UNIVERSITÀ DI TRENTO

TRENTINO SCHOOL OF MANAGEMENT

— Parte 3. —

Serie temporali e ARIMA

Paolo Bosetti (`paolo.bosetti@unitn.it`)

Data creazione: 2022-01-28 11:02:21

Indice

1 Time series

1.1 La classe `ts`

Le serie temporali vengono create con la funzione nativa `ts(data, start, end, frequency)`, dove:

- `data` è un vettore di dati equi-spaziati nel tempo
- `start` è la data della prima osservazione
- `end` è la data dell'ultima osservazione
- `frequency` è il numero di osservazioni per unità temporale

Il significato dell'unità tempo base è arbitrario: se ad esempio indichiamo `start=2019` e `frequency=12` significa che i dati partono dal 2019 e hanno cadenza mensile. È possibile indicare `start=c(2019,6)` per stabilire che il primo dato è di Giugno 2019. **NOTA**: `start` deve essere o uno scalare o un vettore di due elementi, nel cui caso il secondo elemento è l'indice (base 1) del sotto-periodo quando `frequency` è maggiore di 1.

Le opzioni `end` o `deltat` possono essere indicate quando si vuole troncare il vettore di ingresso.

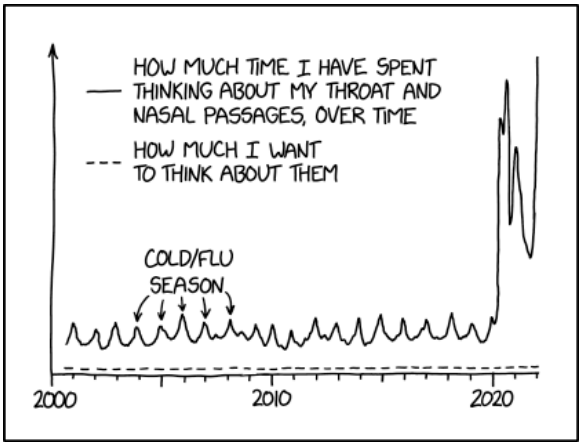


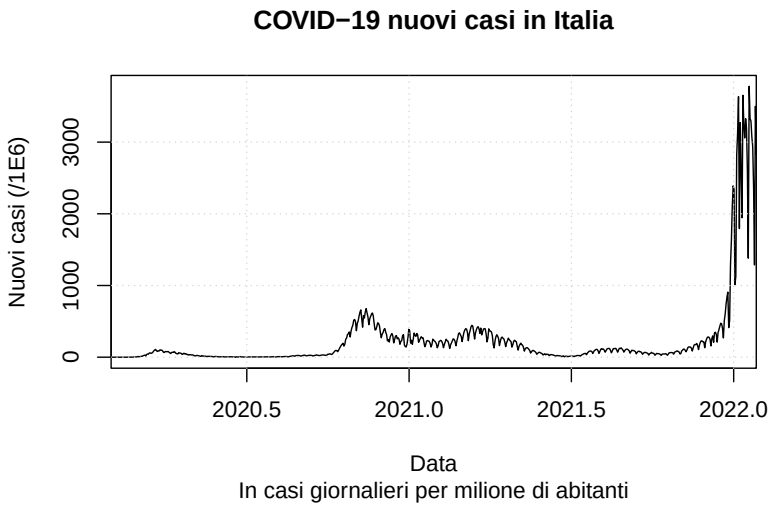
Figura 1: https://xkcd.com/2563/

Come dati di esempio, carichiamo i dati della pandemia COVID-19 da Our World in Data:

```
url <- "https://covid.ourworldindata.org/data/owid-covid-data.csv"
datafile <- basename(url)
if (!file.exists(datafile) | difftime(now(),  file.mtime(datafile), units="hours") > 24 ) {
  print("Downloading new data from the Internet")
  download.file(url, datafile)
}
covid <- read.csv(datafile)
```

Dell'intero set di dati filtriamo e selezioniamo solo i nuovi casi per milione in Italia, costruendo poi un oggetto time series. Usiamo la libreria `lubridate` per semplificare la gestione delle date:

```
st <- decimal_date(ymd(covid[covid$location=="Italy",]$date[1]))
cpm <- ts(
  covid[covid$location=="Italy",]$new_cases_per_million,
  start=st,
  frequency=365.25
)
plot(cpm,
     main="COVID-19 nuovi casi in Italia",
     sub="In casi giornalieri per milione di abitanti",
     xlab="Data",
     ylab="Nuovi casi (/1E6)",
     xaxs="i"
     )
grid()
```



Si noti che l'espressione `decimal_date(ymd(covid$date[1]))` converte la data 2020-02-24 (una stringa) in un oggetto tempo 2020-02-24 e infine in un valore decimale a base annuale: 2020.147541 (*data astrale*):

```
cat("Data astrale: "); print(c(start(cpm), end(cpm)))
```

```
## Data astrale:
```

```
## [1] 2020.082 2022.070
```

```
cat("Data POSIX: "); print(date_decimal(c(start(cpm), end(cpm))))
```

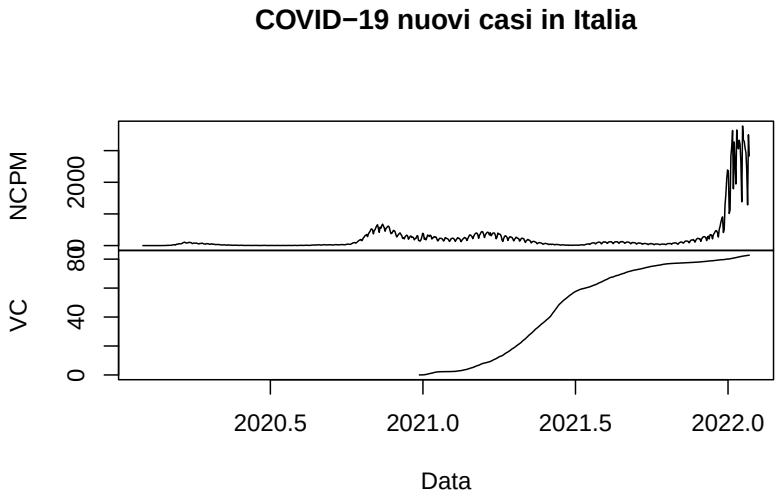
```
## Data POSIX:
```

```
## [1] "2020-01-30 23:59:59 UTC" "2022-01-26 10:06:24 UTC"
```

1.2 Multivarate time series

È possibile creare oggetti timeseries multi-variati, passando all'argomento `data` una matrice con più colonne:

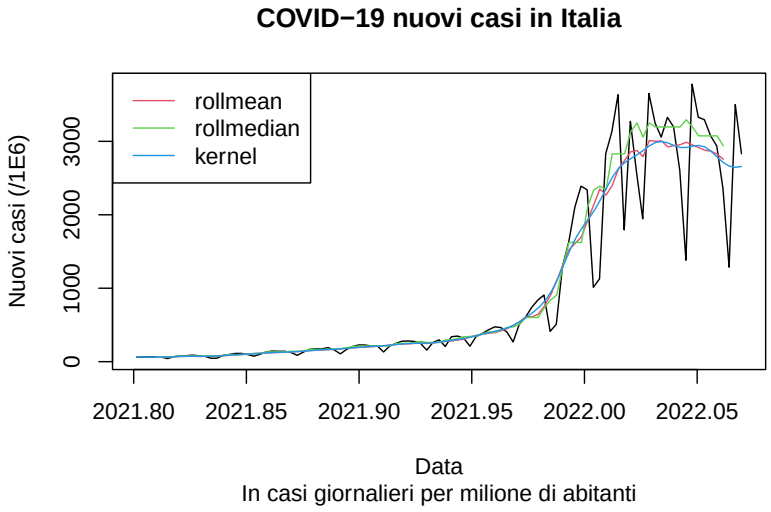
```
cpmv <- ts(
  data=cbind(
    covid[covid$location=="Italy",]$new_cases_per_million,
    covid[covid$location=="Italy",]$people_vaccinated_per_hundred
  ),
  names=c("NCPM", "VC"),
  start=st,
  frequency=365.25
)
plot(cpmv,
     main="COVID-19 nuovi casi in Italia",
     sub="In casi giornalieri per milione di abitanti",
     xlab="Data",
     ylab="Nuovi casi (/1E6)",
     )
```



1.3 Finestre e Smoothing

Funzioni utili per manipolare le serie temporali sono `window()` e `time()`: la prima consente di estrarre una finestra temporale tra due date, la seconda consente di estrarre il vettore dei tempi. Inoltre, sono utili le funzioni di smoothing fornite dalla libreria `zoo`

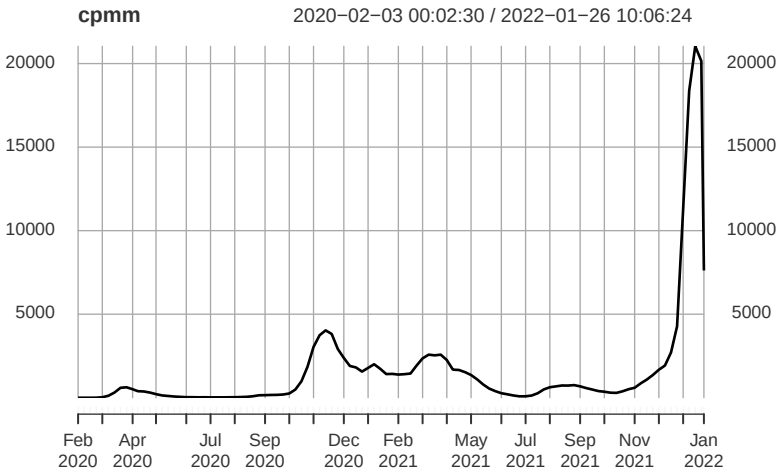
```
win <- window(cpm, start=2021.8, end=end(cpm))
plot(win,
     main="COVID-19 nuovi casi in Italia",
     sub="In casi giornalieri per milione di abitanti",
     xlab="Data",
     ylab="Nuovi casi (/1E6)",
     xaxs="r"
     )
lines(rollmean(win, 7), typ="l", col=2)
lines(rollmedian(win, 7), typ="l", col=3)
lines(ksmooth(time(win), win, "normal", bandwidth=1/(365.25 / 7)), col=4)
legend("topleft", lty=1, col=2:4, legend=c("rollmean", "rollmedian", "kernel"))
```



1.4 Consolidamento

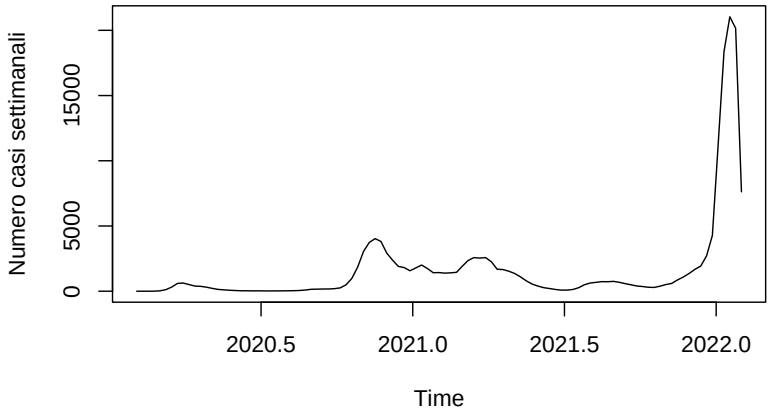
È spesso utile consolidare una serie temporale per sotto-periodi: ad esempio trasformare una serie giornaliera come `cpm` in una serie mensile o settimanale. La libreria `xts` mette a disposizione le funzioni `apply.[dayly|wekly|monthly|quarterly|yearly]()`, che però operano su un differente tipo di oggetti, appunto la classe `xts`. La libreria `tsbox` contiene appunto la funzione `ts_xts()` per convertire un `ts` in un `xts`:

```
cpmm <- apply.weekly(ts_xts(cpm), sum)
plot(cpmm)
```



Ora `cpmm` è un oggetto `xts`: la conversione di nuovo verso `ts` può essere fatta così:

```
cpmm <- ts(coredata(cpmm),
           start = decimal_date(index(cpmm)[1]),
           frequency = 365.25/7)
ts.plot(cpmm, ylab="Numero casi settimanali")
```



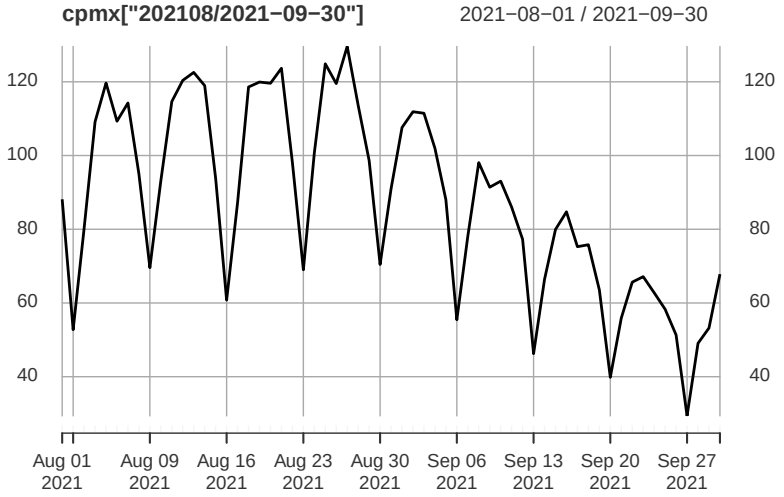
1.5 La classe `xts`

In realtà, la classe `xts` è molto più potente di `ts` nella *gestione* della serie temporale, ed è quindi in certi casi preferibile. Invece che convertire `cpm` come fatto sopra, vediamo come creare direttamente un oggetto `xts`:

```
cpmx <- xts(covid[covid$location=="Italy",]$new_cases_per_million,
            order.by = ymd(covid[covid$location=="Italy",]$date)
            )
```

L'estrazione di sottoinsiemi (subsetting) viene effettuata, anziché con il metodo `window()`, come una semplice indicizzazione (cioè il metodo `[.xts()`). È possibile usare sia indici numerici (convenzionali) sia stringhe in standard ISO-8601. La data può cioè essere espressa come intervallo:

```
plot(cpmx["202108/2021-09-30"])
```




```
# p1 <- autoplot(cpmx["202108/2021-09-30"]) +
#   geom_line() +
#   geom_area(fill="gray", alpha=1/3) +
#   geom_line(data=cpmx["2021-10-1/"], aes(x=Index, y=cpmx["2021-10-1/"]))
# p1
```

La data di inizio (prima di `/`) o di fine dell'intervallo (dopo la `/`) possono essere omesse, in tal caso significa "dall'inizio fino a ..." oppure "da ... fino alla fine". Inoltre, è possibile omettere componenti della data, intendendo così un intero sotto-periodo:

```
length(cpmx["2021"]) # Tutto l'anno
```

```
## [1] 365
```

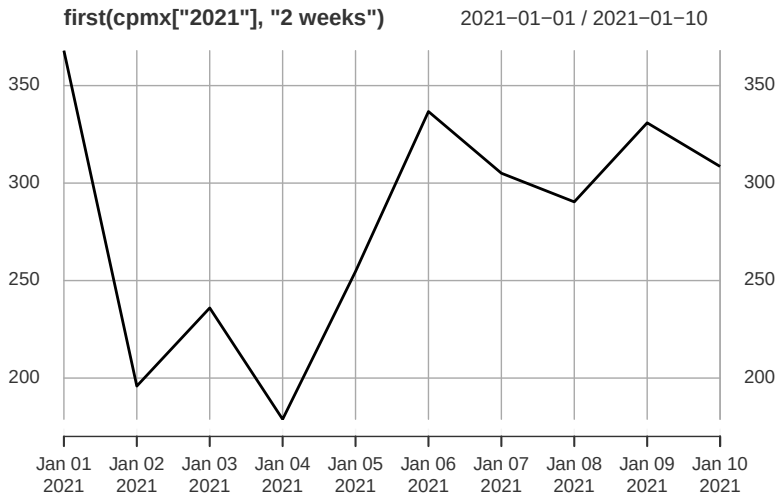
```
length(cpmx["2021-6"]) # Tutto Giugno
```

```
## [1] 30
```

```
last(cpmx, "2 week") # Ultime due settimane
```

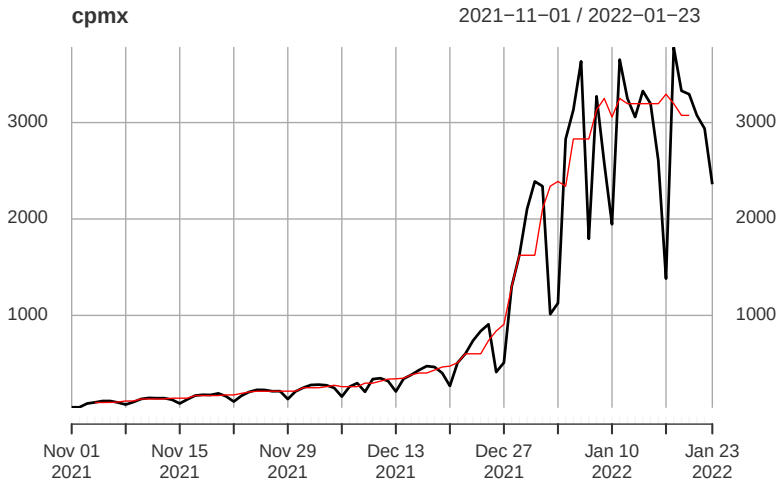
```
##                [,1]
## 2022-01-17 1381.323
## 2022-01-18 3778.906
## 2022-01-19 3329.045
## 2022-01-20 3294.241
## 2022-01-21 3074.503
## 2022-01-22 2937.592
## 2022-01-23 2360.327
## 2022-01-24 1286.554
## 2022-01-25 3499.848
## 2022-01-26 2831.657
```

```
plot(first(cpmx["2021"], "2 weeks")) # Prime due settimane del 2021"
```



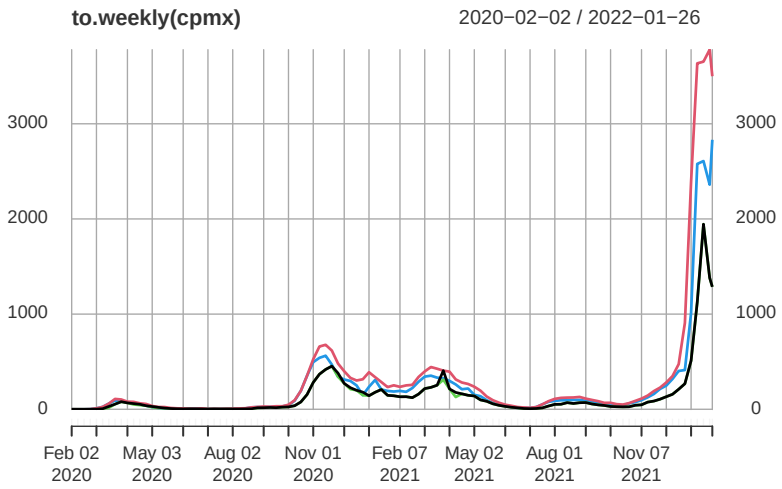
Infine, la funzione `endpoints()` consente di identificare gli indici della serie a cui terminano specifici periodi (anno, mese, settimana, giorno...). Inoltre, combinazioni di `first()` e `last()` possono essere utilizzate per selezionare i dati fino all'ultima domenica:

```
invisible( # necessario, addPanel crea un nuovo grafico
  plot(cpmx[paste0("/", index(last(first(last(cpmx, "2 weeks"), "week"))))]
       ["2021-11/"],
     main="cpmx")
)
addPanel(rollmedian, k=7, on=1, col="red")
```



Ci sono anche utili funzioni per convertire il periodo in un periodo più lungo: ad esempio, da una serie giornaliera ad una serie settimanale mediante `to_weekly()`. Questi comandi restituiscono quattro serie "OHLC": *Opening, High, Low, Closing*, cioè il primo valore del sotto-periodo, il massimo, il minimo e l'ultimo valore:

```
plot(to.weekly(cpmx))
```



La classe `xts` è quindi molto potente ma ha alcuni punti deboli:

- non va molto d'accordo con le funzioni `Arima()` e `predict()`: gli oggetti regressione che si ottengono sono convertiti nella classe base `ts` ma perdono l'informazione temporale (quindi iniziano con tempo 1 e hanno passo 1)
- il metodo `xts.plot()` è apparentemente più carino, ma molto meno flessibile dell metodo generico: ad esempio è molto complesso estendere una serie sullo stesso plot con dati successivi.

Per questi motivi, si consiglia l'uso di `xts` per la gestione della serie temporale, l'estrazione di sotto-periodi e l'eventuale aggregazione, ma poi si consiglia di convertire di nuovo in `ts` mediante il metodo `ts_ts()` prima di effettuare le regressioni.

2 Regressione e Predizione

2.1 Verifiche iniziali

La prima verifica è sempre quella sui dati mancanti. Eliminiamo qualche dato dalla serie `cpm` per vedere, in seguito, come gestire i dati mancanti:

```
cpmx[c(30, 213, 401)] <- NA
```

Decidiamo di sostituire i dati mancanti con la mediana dei dati adiacenti:

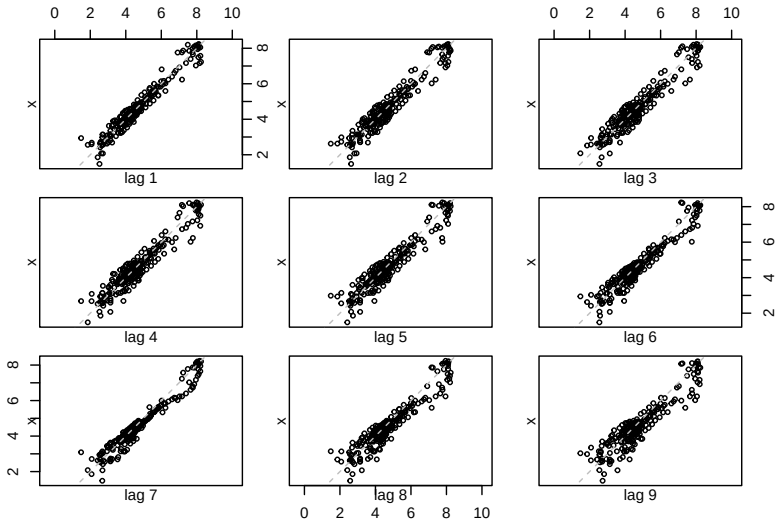
```
nas <- which(is.na(cpmx))
for (i in nas) {
  cpmx[i] = median(cpmx[i-1], cpmx[i+1])
}
```

In maniera più efficiente si possono usare le funzioni `na.locf()` o `na.approx()`: la prima sostituisce ogni `NA` con l'ultimo valore noto, la seconda lo approssima con un'interpolazione lineare (e `na.spline()` con una bicubica):

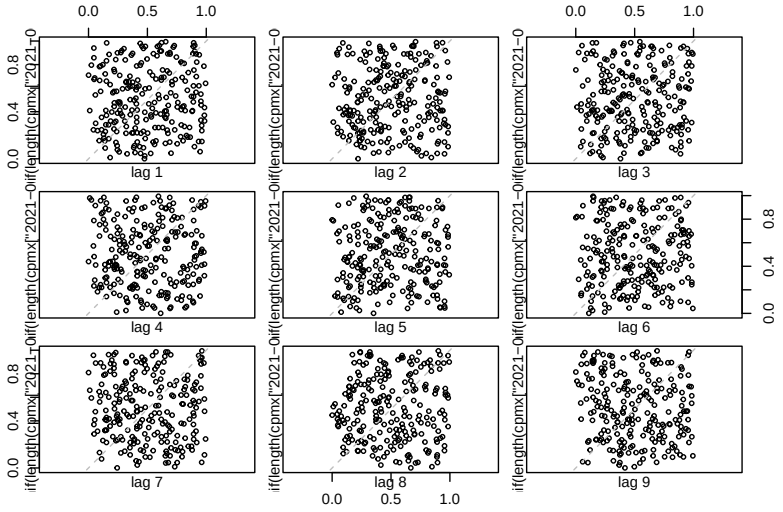
```
cpmx[c(30, 213, 401)] <- NA
cpmx <- na.locf(cpmx)
```

Prima di qualsiasi analisi su una serie temporale è utile visualizzare il cosiddetto **lag plot**, che è un particolare grafico a dispersione in cui si confrontano i dati di una serie con gli stessi dati con un certo ritardo: se il segnale è puramente casuale, il risultato sarà una nuvola dispersa; viceversa, ogni pattern significa che i dati sono affetti da un andamento regolare. Inoltre, nel nostro caso si nota che la dispersione è molto stretta al lag 7, il che dimostra la regolarità settimanale della serie temporale.

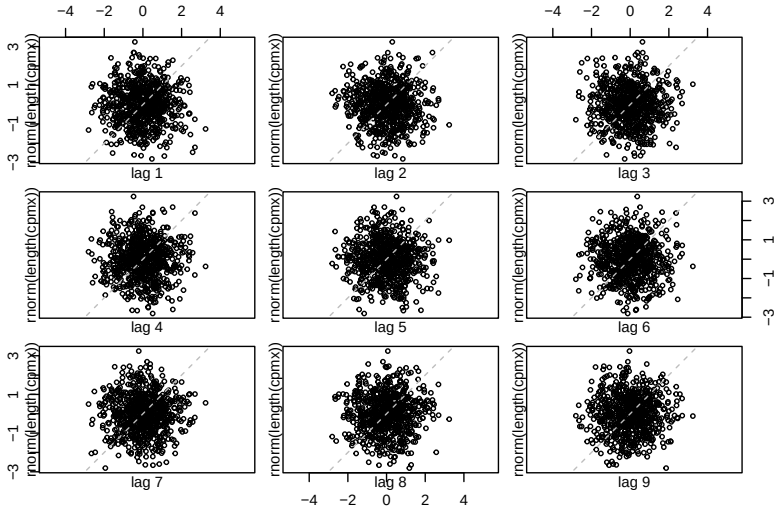
```
lag.plot(log(cpmx["2021-06/"]), lags=9)
```




```
lag.plot(runif(length(cpmx["2021-06/"])), lags=9)
```




```
lag.plot(rnorm(length(cpmx)), lags=9)
```



2.2 Auto-ARIMA

La libreria `forecast` mette a disposizione il metodo più semplice per effettuare la regressione di una serie temporale mediante ARIMA (*Auto-Regressive Integrative Moving Average*). Mettiamolo alla prova sulla serie temporale COVID-19, addestrando il modello fino alla data 2021.7 = 2021-09-13 12:00:00, utilizzando il modello per predire i successivi 30 giorni, e poi confrontandolo con i dati reali.

È importante ricordare che un modello ARIMA si applica a serie temporali che devono essere **stazionarie** (cioè a media più o meno costante) e **a varianza costante**. Se le oscillazioni della serie storica non sono costanti si può applicare una trasformazione: tipicamente si prova con il logaritmo, ma altre possibilità sono l'inversa, la radice, o una potenza. In generale, si parla di *trasformazioni Box-Cox* utilizzando il parametro λ: se esso è uguale a zero, la trasformazione è il logaritmo, altrimenti è l'elevazione alla potenza λ (ad es. $\lambda = -0.5$ corrisponde all'inverso della radice, ecc.)

Se la serie non ha una media stazionaria si può effettuare una **differenziazione** (cioè derivata), un numero di volte sufficiente. L'ordine di differenziazione è il parametro d del modello ARIMA.

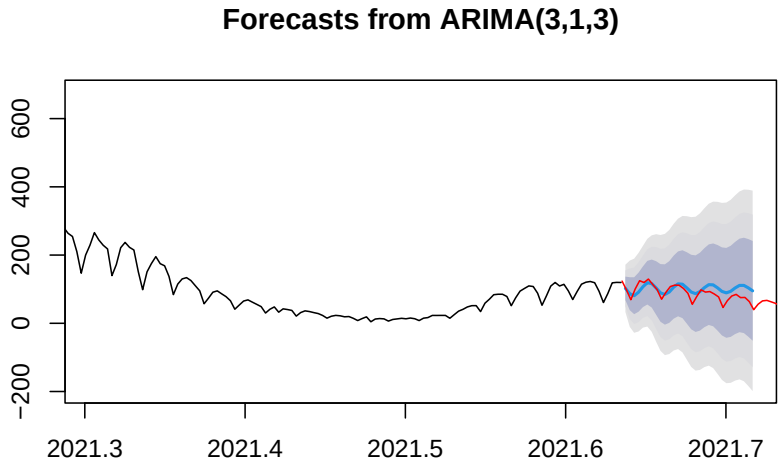
La funzione `auto.arima()` individua automaticamente i coefficienti p, d, q e, se si specifica il parametro `lambda="auto"`, anche il λ più appropriato per rendere stazionaria anche la varianza.

Si noti che le funzioni `auto.arima()` e `forecast()` perdono l'asse dei tempi quando vengono utilizzate su oggetti `xts`, quindi usiamo `xts` per selezionare i peridi (più comodo) ma convertiamo in oggetti `ts` per l'analisi:

```
d0 <- "/2021-08-20"
d1 <- "2021-08-21/"
win <- ts_ts(cpmx[d0])
(fit2 <- auto.arima(win, lambda="auto"))
```

```
## Series: win
## ARIMA(3,1,3)
## Box Cox transformation: lambda= 1
##
## Coefficients:
##          ar1     ar2      ar3      ma1      ma2     ma3
##       0.3362  0.1400  -0.8473  -0.3508  -0.4186  0.7974
## s.e.  0.0388  0.0468   0.0480   0.0456   0.0876  0.0571
##
## sigma^2 estimated as 722.2:  log likelihood=-2668.85
## AIC=5351.69   AICc=5351.89   BIC=5382.08
```

```
plot(forecast(fit2, 30, level=c(80, 95, 99)),
     xlim=c(-120,+30)/365+decimal_date(ymd(d0))
     )
new <- ts_ts(cpmx[d1])
lines(new, col="red")
```

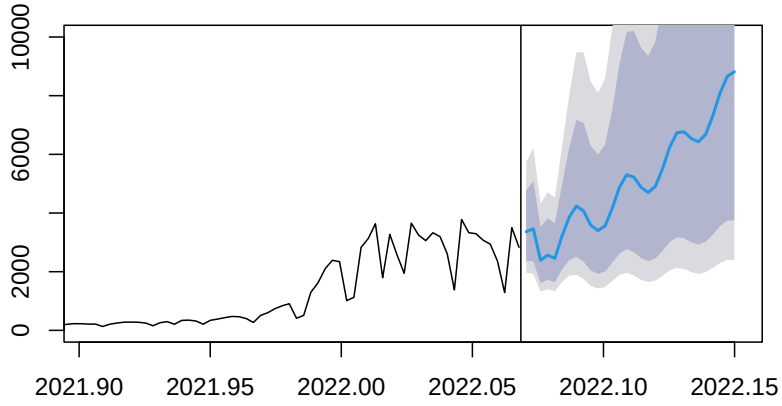


Vediamo le predizioni odierne:

```
win <- last(cpmx, "16 weeks")
(fit <- auto.arima(ts_ts(win), lambda=0))
```

```
## Series: ts_ts(win)
## ARIMA(4,1,3) with drift
## Box Cox transformation: lambda= 0
##
## Coefficients:
##           ar1     ar2      ar3      ar4      ma1      ma2     ma3   drift
##       -0.0044  0.0881  -0.5461  -0.4075  -0.5408  -0.3797  0.6725  0.0399
## s.e.   0.1303  0.1455   0.1108   0.1243   0.1287   0.1950  0.1017  0.0103
##
## sigma^2 estimated as 0.07441:  log likelihood=-10.1
## AIC=38.19   AICc=40.05   BIC=62.25
```

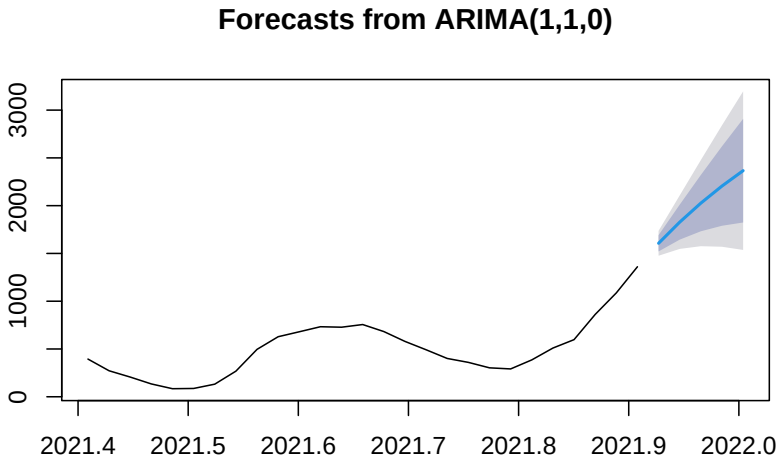
```
plot(forecast(fit, 30),
     xlim=c(-60,+30)/365+decimal_date(end(cpmx)),
     ylim=c(0, 10000)
)
abline(v=decimal_date(end(cpmx)))
```



Si noti che si è forzata una trasformazione con $\lambda = 0$, cioè il logaritmo, dato che la varianza della finestra considerata è evidentemente non costante, anche se `auto.arima(..., lambda="auto")` individuerebbe un $\lambda = 1$ (nessuna trasformazione). A volte gli automatismi non funzionano!

In realtà, le oscillazioni settimanali sono più un artefatto di misura che una proprietà intrinseca del fenomeno, quindi è più corretto effettuare predizioni su, ad esempio, i valori settimanali. Quindi utilizziamo lo stesso oggetto `cpmm` sopra ottenuto sommando i valori settimanali, e ci concentriamo sulla finestra `2021.4` – `2021.911`. Inoltre, come vedremo più avanti, il metodo ARIMA si applica a serie *stazionarie*, in cui cioè valore medio e varianza sono stabili. Il metodo più comune per stabilizzare la varianza è *trasformare* i dati applicando il logaritmo:

```
cpmm <- apply.weekly(ts_xts(cpm), sum)
win <- ts_ts(cpmm["2021-05-27/2021-11-29"])
# Fino all'ultima domenica
#win <- ts_ts(cpmm[1:(last(endpoints(cpmm, on="weeks")-1))]["2021-6/"])
fit <- auto.arima(win, lambda="auto")
plot(forecast(fit, h=5))
```



2.3 ARIMA, the hard way

2.3.1 Parametri del modello

Per calibrare un modello ARIMA è necessario identificare i parametri p, d e q.

Anzitutto, come deto sopra un modello ARMA ($d = 0$) si può applicare solo ad una serie temporale *stazionaria*, cioè priva di deriva e a varianza costante. Se la serie in questione non ha queste caratteristiche, è possibile applicare delle trasformazioni: ad esempio, possiamo applicare il logaritmo per comprimere la varianza, e differenziare una o più volte per rimuovere la deriva. Il numero di differenziazioni corrisponde al parametro d che trasforma un modello ARMA(p, q) in ARIMA(p, d, q).

Il passo successivo è individuare il grado dei processi AR e MA. Per quanto riguarda un processo MA, il suo grado q è il numero di elementi consecutivi interessati alla media mobile:

$$x_t = w_t + \theta_1 w_{t-1} + \theta_2 w_{t-2} + \cdots + \theta_q w_{t-q}$$

È evidente, quindi, che i campioni più vicini di q saranno fortemente correlati, mentre quelli più lontani risulteranno non correlati. Possiamo cioè stimare q sulla base della *funzione di autocorrelazione* (ACF), che valuta l'autocorrelazione tra due copie della stessa serie traslate di una certa distanza in passi temporali h, detta *lag*:

$$\text{ACF}(h) = \text{corr}(x_t, x_{t+h})$$

Tale funzione vale sempre 1 per un lag 0 (autocorrelazione con se stesso), e per un processo MA(q) va a zero al lag $q + 1$.

Per quanto riguarda i processi AR(p), essi rappresentano un'auto-regressione:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots + \phi_p x_{t-p} + w_t$$

Per stimare p abbiamo quindi bisogno di stimare la correlazione tra x_t e una sua versione ritardata, eliminando i contributi a lag intermedi. Si costruisce cioè la *funzione di autocorrelazione parziale* (PACF), che riporta, in funzione del lag h, l'autocorrelazione avendo eliminato (sostituendolo con una regressione) il contributo tra lag 1 e lag $n-1$:

$$\text{PACF}(h) = \text{corr}(x_{t+h} - \hat{x}_{t+h}, x_t - \hat{x}_t)$$

dove $\hat{x}_{t+h} = \beta_1 x_{t+h-q} + \beta_2 x_{t+h-2} + \ldots \beta_{h-1} x_{t+1}$ e $x_t = \beta_1 x_{t+1} + \beta_2 x_{t+2} + \cdots + \beta_{h-1} x_{t+h-q}$, e i coefficienti β_i sono calcolati minimizzando i residui.

Anche in questo caso, il grado del processo q corrisponde al lag al di là del quale la PACF va a zero (*drop-off*).

Quindi, come regola base, dopo aver reso stazionaria la serie storica mediante differenziazione, si studiano ACF e PACF per identificare q e p, rispettivamente. Valgono le seguenti linee guida:

- se il processo è AR, la PACF ha un drop-off dopo il lag p e la ACF decade geometricamente
- se il processo è MA, la ACF ha un drop-off dopo il lag q e la PACF decade geometricamente
- se il processo è ARMA, sia ACF che PACF manifestano un drop-off, e possono essere utilizzate per stimare p e q; tuttavia esse sono spesso meno chiare che nei casi precedenti
- se un processo è puro noise, né ACF né PACF mostrano alcuna struttura
- eventuali *stagionalità* si mostrano come picchi intensi a lag elevati (corrispondenti al periodo della stagionalità)

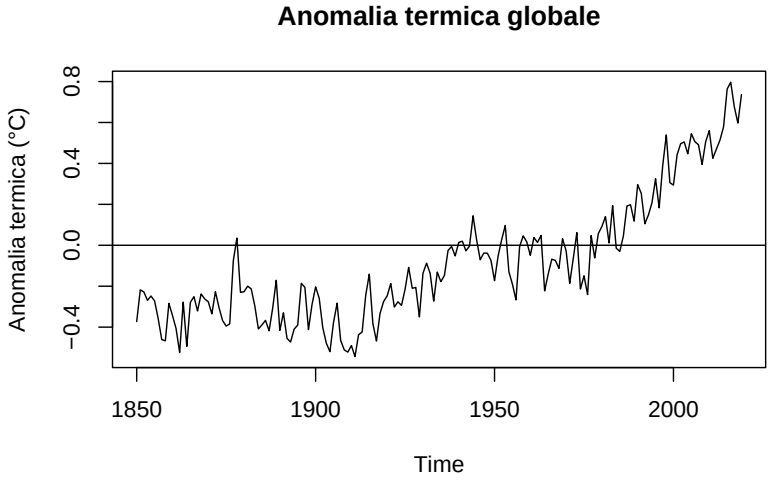
Generalmente, a meno che un processo non risulti AR o MA puro, le funzioni ACF e PACF vengono utilizzate per identificare *set* di possibili parametri p e q, scegliendo poi la combinazione migliore mediante gli stimatori di bontà della regressione. Il più adatto a questo scopo è AIC (*Akaike Information Criterion*), che deve essere minimizzato.

2.3.2 Esempio: Anomalia terra-mare

Consideriamo i dati di anomalia termica terra-mare, disponibili su Our World in Data.

Carichiamo i dati e li importiamo in una serie temporale:

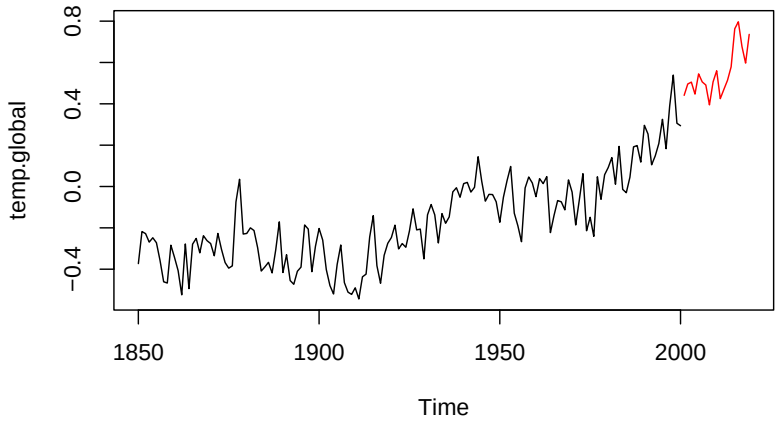
```
datafile <- "temperature-anomaly.csv"
data <- read.csv(mydata(datafile))
t.global <- ts(data[data$Entity=="Global",]$Median.temp, start=1850)
plot(t.global, ylab="Anomalia termica (°C)", main="Anomalia termica globale")
abline(h=0)
```



La serie temporale rappresenta i valori tra 1850, 1, 2019, 1.

Dividiamo il dataset in due parti: dal 1850 fino al 2000, da usare per il training del modello, e una dal 1851 fino al 2019 da usare per la validazione:

```
temp.global <- window(t.global, end=2000)
temp.global.test <- window(t.global, start=2001)
plot(temp.global,
     xlim=c(start(temp.global)[1], end(temp.global.test)[1]),
     ylim=c(min(temp.global), max(temp.global.test))
)
lines(temp.global.test, col="red")
```



Un modello ARIMA deve essere applicato ad una serie **stazionaria**: la serie cioè deve avere una varianza stabile nel tempo e non deve mostrare trend. Per stabilizzare la varianza si applicano delle *trasformazioni* alla serie: elevazioni a potenza o logaritmi. Per eliminare i trend si differenzia il segnale una o più volte: il numero di differenziazioni è l'indice di integrazione del modello ARIMA.

La trasformazione migliore è quella che minimizza il coefficiente di varianza della serie. Il metodo Box-Cox è comunemente adottato per individuare il parametro di trasformazione λ che minimizza il coefficiente di variazione:

```
(lambda <- BoxCox.lambda(temp.global))
```

```
## [1] 0.7753935
```

Il valore di λ ottenuto è vicino a 1, per cui si ritiene che non sia necessaria alcuna trasformazione.

Se λ fosse molto diversa da 1, si procederebbe così:

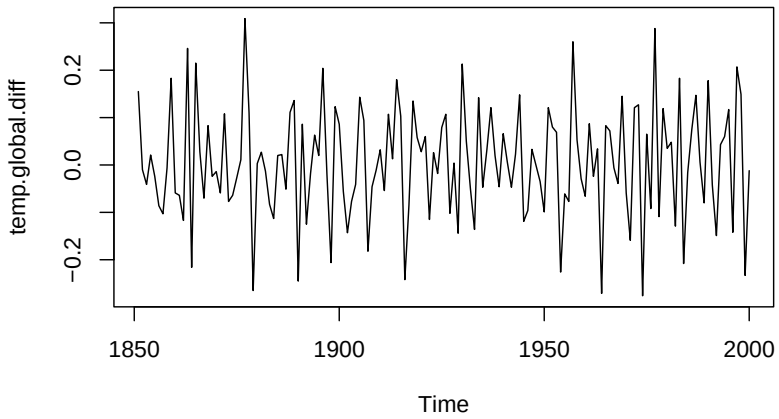
```
temp.global.BC <- BoxCox(temp.global + 273.15, lambda)
# utilizzando di seguito la trasformata temp.global.BC
```

Il prossimo passo è eliminare il trend mediante differenziazione. Il comando `ndiffs()` restituisce l'opportuno ordine di differenziazione per stabilizzare la serie, dopodiché il comando `diff(ts, differences=n)` applica la differenziazione di ordine `n`:

```
(d <- ndiffs(temp.global))
```

```
## [1] 1
```

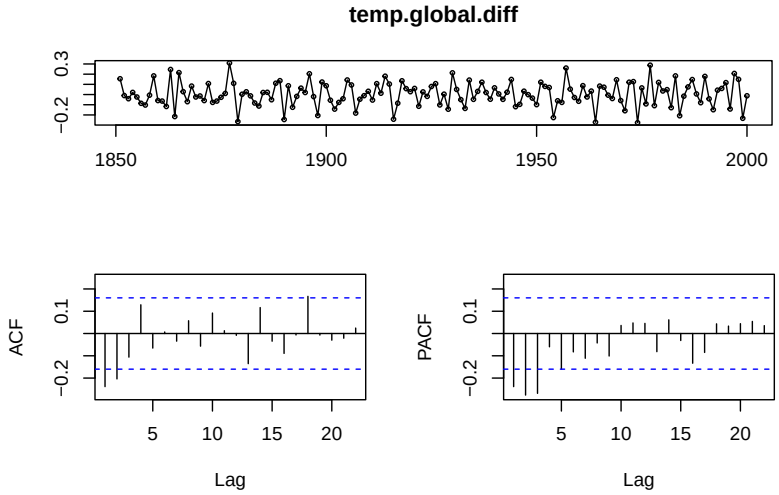
```
temp.global.diff <- diff(temp.global, diff=d)
plot(temp.global.diff)
```



Come si vede, la varianza è stazionaria e la serie trasformata non mostra tendenze.

A questo punto applichiamo quindi le funzioni di autocorrelazione (`acf`) e di autocorrelazione parziale (`pacf`) per identificare i parametri rispettivamente q e p del modello ARIMA(p, d, q), avendo già identificato d con il comando `ndiffs`.

```
# Separatamente:
## Pacf(temp.global.diff)
## Acf(temp.global.diff)
# in alternativa:
tsdisplay(temp.global.diff)
```



La `pacf` mostra 3 picchi prima del drop-off, quindi $p = 3$. Analogamente, anche la `acf` mostra due picchi prima del drop-off, quindi $q = 2$

Possiamo effettuare la regressione ARIMA con i parametri $(3, 1, 2)$. Utilizziamo la funzione `Arima` della libreria `forecast` anziché la versione standard `arima`, dato che la prima consente anche di considerare il trend (o drift) e di specificare il parametro λ della trasformazione. Per confronto, verifichiamo anche il modello ottenuto con `auto.arima`, laciando anche stimare il valore appropriato di λ con `lambda="auto"`:

```
fit <- Arima(temp.global, order=c(3, 1, 2), include.drift = T, lambda=1)
summary(fit)
```

```
## Series: temp.global
## ARIMA(3,1,2) with drift
## Box Cox transformation: lambda= 1
##
```

```
## Coefficients:
##           ar1     ar2      ar3     ma1      ma2   drift
##       -0.6007  0.0957  -0.2149  0.1703  -0.6394  0.0043
## s.e.   0.1468  0.1687   0.0961  0.1356   0.1264  0.0026
##
## sigma^2 estimated as 0.01103:  log likelihood=127.89
## AIC=-241.79   AICc=-241   BIC=-220.71
##
## Training set error measures:
##                        ME      RMSE        MAE      MPE     MAPE      MASE
## Training set 0.0003292531 0.1025512 0.08439941 5.178568 96.67594 0.9003564
##                     ACF1
## Training set 0.003542436
```

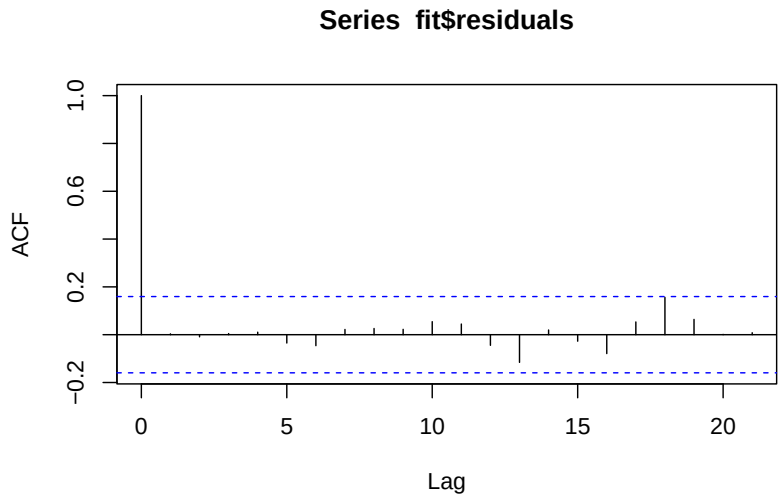
```
fit.auto <- auto.arima(temp.global, lambda="auto")
summary(fit.auto)
```

```
## Series: temp.global
## ARIMA(3,1,2) with drift
## Box Cox transformation: lambda= 0.7753935
##
## Coefficients:
##           ar1     ar2      ar3     ma1      ma2   drift
##       -0.5966  0.0394  -0.2764  0.1745  -0.5912  0.0072
## s.e.   0.1358  0.1696   0.0955  0.1300   0.1255  0.0043
##
## sigma^2 estimated as 0.02741:  log likelihood=59.54
## AIC=-105.08   AICc=-104.29   BIC=-84.01
##
## Training set error measures:
##                         ME      RMSE        MAE      MPE     MAPE      MASE
## Training set -0.004164867 0.1029321 0.08476047 10.13711 87.57378 0.9042081
##                     ACF1
## Training set 0.00687904
```

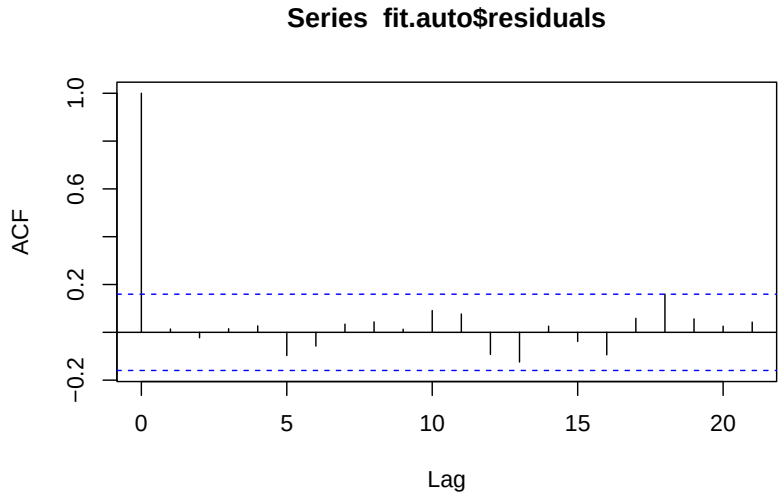
Come si osserva, la versione automatica propone lo stesso modello ARIMA$(3, 1, 2)$.

Il prossimo passo è verificare i residui: perché il modello sia adeguato, essi devo essere casuali e normali. La casualità può essere studiata con la `acf`: se la serie temporale è casuale, l'unico indice di correlazione deve essere il primo.

```
acf(fit$residuals)
```

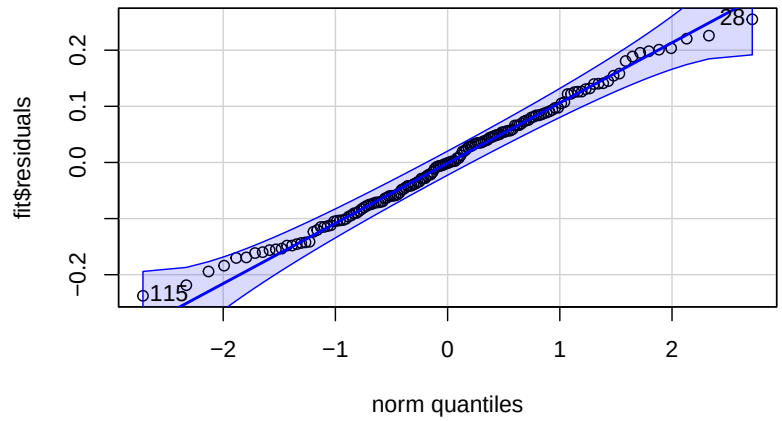



```
acf(fit.auto$residuals)
```



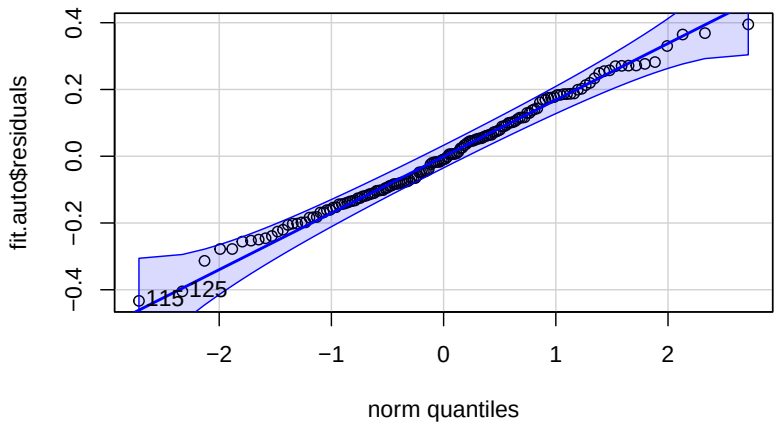
La normalità può essere studiata al solito con un diagramma Q-Q:

```
qqPlot(fit$residuals)
```




```
## [1]  28 115
```

```
qqPlot(fit.auto$residuals)
```

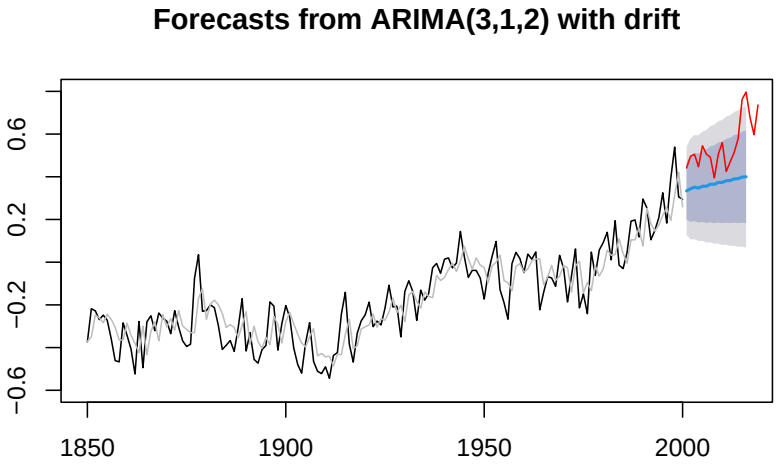



```
## [1] 115 125
```

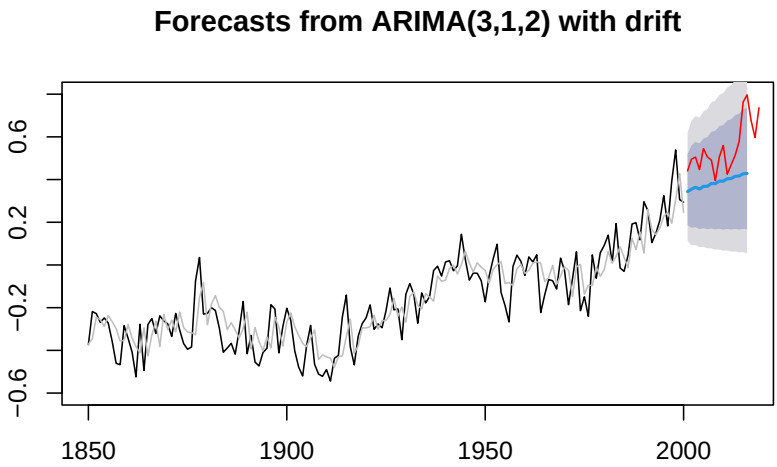
Entrambi i modelli risultano quindi adeguati.

Possiamo infine verificare la predizione, confrontandola con i dati successivi al 2000, per entrambi i modelli:

```
plot(forecast(fit, h=16), ylim=c(-0.6, 0.8))
lines(temp.global.test, col="red")
lines(fit$fitted, col="gray")
```




```
plot(forecast(fit.auto, h=16), ylim=c(-0.6, 0.8))
lines(temp.global.test, col="red")
lines(fit.auto$fitted, col="gray")
```

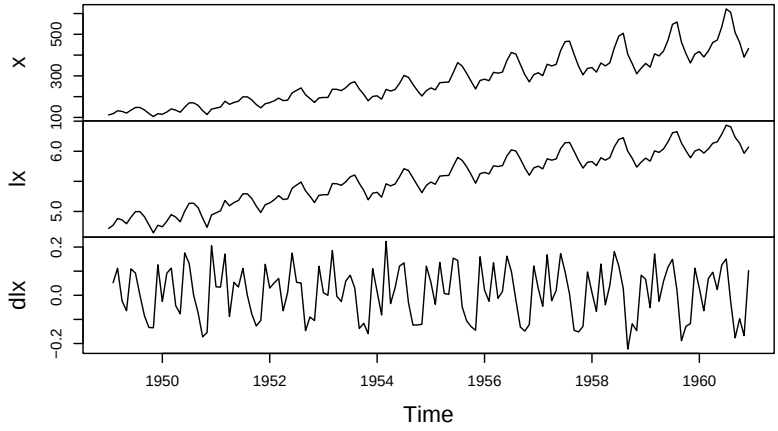



```
## ggplot:
# autoplot(forecast(fit, h=16)) + geom_line(aes(x=index(fit$x), y=fit$fitted), color="gray") + geom_lin
```

2.3.3 Esempio: Seasonal ARIMA (SARIMA)

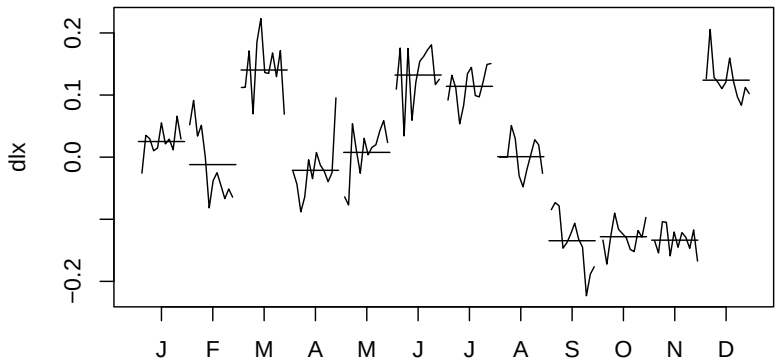
Consideriamo l'effetto della stagionalità. Utilizziamo la serie storica `AirPassengers` integrata in R.

```
x <- AirPassengers
lx <- log(x) # logaritmo per stabilizzare la varianza
dlx = diff(lx) # prima differenziazione
plot.ts(cbind(x,lx,dlx), main="")
```



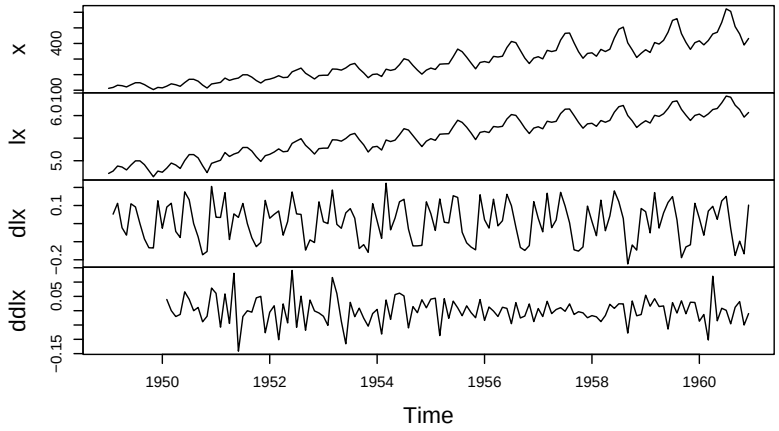
La serie `dlx` mostra ancora un evidente periodicità stagionale. Questa può essere evidenziata mediante la funzione `monthplot()`, che raggruppa anni diversi per lo stesso mese:

```
monthplot(dlx)
```

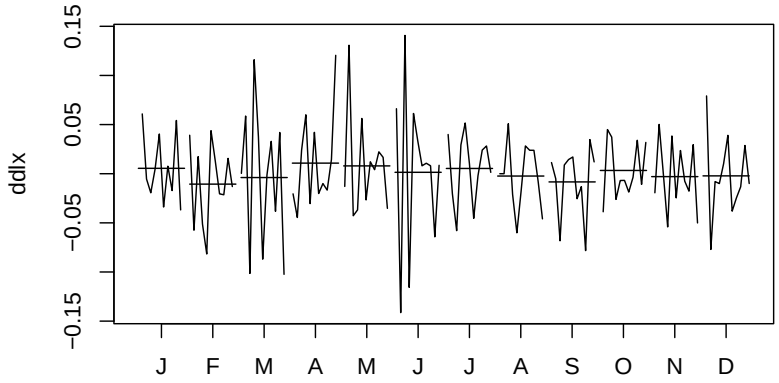


È evidente come i valori per lo stesso mese tendono a raggrupparsi. Possiamo quindi provare a differenziare con lag 12 oltre che con lag 1:

```
ddlx <- diff(dlx, 12)
plot.ts(cbind(x,lx,dlx, ddlx), main="")
```

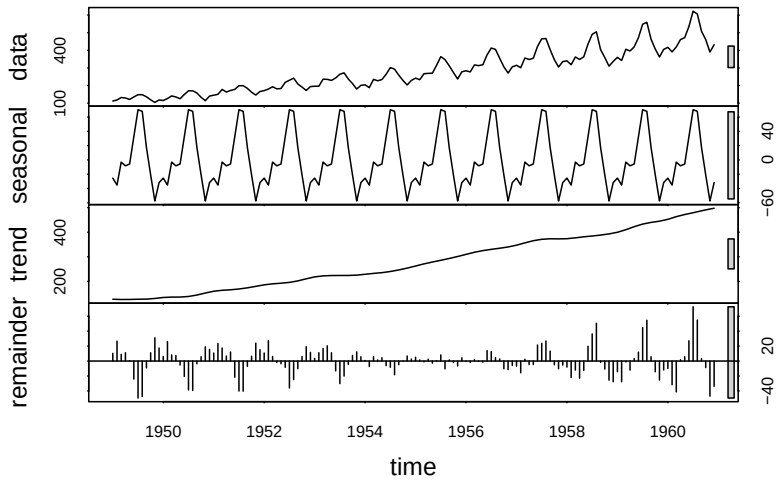



```
monthplot(ddlx)
```



La stagionalità può essere analizzata anche con il metodo `stl()`:

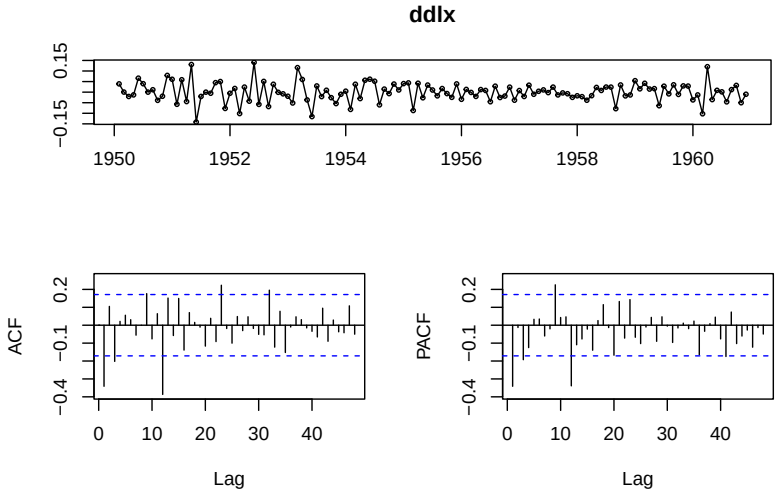
```
plot(stl(x, "periodic"))
```



Si noti che `stl(x, "periodic")` restituisce un oggetto `ts` multi-variato, le cui colonne possono essere estratte, ad es., così: `plot(s$time.series[,"seasonal"])`.

A questo punto studiamo l'autocorrelazione per identificare i parametri del modello SARIMA:

```
tsdisplay(ddlx, lag.max = 4*12)
```



Anzitutto, per eliminare il trend abbiamo differenziato 1 volta sia a lag 1 che a lag 12, quindi i parametri d della parte stagionale e di quella non stagionale saranno entrambi 1. In formula, si scrive che il modello trasformato è $\nabla_{12}\nabla \log x_t$.

Per quanto riguarda i parametri p e q, entrambi i diagrammi di autocorrelazione mostrano un forte picco a lag 12 (riprova della stagionalità) e entrambi i grafici mostrano una rapida caduta verso un'oscillazione stabilizzata: dopo un picco a lag 1, sia la PACF che la ACF passano all'oscillazione stabilizzata, quindi $p = 1$ e $q = 1$. Dopo il picco a lag 12, invece, la PACF mostra una decrescita geometrica, il che indica il termine $p = 0$ (modello AR), mentre la ACF mostra un rapido smorzamento subito dopo il primo picco, che indica $q = 1$ nel modello MA. Secondo la notazione comune, il modello appropriato è quindi ARIMA$(1,1,1) \times (0,1,1)_{12}$, ovvero un modello stagionale con lag 12 con parametri $(1,1,1)$ per la parte non-stagionale, e $(0,1,1)$ per la parte stagionale.

```
(fit1 <- arima(lx, order=c(1,1,1), seasonal=list(order=c(0,1,1), period = 12)))
```

```
##
## Call:
## arima(x = lx, order = c(1, 1, 1), seasonal = list(order = c(0, 1, 1), period = 12))
```

```
##
## Coefficients:
##          ar1      ma1     sma1
##       0.1960  -0.5784  -0.5643
## s.e.  0.2475   0.2132   0.0747
##
## sigma^2 estimated as 0.001341:  log likelihood = 244.95,  aic = -481.9
```

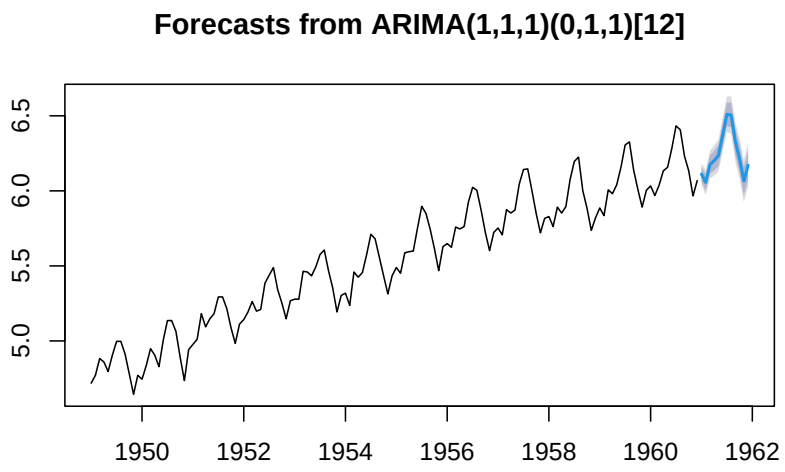
Per sicurezza valutiamo anche il modello ARIMA$(1,1,1) \times (1,1,1)_{12}$:

```
(fit2 <- arima(lx, order=c(1,1,1), seasonal=list(order=c(1,1,1), period = 12)))
```

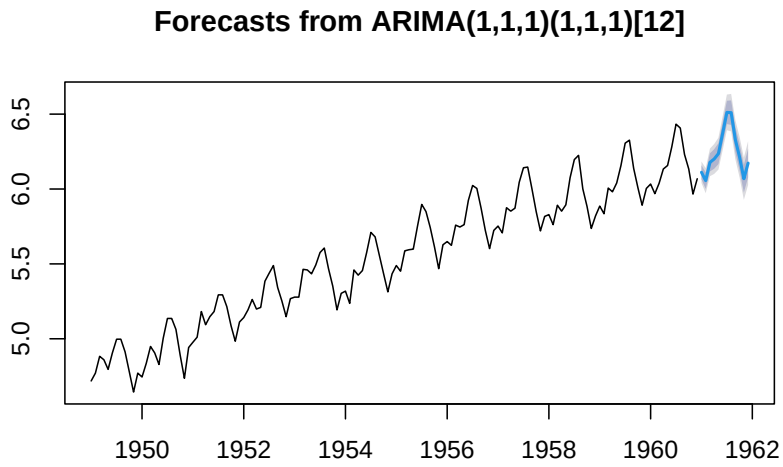
```
##
## Call:
## arima(x = lx, order = c(1, 1, 1), seasonal = list(order = c(1, 1, 1), period = 12))
##
## Coefficients:
##          ar1      ma1    sar1     sma1
##       0.1666  -0.5615  -0.099  -0.4973
## s.e.  0.2459   0.2115   0.154   0.1360
##
## sigma^2 estimated as 0.001336:  log likelihood = 245.16,  aic = -480.31
```

Come si nota, il valore di AIC è leggermente inferiore, quindi potremmo adottare il secondo modello ed effettuare una predizione per i successivi 12 mesi:

```
plot(forecast(fit1, h=12))
```




```
plot(forecast(fit2, h=12))
```



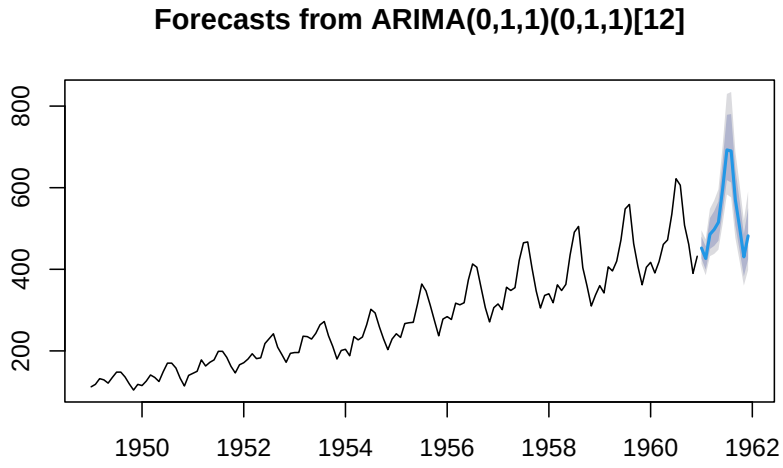
Si noti che i grafici sopra riportano la regressione di `lx`, che è `log(x)`, quindi la scala delle ordinate andrebbe opportunamente anti-trasformata. Purtroppo gli oggetti `fit1` e `fit2` non sono immediatamente trasformabili: sarebbe necessario scomporli nei dati originali e poi applicare l'esponenziale `exp()`. Per questo motivo **è sempre preferibile applicare le trasformazioni con il parametro `lambda` della funzione `arima()`**, dato che è trasparentemente gestito nei plot. In questo esempio si è preferito trasformare manualmente la serie storica in modo da rendere più chiara la logica del processo.

Affidandoci agli automatismi, possiamo ridurre quanto sopra a:

```
(fit3 <- auto.arima(x, lambda="auto"))
```

```
## Series: x
## ARIMA(0,1,1)(0,1,1)[12]
## Box Cox transformation: lambda= -0.2947046
##
## Coefficients:
##           ma1     sma1
##       -0.4355  -0.5847
## s.e.   0.0908   0.0725
##
## sigma^2 estimated as 5.856e-05:  log likelihood=451.59
## AIC=-897.18   AICc=-896.99   BIC=-888.55
```

```
plot(forecast(fit3, h=12))
```

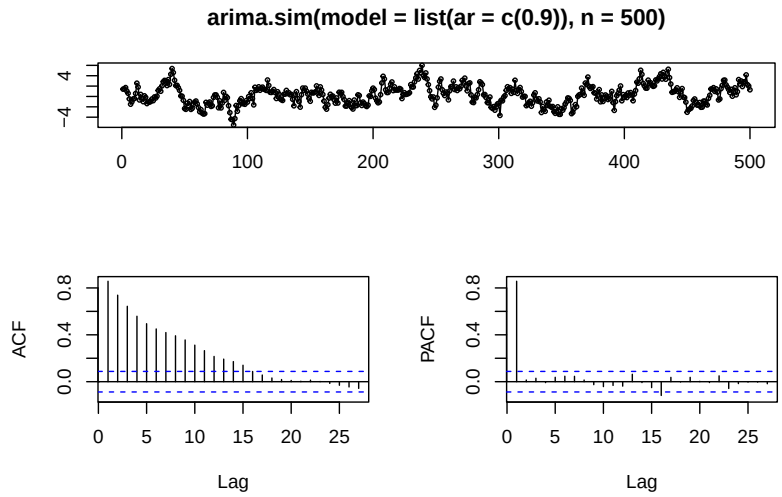


3 Simulazione di processi ARIMA

Per motivi di studio è spesso utile poter *simulare* un processo ARIMA. A questo scopo possiamo utilizzare la funzione `arima.sim()`, che genera una serie temporale a partire dai termini p, d, e q del modello desiderato.

Vediamo ad esempio un processo auto-regressivo di tipo AR(1):

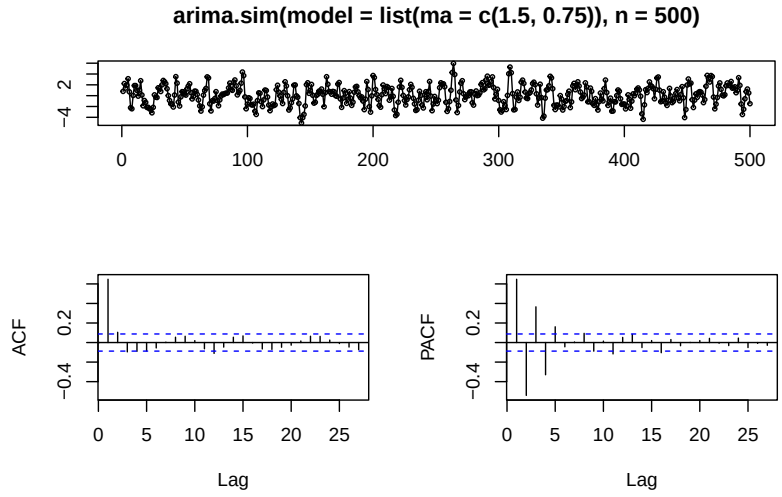
```
set.seed(123)
tsdisplay(arima.sim(model=list(ar=c(0.9)), n=500))
```



Come si vede, la ACF degrada in maniera geometrica mentre la PACF ha un brusco calo sotto la soglia di significatività a lag=1, indice appunto di un modello con $p = 1$

Simuliamo invece un processo a media mobile MA(2):

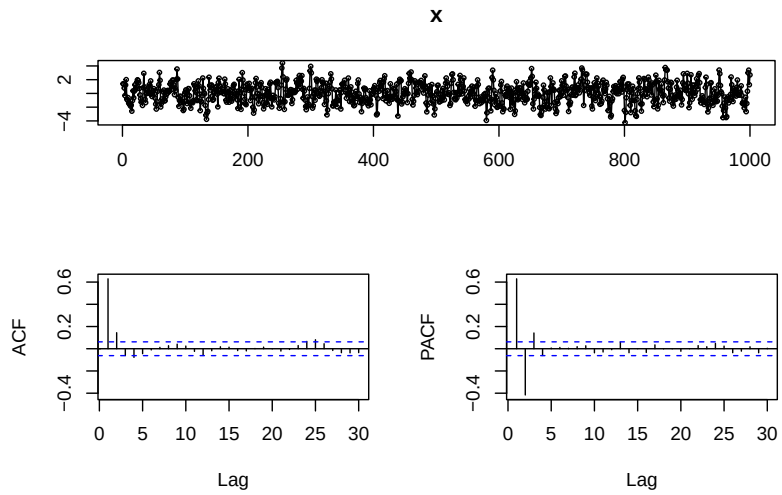
```
set.seed(123)
tsdisplay(arima.sim(model=list(ma=c(1.5, 0.75)), n=500))
```



Questa volta è la PACF a diminuire geometricamente (seppure con segni alternati), mentre la ACF si smorza rapidamente dopo due lag, per cui si deduce $q = 2$.

Vediamo ora l'effetto combinato, ARMA$(1, 2)$, per cui ci aspettiamo $p = 1$ e $q = 2$:

```
set.seed(123)
x <- arima.sim(model=list(ar=c(0.6, -0.2), ma=c(0.4)), n=1000)
tsdisplay(x)
```



Come si vede, quando entrambi i termini sono presenti i grafici ACF e PACF possono non essere facilmente interpretabili. In questo caso, ad esempio, saremmo portati a proporre un modello ARMA$(3, 2)$. Per questo motivo è opportuno, partendo da questa ipotesi, valutare anche condizioni simili e scegliere quella con AIC minimo, che è appunto ciò che fa `auto.arima()`, di cui possiamo vedere il processo mediante il parametro `trace=T`:

```
auto.arima(x, start.p=3, start.q=2, trace=T)
```

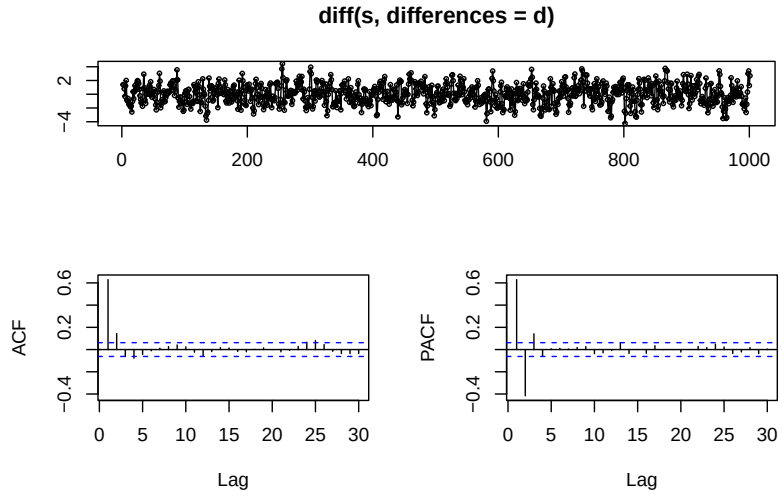
```
##
##  Fitting models using approximations to speed things up...
##
##  ARIMA(3,0,2) with non-zero mean : 2848.929
##  ARIMA(0,0,0) with non-zero mean : 3564.117
##  ARIMA(1,0,0) with non-zero mean : 3059.461
##  ARIMA(0,0,1) with non-zero mean : 2948.729
##  ARIMA(0,0,0) with zero mean     : 3562.742
##  ARIMA(2,0,2) with non-zero mean : 2846.754
##  ARIMA(1,0,2) with non-zero mean : 2847.234
##  ARIMA(2,0,1) with non-zero mean : 2844.742
##  ARIMA(1,0,1) with non-zero mean : 2861.169
##  ARIMA(2,0,0) with non-zero mean : 2869.806
##  ARIMA(3,0,1) with non-zero mean : 2847.434
##  ARIMA(3,0,0) with non-zero mean : 2849.622
##  ARIMA(2,0,1) with zero mean     : 2843.007
##  ARIMA(1,0,1) with zero mean     : 2859.387
##  ARIMA(2,0,0) with zero mean     : 2868.114
##  ARIMA(3,0,1) with zero mean     : 2845.667
##  ARIMA(2,0,2) with zero mean     : 2845.015
##  ARIMA(1,0,0) with zero mean     : 3057.599
##  ARIMA(1,0,2) with zero mean     : 2845.52
##  ARIMA(3,0,0) with zero mean     : 2847.854
##  ARIMA(3,0,2) with zero mean     : 2847.142
##
##  Now re-fitting the best model(s) without approximations...
##
##  ARIMA(2,0,1) with zero mean     : 2842.923
##
##  Best model: ARIMA(2,0,1) with zero mean

## Series: x
```

```
## ARIMA(2,0,1) with zero mean
##
## Coefficients:
##          ar1      ar2     ma1
##       0.5987  -0.2312  0.3710
## s.e.  0.0621   0.0499  0.0607
##
## sigma^2 estimated as 0.999:  log likelihood=-1417.44
## AIC=2842.88   AICc=2842.92   BIC=2862.51
```

È possibile ovviamente generare una serie temporale con un parametro d non nullo:

```
set.seed(123)
ar <- c(0.6, -0.2)
ma <- c(0.4)
d <- 1
order <- c(length(ar), d, length(ma))
s <- arima.sim(model=list(ar=ar, ma=ma, order=order), n=1000)
tsdisplay(diff(s, differences = d))
```



Anche in questo caso possiamo verificare il risultato:

```
auto.arima(s, start.p=3, start.q=2, trace=T)
```

```
##
##  Fitting models using approximations to speed things up...
##
##  ARIMA(3,1,2) with drift         : 2848.206
##  ARIMA(0,1,0) with drift         : 3563.394
##  ARIMA(1,1,0) with drift         : 3058.739
##  ARIMA(0,1,1) with drift         : 2948.006
##  ARIMA(0,1,0)                    : 3562.019
##  ARIMA(2,1,2) with drift         : 2846.031
##  ARIMA(1,1,2) with drift         : 2846.511
##  ARIMA(2,1,1) with drift         : 2844.019
##  ARIMA(1,1,1) with drift         : 2860.446
##  ARIMA(2,1,0) with drift         : 2869.083
##  ARIMA(3,1,1) with drift         : 2846.711
##  ARIMA(3,1,0) with drift         : 2848.9
##  ARIMA(2,1,1)                    : 2842.284
```

```
##  ARIMA(1,1,1)                    : 2858.664
##  ARIMA(2,1,0)                    : 2867.391
##  ARIMA(3,1,1)                    : 2844.944
##  ARIMA(2,1,2)                    : 2844.292
##  ARIMA(1,1,0)                    : 3056.876
##  ARIMA(1,1,2)                    : 2844.797
##  ARIMA(3,1,0)                    : 2847.132
##  ARIMA(3,1,2)                    : 2846.419
##
##  Now re-fitting the best model(s) without approximations...
##
##  ARIMA(2,1,1)                    : 2842.923
##
##  Best model: ARIMA(2,1,1)
```

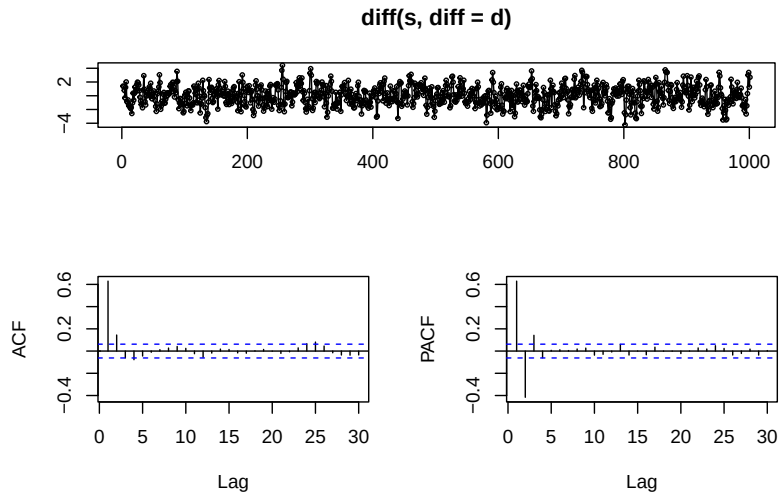
```
## Series: s
## ARIMA(2,1,1)
##
## Coefficients:
##          ar1      ar2     ma1
##       0.5987  -0.2312  0.3710
## s.e.  0.0621   0.0499  0.0607
##
## sigma^2 estimated as 0.999:  log likelihood=-1417.44
## AIC=2842.88   AICc=2842.92   BIC=2862.51
```

Il numero di differenziazioni necessarie per rendere la serie stazionaria può essere calcolato con `ndiffs()`:

```
(d <- ndiffs(s))
```

```
## [1] 1
```

```
tsdisplay(diff(s, diff=d))
```



Se non volessimo utilizzare `auto.arima()`, potremmo verificare l'AIC di una combinazione di parametri esplorati a tappeto. Le funzioni di autocorrelazione suggeriscono un modello ARIMA$(3, 1, 2)$. Il modello ottimale dovrebbe avere quindi una combinazione di p e q inferiori a 3 e 2. Li proviamo tutti e selezioniamo quello con AIC minore:

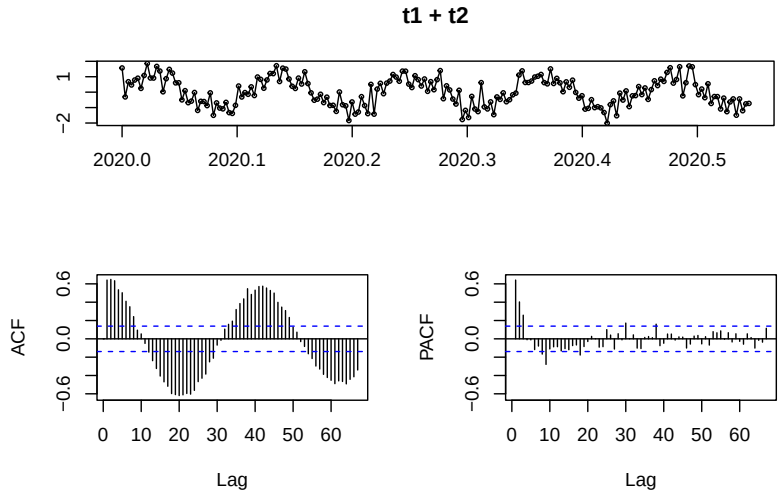
```
g <- expand.grid(p=1:3, q=1:2, drift=c(F, T), aic=NA)
for (i in 1:dim(g)[1]) {
```

```
  g$aic[i] <- Arima(s, order=c(g$p[i], d, g$q[i]), include.drift=g$drift[i])$aic
}
g[which.min(g$aic),]
```

```
##   p q drift      aic
## 2 2 1 FALSE 2842.883
```

Vediamo come si comporta `auto.arima()` su serie temporali generate in altro modo:

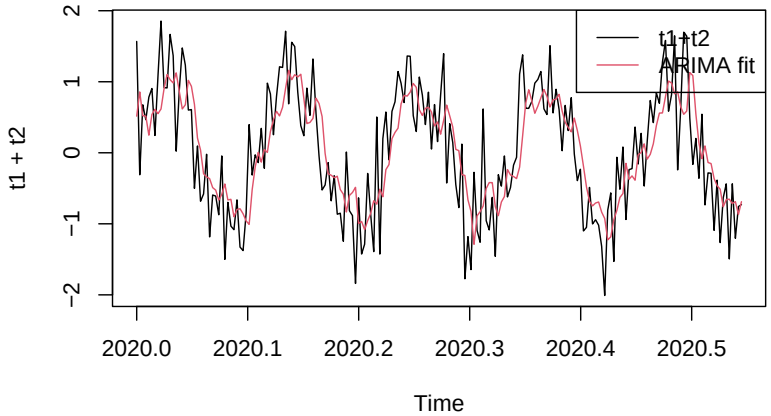
```
n <- 200
t1 <- ts(0.5*rnorm(n), start=2020, frequency = 365.25)
t2 <- ts(sin((1:n)*365.25/(12*n)), start=2020, frequency=365.25)
tsdisplay(t1+t2)
```




```
fit <- auto.arima(t1+t2)
```

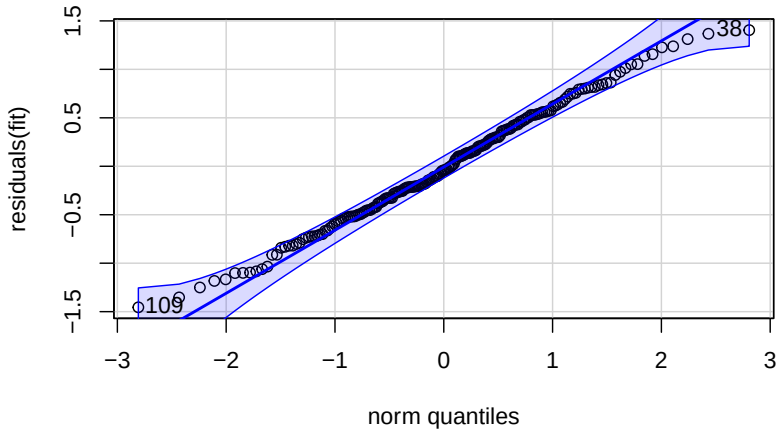
Possiamo visualizzare la *regressione*, ossia il termine `fit$fitted`:

```
plot(t1+t2, col=1)
lines(fit$fitted, col=2)
legend("topright", legend=c("t1+t2", "ARIMA fit"), lty=1, col=1:2)
```



Infine, ricordiamo sempre di verificare la normalità dei residui:

```
invisible(qqPlot(residuals(fit)))
```


residuals(fit)
1.5
0.5
−0.5
−1.5
−3
−2
−1
0
1
2
3
norm quantiles
109
38