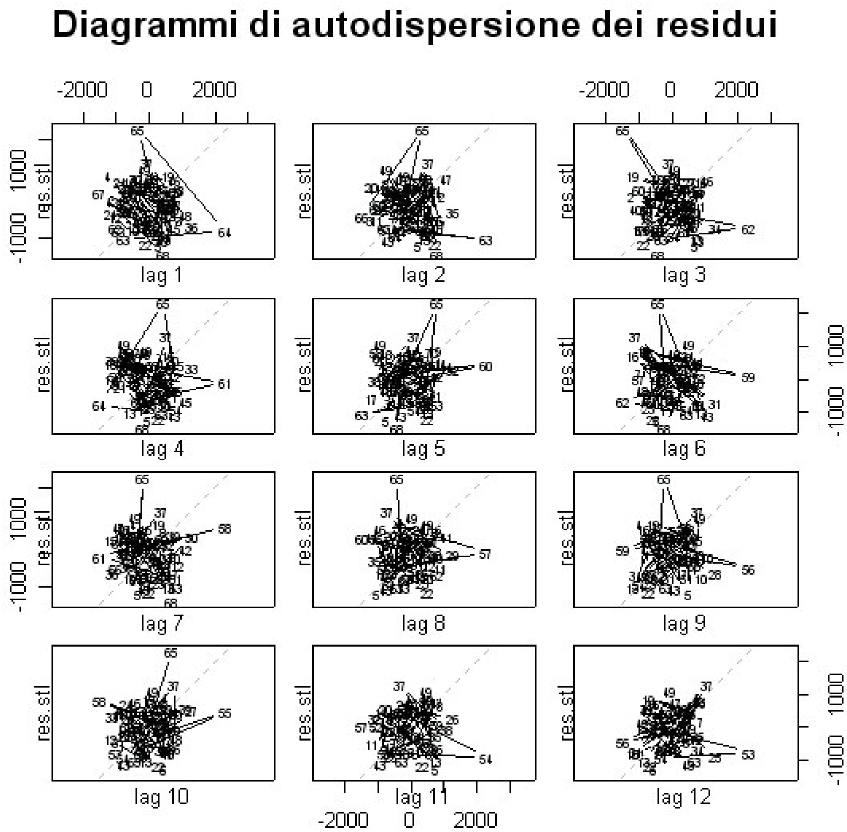
```
lag.plot(ore,lags=12,main="Diagrammi di autodispersione delle ore lavorate")
```



Graf. 29

Il parametro `lags` consente di specificare il numero di ritardi per i quali si vogliono tracciare i grafici. Nella fattispecie, trattando di una serie con periodicità mensile, abbiamo esaminato le serie ritardate da 1 a 12 mesi e ne è stato fornito il relativo grafico di autospersione. Come si può facilmente notare esiste un legame forte tra la serie originale e quella ritardata di 12 mesi, del resto era da aspettarsi un tale risultato.
Se consideriamo invece la serie dei residui, il relativo grafico di autodispersione mostra assenza di legami tra la serie originaria e quelle ritardate (Graf. 30):

```
lag.plot(res.stl,12,main="Diagrammi di autodispersione dei residui")
```



Graf. 30

Per inciso ricordiamo che con il comando `lag()` è possibile ottenere una serie ritardata di un dato lag (specificato come argomento del comando) partendo dalla serie originale.

6 Modelli stocastici

6.1 Introduzione

L'approccio moderno o stocastico (contrapposto a quello classico o deterministico) si basa sul concetto mutuato dal calcolo delle probabilità di processo stocastico.
Definiamo il processo stocastico come una famiglia (o successione) di variabili aleatorie $X_t(\Omega)$ definite sullo stesso spazio degli eventi Ω e ordinate secondo un parametro t (appartenente allo spazio parametrico T) che nell'analisi delle serie storiche è il tempo. In questo modo la serie storica può considerarsi come una realizzazione campionaria finita del processo stocastico. Utilizzeremmo la notazione X_t per indicare il processo stocastico e la notazione x_t per riferirci ad una sua determinazione, ovvero ad una serie storica.
Del processo stocastico possono considerarsi alcuni indicatori di sintesi o valori caratteristici:

1) Media di X_t: $E(X_t) = \mu_t$ dove $t \in T$
2) Varianza X_t: $E[X_t - E(X_t)]^2 = E[X_t - \mu_t]^2 = \sigma_t^2$ dove $t \in T$
3) Covarianza tra X_t e X_{t-k}: $Cov(X_t, X_{t-k}) = E[(X_t - \mu_t)(X_{t-k} - \mu_{t-k})] = \gamma(t,t-k)$ dove $t \in T, t-k \in T$
4) Correlazione tra X_t e X_{t-k}: $\dfrac{\gamma(\mathrm{t, t-k})}{\sqrt{\sigma_{\mathrm{t}}^2 \sigma_{\mathrm{t-k}}^2}} = \rho(t, t-k)$ dove $t \in T, t-k \in T$

Di solito si fa riferimento ad una particolare categoria di processi stocastici: quelli stazionari in senso debole (si trascura in tale contesto la stazionarietà in senso forte). Un processo si dice stazionario in senso debole:

in media se $\mathrm{E}(\mathrm{X}_t) = \mu < \infty \quad \forall t \in T$ (ossia la media è costante ed è finita e non dipende dal parametro t)

in varianza se $\mathrm{E}(\mathrm{X}_t - \mu)^2 = \sigma^2 < \infty \quad \forall t \in T$ (ossia la varianza è costante ed è finita e non dipende da t) in covarianza se Cov(X_t, X_{t-k}) = E[(X_t -μ)(X_{t-k} -μ)] = γ(k) = γ_k $\forall t \in T$ (ossia le covarianze sono finite e dipendono solo dal ritardo temporale k)

Se k=0 si che γ(0) = σ^2 ; la funzione { γ(k), k>0} è detta *funzione di autocovarianza* del processo stazionario. Un processo è stazionario in senso debole tout court se è stazionario in media, varianza e covarianza. In processo stazionario anche la funzione di correlazione dipende solo dal lag temporale k, infatti:

$$\frac{\gamma(\mathrm{k})}{\sqrt{\sigma_{\mathrm{t}}^2\sigma_{\mathrm{t-k}}^2}} = \frac{\gamma(k)}{\sqrt{\sigma^2\sigma^2}} = \frac{\gamma(k)}{\gamma(0)} = \rho(k) = \rho_k$$

ρ_k , essendo un coefficiente di correlazione, varia tra –1 e 1. Se k=0 chiaramente ρ_0 =1. La funzione { ρ(k), k>0} è detta *funzione di autocorrelazione* del processo stazionario.
Per poter applicare l'approccio moderno all'analisi delle serie storiche è necessario che queste vengano rese stazionarie eliminando il trend e la stagionalità con i metodi esposti nei paragrafi precedenti. È sulla serie dei residui, quindi, che si dovrà operare, magari dopo aver applicato alcuni tests di specificazione, come ad esempio quello per verificare la omoschedasticità, ossia la stazionarietà in varianza.
Di un processo stocastico stazionario X_t si possono stimare i valori caratteristici con i dati delle n osservazioni della serie storica x_t:

1) media aritmetica temporale: $\hat{\mu} = \frac{1}{n}\sum_{t=1}^{n} x_t$ che è uno stimatore corretto della media del processo stocastico;
2) varianza temporale: $\hat{\sigma}^2 = \hat{\gamma}(0) = \frac{1}{n}\sum_{t=1}^{n}(x_t - \hat{\mu})^2$
3) autocovarianza: $\hat{\gamma}(k) = \frac{1}{n}\sum_{t=1}^{n-k}(x_t - \hat{\mu})(x_{t+k} - \hat{\mu})$
4) autocorrelazione: $\hat{\rho}(k) = \frac{\hat{\gamma}(k)}{\hat{\gamma}(0)}$

Applichiamo questi concetti alla serie dei residui standardizzati stimati con il metodo `stl()` contenuti nel vettore `res.stl`. Calcoliamo la stima della media del processo:

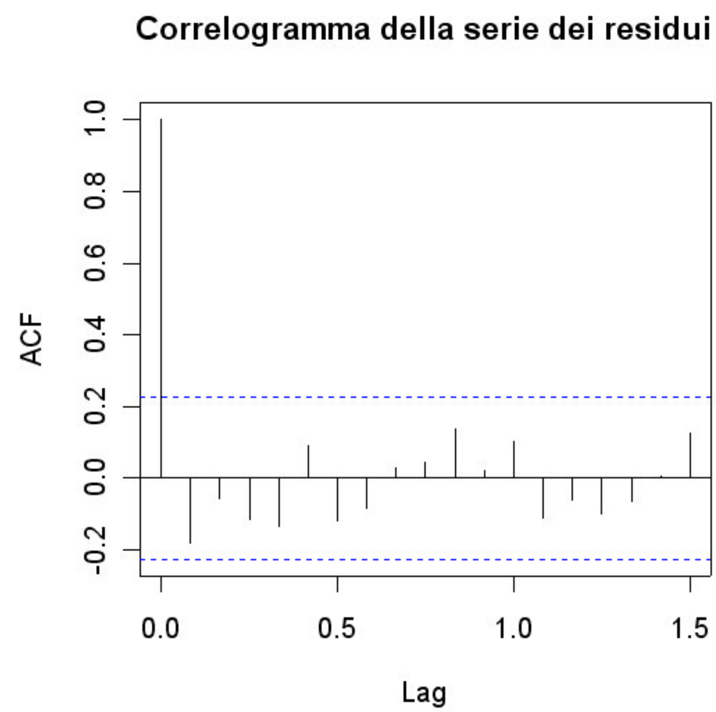
```
mean(res.stl)
-7.824039
```

e della varianza temporale:

```
var(res.stl)
468553
```

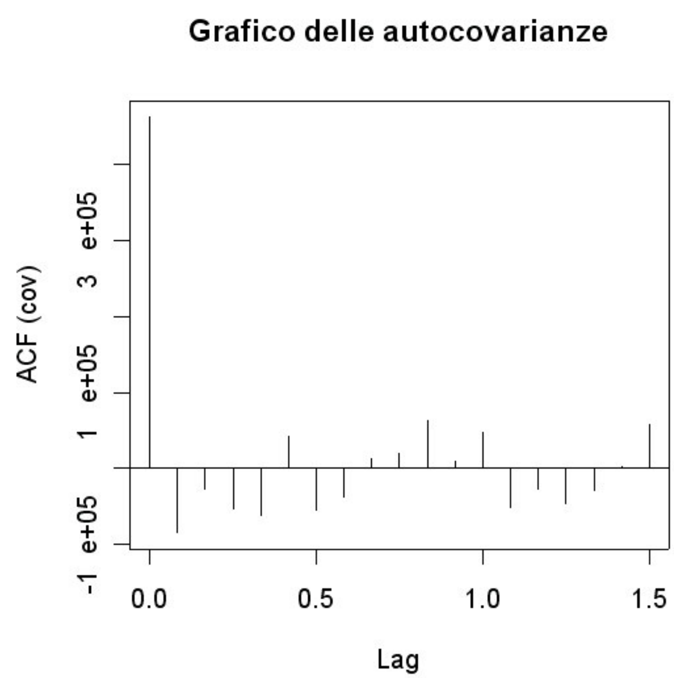
Per quanto riguarda le funzioni di autocovarianza e di autocorrelazione si usa il comando `acf()` che permette di ottenere il correlogramma (Graf.31) o il grafico delle autocovarinaze (Graf. 32) oppure la semplice stampa delle autocovarianze o dei coefficienti di autocorrelazione:

```
acf(res.stl,type="correlation",plot=TRUE,main="Correlogramma della serie dei residui")
```



Graf.31

```
acf(res.stl,type="covariance",plot=TRUE, main="Grafico delle autocovarianze")
```



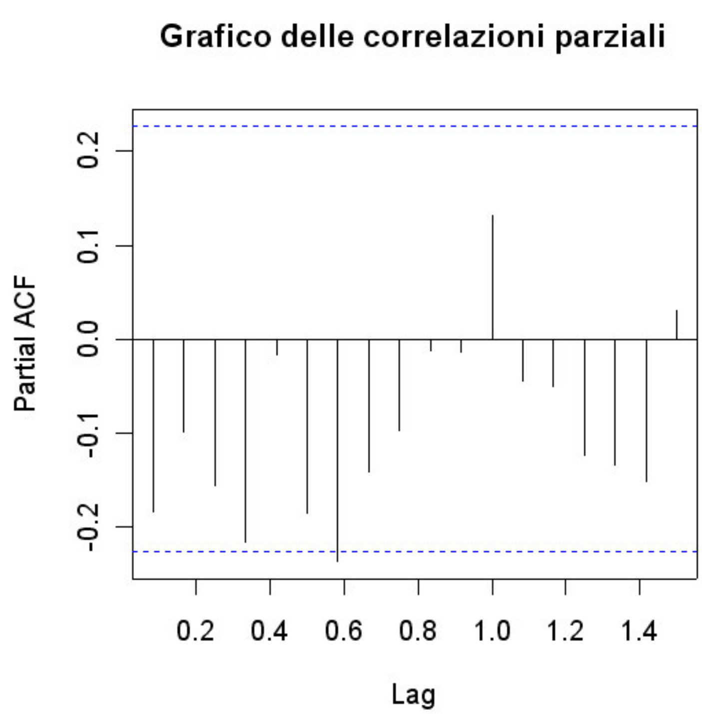
Graf. 32

I principali argomenti di `acf()`, oltre alla serie storica, sono `type`, con il quale ci specifica se si vogliono calcolare le autocovarianze (`type="covariance"`) o i coefficienti di correlazione `type="correlation"`) e `plot` che serve per indicare se tracciare il grafico (`plot=TRUE`) che è il valore di default oppure se stampare l'output (`plot=FALSE`)

```
acf(res.stl,type="correlation",plot=FALSE)

Autocorrelations of series 'res.stl', by lag

0.0000 0.0833 0.1667 0.2500 0.3333 0.4167 0.5000 0.5833 0.6667 0.7500 0.8333
 1.000 -0.185 -0.063 -0.119 -0.140  0.092 -0.124 -0.087  0.027  0.042  0.135
0.9167 1.0000 1.0833 1.1667 1.2500 1.3333 1.4167 1.5000
 0.021  0.102 -0.115 -0.064 -0.103 -0.068  0.006  0.124

acf(res.stl,type="covariance",plot=FALSE)

Autocovariances of series 'res.stl', by lag

0.0000 0.0833 0.1667 0.2500 0.3333 0.4167 0.5000 0.5833 0.6667 0.7500 0.8333
462306 -85433 -28996 -55091 -64518  42457 -57438 -40245  12409  19603  62323
0.9167 1.0000 1.0833 1.1667 1.2500 1.3333 1.4167 1.5000
  9599  47206 -53163 -29761 -47588 -31225   2795  57294
```

che fornisce i lag e I corrispondenti coefficienti di autocorrelazione o le autocovarianze. Con il comando `acf()` è possibile stimare anche i coefficienti di correlazione parziale specificando l'argomento `type="partial"`. Analogo risultato si ottiene con il comando `pacf()`.
La correlazione ρ(k) tra dati distanti k lag è influenzata dalle relazioni lineari con i dati intermedi. La funzione di autocorrelazione parziale π(k) misura la correlazione tra x_t e x_{t+k} dopo che sia stata eliminata la parte "spiegabile linearmente" da x_{t+1}, x_{t+2}, . . . , x_{t+k-1} . E' una misura dei legami lineari tra x_t e x_{t+k} depurata dall'influenza delle variabili che stanno in mezzo. Per stimare le π(k) si può far ricorso a formule che le esprimono in funzione della autocorrelazioni ordinarie oppure a procedure ricorsive.

```
pacf(res.stl,plot=TRUE,main="Grafico delle correlazioni parziali")
```



Graf. 33

```
pacf(res.stl,plot=FALSE)
```

```
Partial autocorrelations of series 'res.stl', by lag

0.0833 0.1667 0.2500 0.3333 0.4167 0.5000 0.5833 0.6667 0.7500 0.8333 0.9167
-0.185 -0.100 -0.157 -0.218 -0.018 -0.187 -0.238 -0.143 -0.098 -0.014 -0.015
1.0000 1.0833 1.1667 1.2500 1.3333 1.4167 1.5000
 0.131 -0.046 -0.053 -0.125 -0.136 -0.154  0.031
```

La funzione `ccf()` consente di calcolare le correlazioni o le covarianze incrociate tra due serie storiche.

6.2 Esame di alcuni processi stocastici

Prendiamo ora in esame alcuni processi stocastici particolarmente ricorrenti:

1) processo IID $\{\varepsilon_t,\ t \in T\}$
con ε_1, ε_3, ε_3 … variabili aleatorie IID con le seguenti caratteristiche:
a) $E(\varepsilon_t) = 0 \quad \forall t \in T$
b) $Var(\varepsilon_t) = \sigma_\varepsilon^2 < \infty \ \forall t \in T$ (omoschedasticità)
$\varepsilon_t \sim \mathrm{IID}(0, \sigma_\varepsilon^2)$
Un processo IID è stazionario in senso debole ed anche in senso forte.

Per generare un processo stocastico IID(0,2) in R:

```
e<-rnorm(10,0,2)
e
 -0.6959816 -0.2022746 -1.9541133  1.6746485  0.8520543  0.1414832
 -4.2672333 -2.0601834  2.5948709 -1.3254589

mean(e)# media del processo stocastico IID(0,2)
 -0.5242188
var(e)# varianza del processo stocastico IID(0,2)
 4.010012
```

Si è scelta per semplicità una variabile aleatoria gaussiana.

2) processo WHITE NOISE (rumore bianco) $\{\varepsilon_t,\ t \in T\}$
con ε_1, ε_3, ε_3 … variabili aleatorie con le seguenti caratteristiche;
a) $E(\varepsilon_t) = 0 \quad \forall t \in T$
b) $Var(\varepsilon_t) = \sigma_\varepsilon^2 < \infty$ (omoschedasticità)
c) $Cov(\varepsilon_t, \varepsilon_{t+k}) = 0 \quad \forall k \neq 0$ (in correlazione tra le variabile distanti k lags, per qualsiasi valore di k)
$\varepsilon_t \sim \mathrm{WN}(0, \sigma_\varepsilon^2)$
Il processo stocastico di tipo white noise è stazionario in senso debole, ma non in senso forte. Questo processo rispetto al precedente prevede anche l'assenza di correlazione tra le variabili aleatorie. Se queste si distribuiscono secondo una distribuzione normale si parla di white noise gaussiano.
Un esempio di processo white noise gaussiano è offerto dal vettore dei residui standardizzati `res.stand` del quale abbiamo provato con i test di specificazione la normalità, l'omoschedasticità e l'assenza di autocorrelazione.

3) processi AUTOREGRESSIVI $\{Z_t,\ t \in T\}$
sia $\varepsilon_t \sim \mathrm{WN}(0, \sigma_\varepsilon^2)$

$Z_t = \Phi_1 Z_{t-1} + \varepsilon_t \quad Z_t \sim AR(1)$ processo autoregressivo del primo ordine
$Z_t = \Phi_1 Z_{t-1} + \Phi_2 Z_{t-2} + \varepsilon_t \quad Z_t \sim AR(2)$ processo autoregressivo del secondo ordine

$Z_t = \Phi_1 Z_{t-1} + \Phi_2 Z_{t-2} + ... + \Phi_p Z_{t-p} + \varepsilon_t$ $\quad Z_t \sim AR(p)$ processo autoregressivo del generico ordine p

dove $\Phi_1, \Phi_2 ... \Phi_p$ sono i parametri del modello AR da stimare. Tali processi sono stazionari solo se i parametri soddisfanno determinate condizioni. Di solito si suppone che la variabile di white noise sia di tipo gaussiano per stimare i parametri in base al metodo della massima verosimiglianza.

4) processi MOVING AVERAGE $\{Z_t,\ t \in T\}$
sia $a_t \sim WN(0, \sigma_a^2)$

$Z_t = a_t - \theta_1 a_{t-1}$ $\quad Z_t \sim MA(1)$ processo a media mobile del primo ordine

$Z_t = a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2}$ $\quad Z_t \sim MA(2)$ processo a media mobile del secondo ordine

$Z_t = a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - ... - \theta_q a_{t-q}$ $\quad Z_t \sim MA(q)$ processo a media mobile del generico ordine q

dove $\theta_1, \theta_2 ... \theta_q$ sono i parametri del modello MA da stimare. Tali processi sono sempre stazionari. Di solito si suppone che la variabile di white noise sia di tipo gaussiano per stimare i parametri in base al metodo della massima verosimiglianza.

5) processi ARMA

un processo ARMA è dato dalla combinazione di un processo AR(p) e un processo MA(q):
$Z_t = \Phi_1 Z_{t-1} + \Phi_2 Z_{t-2} + ... + \Phi_p Z_{t-p} + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - ... - \theta_q a_{t-q}$ $\quad Z_t \sim ARMA(p,q)$ processo autoregressivo media mobile di ordine (p,q)
dove $\Phi_1, \Phi_2 ... \Phi_p$ e $\theta_1, \theta_2 ... \theta_q$ sono i parametri del processo ARMA da stimare. Il processi ARMA sono stazionari solo se si verificano alcune condizioni.

6) altri processi:

VAR[5]: i processi VAR costituiscono la generalizzazione multivariata del processi AR;
ARCH - GARCH: i processi ARCH (Autoregressive Conditionally Heteroskedastic) e GARCH (Generalized Autoregressive Conditionally Heteroskedastic) sono molto usati nell'analisi di serie storiche finanziarie.
I modelli ARCH, proposti dall'econometrico Engle (1982), possiedono molte delle caratteristiche nei loro parametri teorici che possono descrivere relativamente bene il comportamento delle quantità empiriche calcolate sulle serie finanziarie. Essi hanno, ad esempio, una componente erratica che non è autocorrelata, invece la sua varianza non è costante nel tempo ed è autocorrelata, infine la sua distribuzione si presenta con le caratteristiche delle distribuzioni leptocurtiche.

6.3 I modelli ARIMA

I modelli ARIMA (autoregressivi integrati a media mobile) di Box e Jenkins partono dal presupposto che fra due osservazioni di una serie quello che altera il livello della serie e' il cosiddetto disturbo. Un modello generale di Box-Jenkins viene indicato come: ARIMA (p,d,q) dove AR=AutoRegression (autoregressione) e p è l'ordine della stessa, I=Integration (integrazione) e d è l'ordine della stessa, MA=Moving Average (media mobile) e q è l'ordine delle stessa.
Pertanto un modello ARIMA (p,d,q) è analogo ad un modello ARMA(p,q) applicato alle differenze d'ordine "d" della serie dei valori, invece che agli effettivi valori. Se la serie non è stazionaria (la media e la varianza non sono costanti nel tempo) viene integrata a livello 1 o 2, dopo aver eseguito un' eventuale trasformazione dei dati (solitamente quella logaritmica). In tal modo viene ottenuta una serie stazionaria (random walk).
La procedura proposta Box e Jenkins è di tipo iterativo e serve per: l'identificazione, la stima e la verifica di un modello ARIMA ed ha come scopo la costruire un modello che si adatti alla serie storica osservata e che rappresenti il processo generatore della serie stessa.

[5] Per approfondimenti sui processi VAR e GARCH si veda il seguente link:
http://www.econ.unian.it/lucchetti/didattica/matvario/procstoc.pdf

1) Verifica della stazionarietà della serie: analisi grafica della serie; identificazione di eventuali valori anomali; ricerca delle trasformazioni più adeguate a rendere stazionaria la serie (calcolo delle differenze e uso della trasformazione Box-Cox);

2) Identificazione del modello ARIMA: individuazione degli ordine p,d,q del modello mediante l'analisi delle funzioni di autocorrelazione parziale e totale;

3) Stima dei parametri: stima dei parametri del modello ARIMA con il metodo della massima verosimiglianza o dei minimi quadrati;

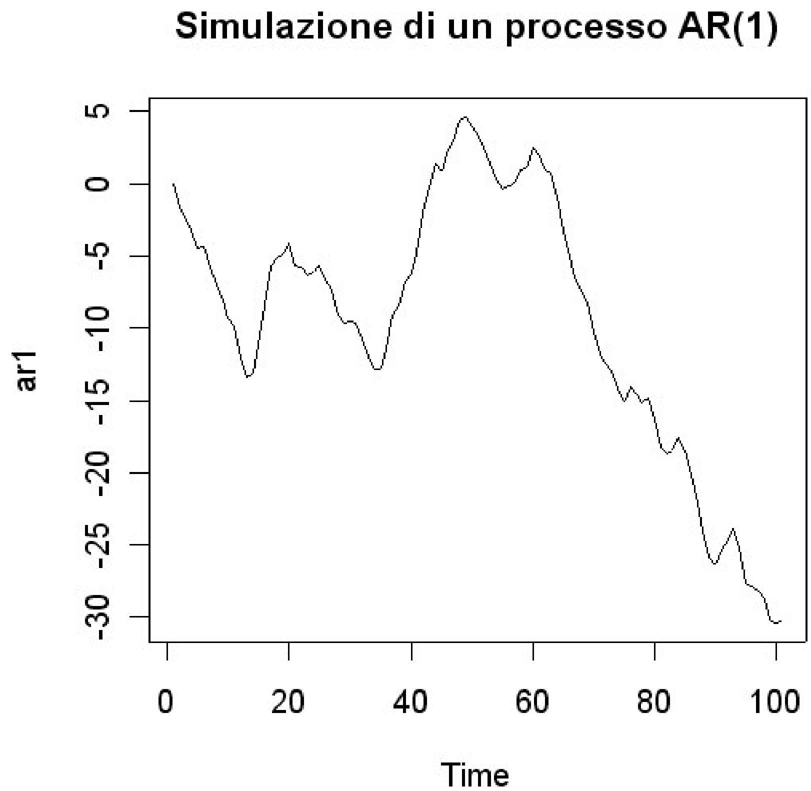
4) Verifica del modello: controllo sui residui del modello stimato per verificare se sono una realizzazione camponaria di un processo white noise a componenti gaussiane. Si effettuano analisi dei residui, test sui coefficienti, cancellazione fra operatori, test Portmanteau.

Se il modello stimato supera la fase di verifica allora può essere usato per la scomposizione e/o le previsioni. Altrimenti si ripetono le fasi di identificazione, stima e verifica (procedura iterativa).

Per quanto riguarda la stima dei parametri dei processi stocastici R mette a disposizione alcune funzioni che ora esaminiamo. Il comando *arima.sim()* permette di ottenere la simulazione di modelli AR, MA, ARMA, ARIMA specificando il numero dei valori che si vogliono ottenere, i parametri e/o l'ordine del modello in una lista.

Simulazione di in modello AR(1) di 100 osservazioni con parametro=0,6 (Graf. 34)

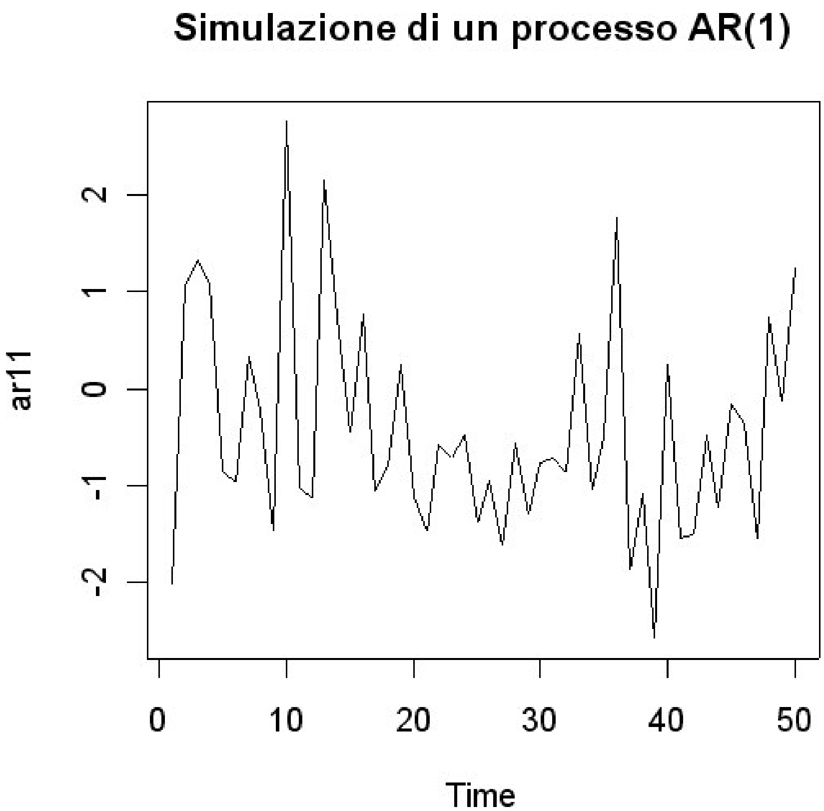
```
ar1<-arima.sim(n=100,list(order=c(1,1,0),ar=0.6))

ar1
Time Series:
Start = 1
End = 101
Frequency = 1
  [1]   0.00000000  -1.57021354  -2.48060061  -3.30206585  -4.43829285
  [6]  -4.36174163  -5.83319722  -6.95056441  -8.05188211  -9.28312665
 …
 [96] -27.84733354 -28.15538392 -28.73398390 -30.18885996 -30.33806228
[101] -30.26118540

plot(ar1, main="Simulazione di un processo AR(1)")
```



Graf. 34

Simulazione di in modello AR(1) di 50 osservazioni con parametro=0,15 e varianza 2,5 (Graf. 35)

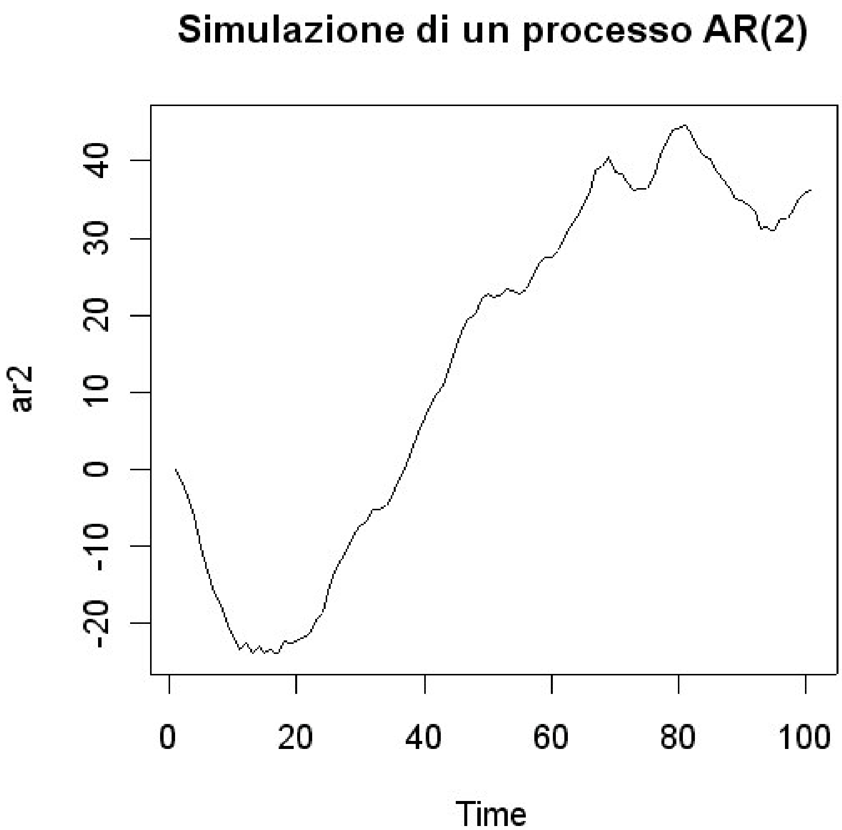
```
ar11<-arima.sim(n=50, list(ar=0.15, sd=sqrt(2.5)))
ar11
Time Series:
Start = 1
End = 50
Frequency = 1
 [1] -2.0292136  1.0657573  1.3317605  1.0925900 -0.8498379 -0.9685883
 [7]  0.3314403 -0.2210967 -1.4728292  2.7522656 -1.0131998 -1.1333778
[13]  2.1445855  0.7348620 -0.4573011  0.7669167 -1.0633599 -0.7914480
[19]  0.2521502 -1.1245348 -1.4718729 -0.5707132 -0.7143969 -0.4785436
[25] -1.3821194 -0.9574408 -1.6266108 -0.5581958 -1.3106483 -0.7678497
[31] -0.7143033 -0.8647428  0.5611187 -1.0419489 -0.4938271  1.7580476
[37] -1.8806451 -1.0812051 -2.5768661  0.2489859 -1.5406977 -1.5002566
[43] -0.4814587 -1.2349026 -0.1630366 -0.3576036 -1.5587616  0.7337884
[49] -0.1406355  1.2466153

plot(ar11, main="Simulazione di un processo AR(1)")
```



Graf. 35

Simulazione di in modello AR(2) di 100 osservazioni con parametri=0,5 e 0,4 (Graf. 36)

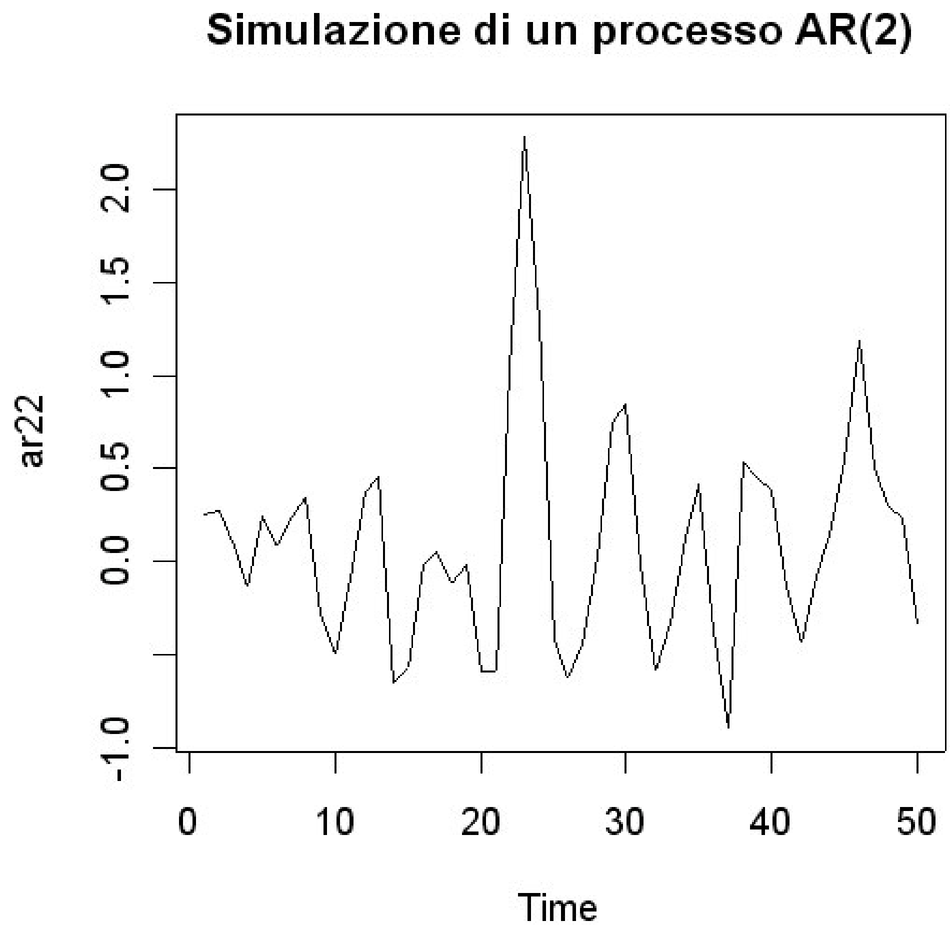
```
ar2<-arima.sim(n=100, list(order=c(2,1,0), ar=c(0.5,0.4)))
ar2
Time Series:
Start = 1
End = 101
Frequency = 1
  [1]  0.000000e+00 -1.680155e+00 -4.027103e+00 -6.107133e+00 -1.021313e+01
  [6] -1.331398e+01 -1.605751e+01 -1.761765e+01 -2.018718e+01 -2.150315e+01
 [11] -2.350094e+01 -2.242997e+01 -2.390150e+01 -2.302758e+01 -2.387115e+01
…
[96]  3.246823e+01  3.248655e+01  3.384704e+01  3.502416e+01  3.590958e+01
[101]  3.620473e+01

plot(ar2, main="Simulazione di un processo AR(2)")
```



Graf. 36

Simulazione di in modello AR(2) di 50 osservazioni con parametri=0,88 e –0,49 e varianza 0,15 (Graf. 37)

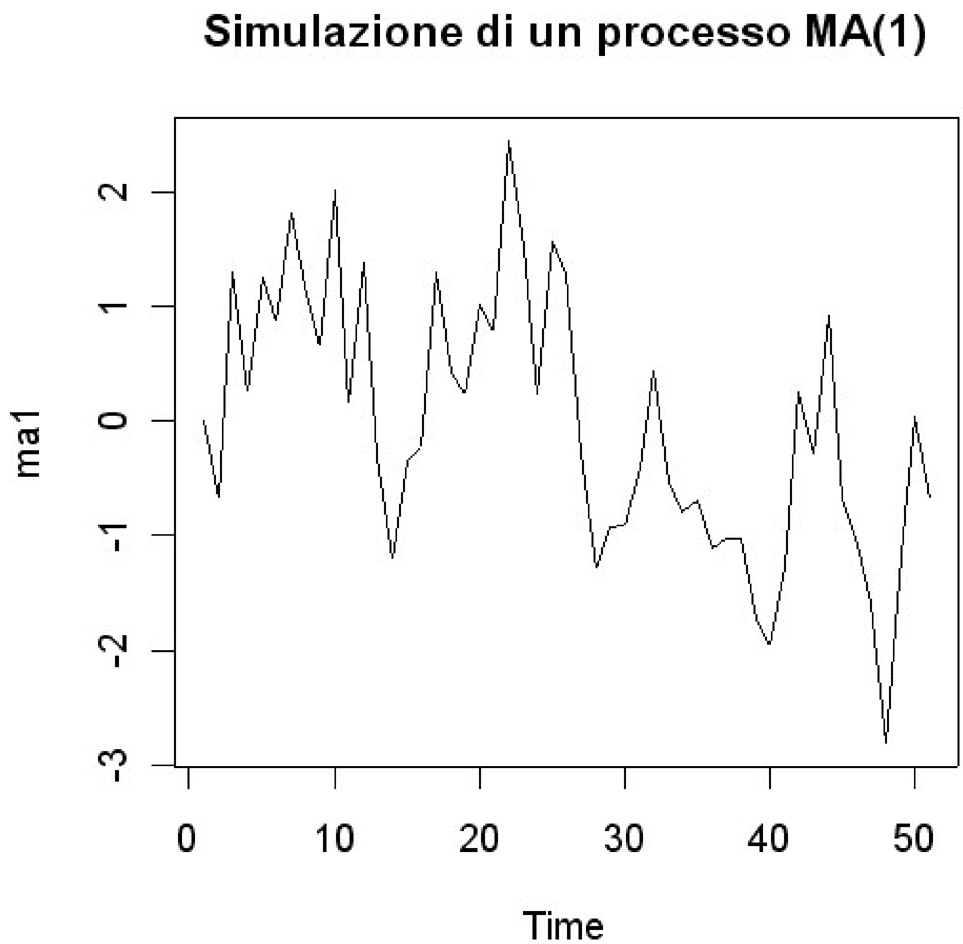
```
ar22<-arima.sim(n = 50, list(ar = c(0.88, -0.49),),  sd = sqrt(0.15))
ar22
Time Series:
Start = 1
End = 50
Frequency = 1
 [1]  0.25332920  0.27134771  0.08040647 -0.14003858  0.24037125  0.08530391
 [7]  0.24503360  0.34506735 -0.28026645 -0.50106147 -0.09352090  0.37181065
[13]  0.45138404 -0.65053733 -0.56744931 -0.02285993  0.04756471 -0.11917105
[19] -0.01767922 -0.59118537 -0.59642897  1.03128611  2.27729429  1.28472638
[25] -0.42294578 -0.63185535 -0.44090745  0.02040023  0.74244322  0.84862357
[31] -0.01866637 -0.59131670 -0.33250861  0.12477229  0.41136537 -0.37311628
[37] -0.89842710  0.53196336  0.45278489  0.38281497 -0.14251964 -0.44298049
[43] -0.07889502  0.13998188  0.53933276  1.18271913  0.50860018  0.30367282
[49]  0.23353659 -0.34218318


plot(ar22, main="Simulazione di un processo AR(2)")
```



Graf. 37

Simulazione di in modello MA(1) di 50 osservazioni con parametro=-0,7 (Graf. 38

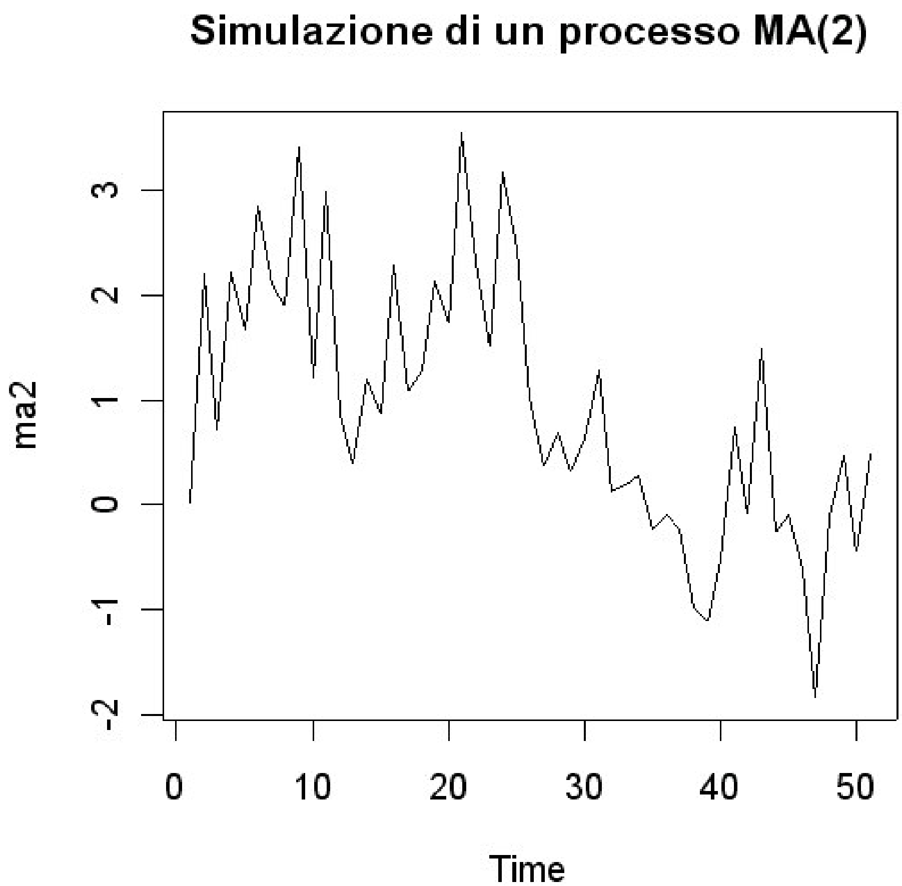
```
ma1<-arima.sim(n=50, list(order=c(0,1,1),ma=-0.7))
plot(ma1,main="Simulazione di un processo MA(1)")
```



Graf. 38

Simulazione di in modello MA(2) di 50 osservazioni con parametri=-0,9 e 0,3 (Graf. 39)

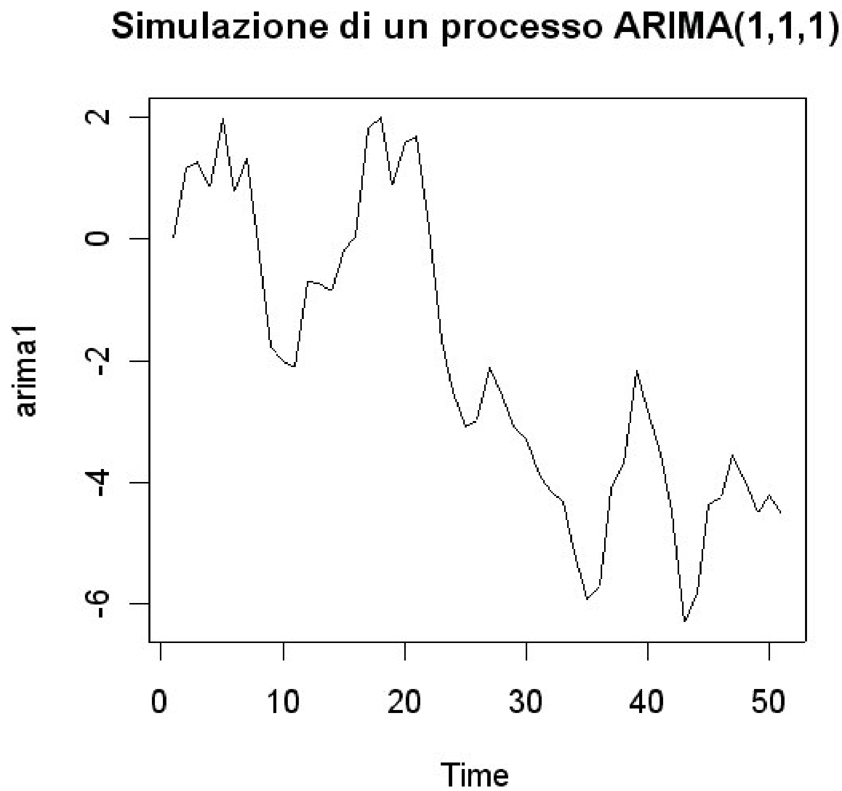
```
ma2<-arima.sim(n=50, list(order=c(0,1,2),ma=c(-0.9,0.3)))
plot(ma2,main="Simulazione di un processo MA(2)")
```



Graf. 39

simulazione di un processo ARIMA(1,1,1) con parametri AR=0,2 e MA=0,3 (Graf. 40)

```
arima1<-arima.sim(n=50, list(order=c(1,1,1), ar=0.2, ma=-0.3))
plot(arima1,main="Simulazione di un processo ARIMA(1,1,1)")
```



Graf. 40

Per stimare i parametri di un modello stocastico (AR, MA, ARMA, ARIMA) si possono usare le funzioni `ar()` e `arima()` del package `stat` e la funzione `arma()` del package `tseries`.
Nella funzione `arima()` occorre specificare l'ordine del modello (nel vettore passato come argomento il primo valore indica la componente AR, il secondo l'ordine dell'integrazione e il terzo la componente MA) e se si vuole includere un termine costante o meno (`include.mean=FALSE`). Si può anche scegliere il metodo per la stima dei parametri (`method`) e se vi è una componente stagionale (`seasonal`).

Stima dei parametri di un modello AR(2) con il comando `arima()`:

```
arfit<-arima(ar22,order=c(2,1,0),include.mean=FALSE)
arfit

Call:
arima(x = ar22, order = c(2, 1, 0), include.mean = FALSE)

Coefficients:
         ar1      ar2
      0.2822  -0.5433
s.e.  0.1190   0.1156

sigma^2 estimated as 0.2478:  log likelihood = -35.71,  aic = 77.43
```

Nel comando `ar()` bisogna specificare l'ordine del modello AR, il metodo per la stima dei parametri (di default è usato quello di Yule-Walker), se calcolare o meno il termine costante (`demean=FALSE`).

Stima dei parametri di un modello AR(2) con il comando `ar()`:

```
arfit2<-ar(ar22, order.max=2,demean=FALSE)
arfit2

Call:
ar(x = ar22, order.max = 2, demean = FALSE)

Coefficients:
      1        2
 0.7398  -0.5985

Order selected 2  sigma^2 estimated as  0.1844
```

Stima dei parametri di un modello MA(1):

```
fitma<-arima(ma1,c(0,1,1))

Call:
arima(x = ma1, order = c(0, 1, 1))

Coefficients:
          ma1
      -0.4351
s.e.   0.1716

sigma^2 estimated as 0.8721:  log likelihood = -67.63,  aic = 139.26
```

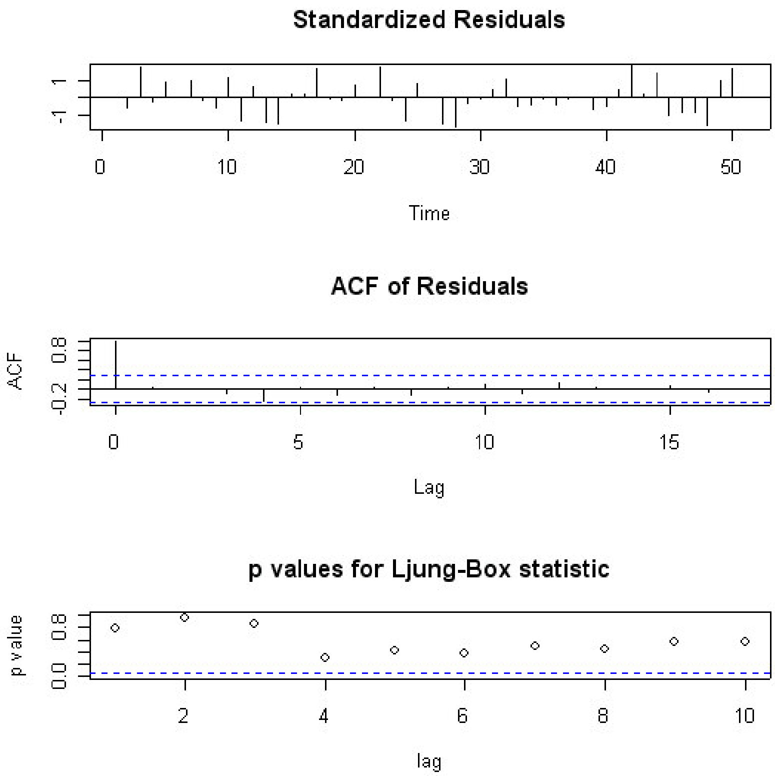
Stima dei parametri di un modello MA(2):

```
fitma2<-arima(ma2, c(0,1,2))
fitma2

Call:
arima(x = ma2, order = c(0, 1, 2))

Coefficients:
          ma1     ma2
      -0.6165  0.1567
s.e.   0.1386  0.1640

sigma^2 estimated as 0.8421:  log likelihood = -66.84,  aic = 139.69
```

Il commando `tsdiad()` consente di effettuare la diagnostica di un modello stimato. Esso fornisce in forma grafica: il grafico dei residui standardizzati, la funzione di autocorrelazione dei residui e il p-value per il test di Ljung-Box per tutti i lag.

La diagnostica del modello MA(1) stimato sembra mostra un buon adattamento:

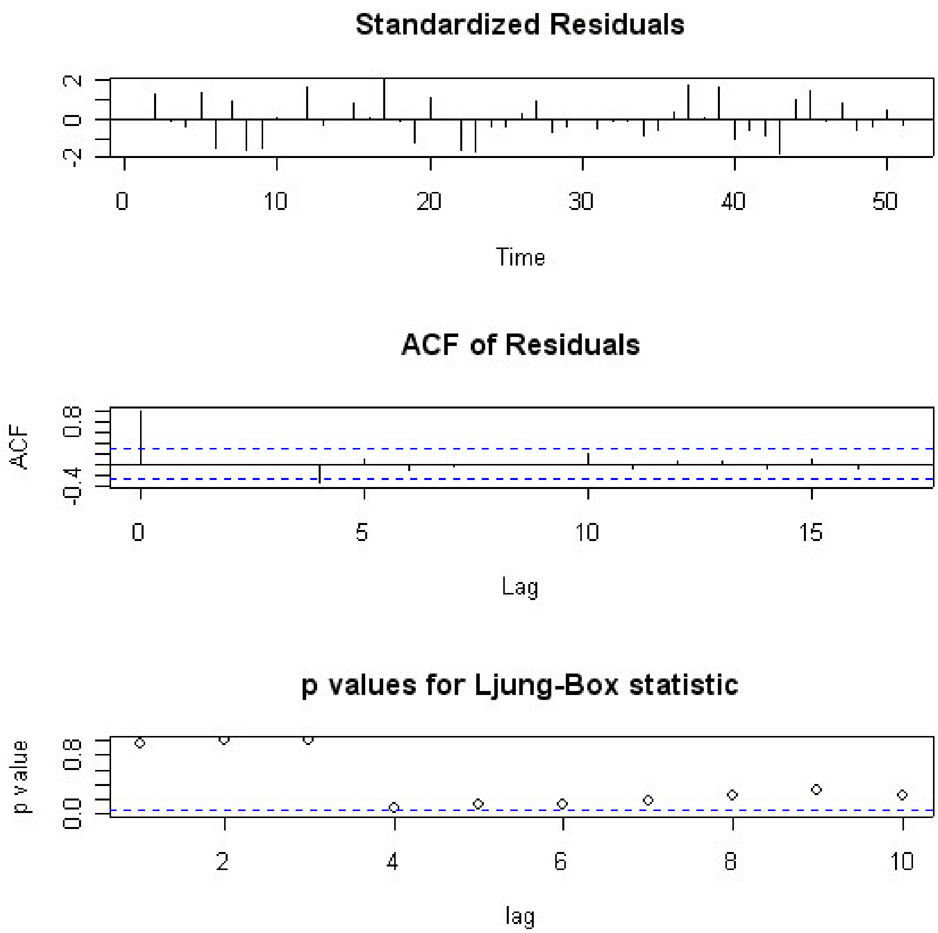
```
tsdiag(fitma)
```



Graf. 41

Stima dei parametri di un modello ARIMA(1,1,1)

```
arimafit<-arima(arima1, order=c(1,1,1), include.mean=FALSE)

arimafit

Call:
arima(x = arima1, order = c(1, 1, 1), include.mean = FALSE)

Coefficients:
         ar1     ma1
      0.1343  0.0709
s.e.  0.7059  0.7119

sigma^2 estimated as 0.7781:  log likelihood = -64.7,  aic = 135.39

tsdiag(arimafit)
```



Graf. 42

Nel comando *arma()* del package `tseries` occorre specificare l'ordine del modello e se includere o meno l'intercetta (`include.intercept=FALSE`).

Stima dei parametri di un modello ARMA(1,1) mediante il commando `arma()`

```
library(tseries)

arimafit2<-arma(arima1, order=c(1,1), include.intercept=FALSE)
arimafit2

Call:
arma(x = arima1, order = c(1, 1), include.intercept = FALSE)

Coefficient(s):
   ar1     ma1
0.9635  0.2225

summary(arimafit2)

Call:
arma(x = arima1, order = c(1, 1), include.intercept = FALSE)
```

```
Model:
ARMA(1,1)

Residuals:
    Min      1Q  Median      3Q     Max
-1.7734 -0.6836 -0.2460  0.5137  1.7899

Coefficient(s):
     Estimate  Std. Error  t value Pr(>|t|)
ar1   0.96354     0.04895   19.686   <2e-16 ***
ma1   0.22245     0.14593    1.524    0.127
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

Fit:
sigma^2 estimated as 0.7675,  Conditional Sum-of-Squares = 38.54,  AIC = 135.23
```

Alle stime dei modelli ottenute con le funzioni `arima()` e `ar()` si può applicare il metodo `predict()` per avere delle previsioni basate sul modello stimato. È necessario specificare il numero di istanti temporali per i quali si vuole effettuare la previsione (`n.ahead`).

```
predict(arfit2,n.ahead=5)
$pred
Time Series:
Start = 51
End = 55
Frequency = 1
[1] -0.39292255 -0.08592028  0.17157831  0.17836027  0.02927696

$se
Time Series:
Start = 51
End = 55
Frequency = 1
[1] 0.4293852 0.5341261 0.5345763 0.5673761 0.5673761

predict(arfit, n.ahead=7)
$pred
Time Series:
Start = 51
End = 57
Frequency = 1
[1] -0.46657296 -0.18890569 -0.04295774 -0.15261519 -0.26285552 -0.23439574
[7] -0.16647208

$se
Time Series:
Start = 51
End = 57
Frequency = 1
[1] 0.4977915 0.8094508 0.9062404 0.9444831 1.0077669 1.1046801 1.1870733
```

Nel package `fSeries` si può trovare il comando `ArmaModelling` che permette la simulazione e la stima dei parametri di modelli AR, MA e ARMA..

Appendice

Elenco dei comandi di R per l'analisi delle serie storiche
Si riportano di seguito le principali funzioni di R utili per l'analisi delle serie storiche contenute nei packages: *stats*, *tseries*, *ast* e *lmtes*t. Altri comandi, che esulano dalla trattazione del presente lavoro, possono essere reperiti nei package *dse1* e *dse2* (Multivariate Time Series Library), *its* (Irregular Time Series), *fBasics*, *fOptions* , *fExtremes* e *fSeries* (Financial Software Collection).

INPUT

cycle(): gives the positions in the cycle of each observation (stats)
deltat(): returns the time interval between observations (stats)
end(): extracts and encodes the times the last observation were taken (stats)
frequency(): returns the number of samples per unit time (stats)
read.ts(): reads a time series file (tseries)
start(): extracts and encodes the times the first observation were taken (stats)
time(): creates the vector of times at which a time series was sampled (stats)
ts(): creates time-series objects (stats)
window(): is a generic function which extracts the subset of the object 'x' observed between the times 'start' and 'end'. If a frequency is specified, the series is then re-sampled at the new frequency (stats)

TS DECOMPOSITION

decompose(): decomposes a time series into seasonal, trend and irregular components using moving averages. Deals with additive or multiplicative seasonal component (stats)
filter(): linear filtering on a time series (stats)
*HoltWinters(): c*omputes Holt-Winters Filtering of a given time series (stats)
sfilter(): removes seasonal fluctuation using a simple moving average (ast)
spectrum(): estimates the spectral density of a time series (stats)
stl(): decomposes a time series into seasonal, trend and irregular components using 'loess' (stats)
tsr(): decomposes a time series into trend, seasonal and irregular. Deals with additive and multiplicative components (ast)

TESTS

adf.test(): computes the Augmented Dickey-Fuller test for the null that 'x' has a unit root (tseries)
Box.test(): computes the Box-Pierce or Ljung-Box test statistic for examining the null hypothesis of independence in a given time series (stats)
bds.test(): computes and prints the BDS test statistic for the null that 'x' is a series of i.i.d. random variables (tseries)
bptest(): performs the Breusch-Pagan test for heteroskedasticity of residuals (lmtest)
dwtest(): performs the Durbin-Watson test for autocorrelation of residuals (lmtest)
jarque.bera.test(): Jarque-Bera test for normality (tseries)
kpss.test(): computes KPSS test for stationarity (tseries)
shapiro.test(): Shapiro-Wilk Normality Test (stats)

STOCHASTIC MODELS

ar(): fits an autoregressive time series model to the data, by default selecting the complexity by AIC (stats)
arima(): fits an ARIMA model to a univariate time series (stats)
arima.sim(): simulate from an ARIMA model (stats)
arma(): fits an ARMA model to a univariate time series by conditional least squares (tseries)
garch(): fits a Generalized Autoregressive Conditional Heteroscedastic GARCH(p, q) time series model to the data by computing the maximum-likelihood estimates of the conditionally normal model (tseries)

GRAPHICS

lag.plot: plots time series against lagged versions of themselves. Helps visualizing "auto-dependence" even when auto-correlations vanish (stats)
monthplot(): plots a seasonal (or other) subseries of a time series (stats)

plot.ts(): plotting time-series objects (stats)
seaplot(): plotting seasonal sub-series or profile (ast)
seqplot.ts(): plots a two time series on the same plot frame (tseries)
tsdiag(): a generic function to plot time-series diagnostics (stats)
ts.plot(): plots several time series on a common plot. Unlike 'plot.ts' the series can have a different time bases, but they should have the same frequency (stats)

MISCELLANEOUS

acf(), *pacf()*, *ccf()*: the function 'acf' computes (and by default plots) estimates of the autocovariance or autocorrelation function. Function 'pacf' is the function used for the partial autocorrelations. Function 'ccf' computes the cross-correlation or cross-covariance of two univariate series (stats)
diff.ts(): returns suitably lagged and iterated differences (stats)
lag(): computes a lagged version of a time series, shifting the time base back by a given number of observations (stats)

Riferimenti

Maria Maddalena Barbieri, “Appunti di statistica economica”, Università Roma 3
http://host.uniroma3.it/docenti/barbieri/statistica-economica.htm [consultato in data 24/11/04]

Alessandro Fassò, “Analisi Temporale di Dati Ambientali”, Scuola Estiva post-laurea GRASPA “Metodologie Statistiche per l'Ambiente”, Caprarola, 14-18 luglio 2003

Riccardo Lucchetti, “Appunti di analisi delle serie storiche”, Università Politecnica della Marche
http://www.econ.unian.it/lucchetti/didattica/matvario/procstoc.pdf [consultato in data 24/11/04]

Guido Masarotto, Analisi delle serie storiche. Materiale didattico, Università di Padova
http://www.statistica.unipd.it/servizi/matdid.asp?idins=7#appunti [consultato in data 24/11/04]

Matteo M. Pelagatti, Dispense del Corso di Serie Storiche Economiche, Università di Milano Bicocca
http://www.statistica.unimib.it/utenti/p_matteo/Didattica/SSE/sse.html [consultato in data 24/11/04]

Tommaso Proietti, Dispense del corso di Statistica per l’Analisi Economica, Università di Udine
http://www.dss.uniud.it/utenti/proietti/StatisticaAnalisiEconomica0304.html [consultato in data 24/11/04]

R Core Team, “An Introduction to R”, ver. 2.0.1, novembre 2004
http://cran.r-project.org/doc/manuals/R-intro.pdf [consultato in data 24/11/04]

Vito Ricci, “R : un ambiente open source per l'analisi statistica dei dati”, Economia e Commercio, (1):69-82, 2004, http://www.dsa.unipr.it/soliani/allegato.pdf [consultato in data 24/11/04]

Gabriele Soffritti, “Introduzione ai metodi per l’analisi delle serie storiche”, Università di Bologna
http://amscampus.cib.unibo.it/archive/00000794/01/analisi_delle_serie_storiche.pdf [consultato in data 24/11/04]

Walter Zucchini, Oleg Nenadic, “Time Series Analysis with R “,Università di Gottinga
http://www.statoek.wiso.uni-goettingen.de/veranstaltungen/zeitreihen/sommer03/ts_r_intro.pdf [consultato in data 24/11/04]

Walter Zucchini, Oleg Nenadic, “Statistical Analysis with R”, Università di Gottinga
http://www.statoek.wiso.uni-goettingen.de/mitarbeiter/ogi/pub/r_workshop.pdf [consultato in data 24/11/04]