RecapCap3

2023-12-31

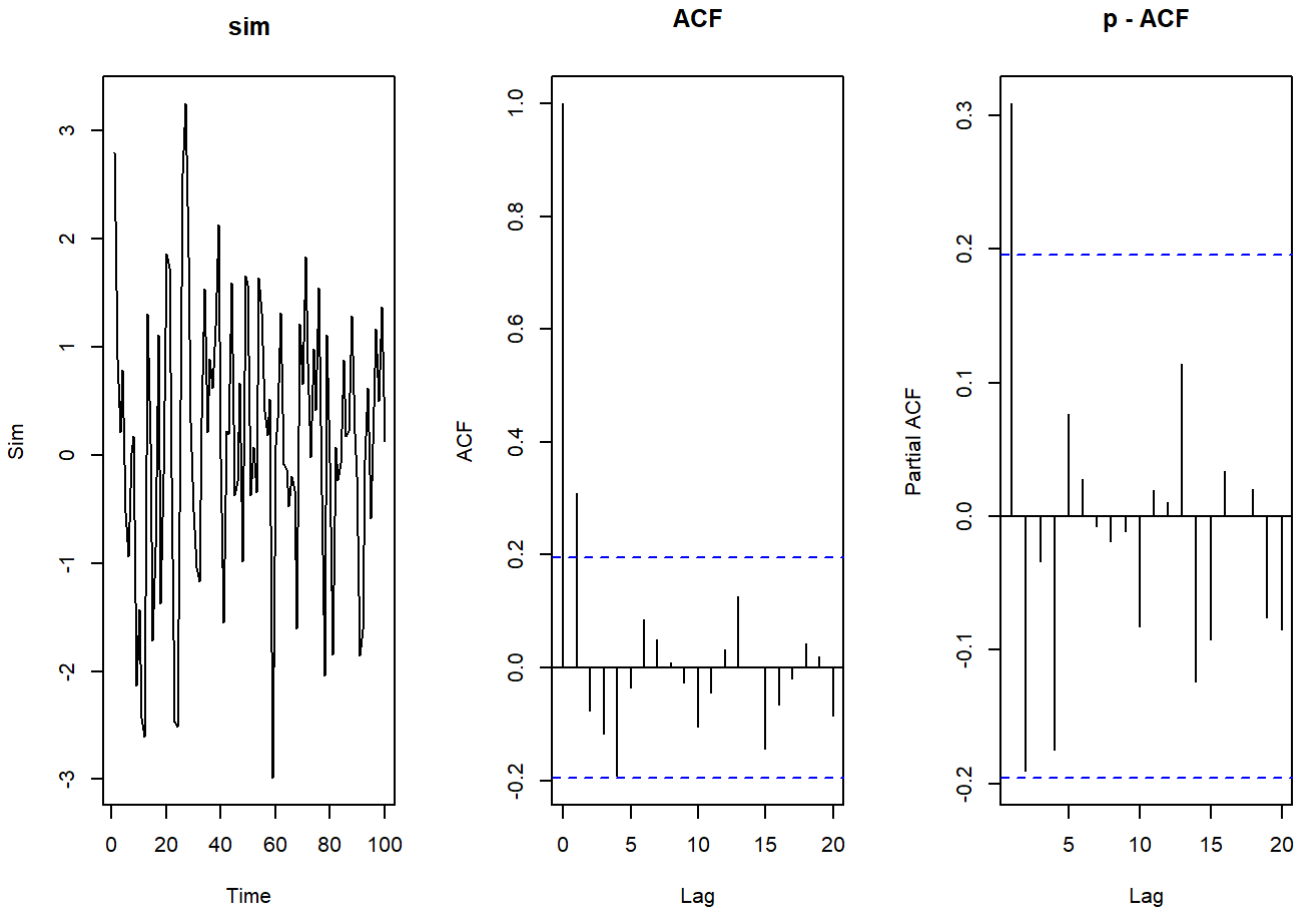
Apprendimento statistico - MASTRANTONIO

TIME SERIES pt. 2

SIMULAZIONE DI UN AR(1) AL VARIARE DI α

Il codice simula una serie temporale stocastica utilizzando un modello autoregressivo di primo ordine AR(1) con errori casuali distribuiti normalmente.

```
x = c() # Valori della serie temporale simulata
alpha = 0.3 # Specifica il coefficiente di autoregressione, che controlla quanto l'osservazio
ne corrente dipenda dall'osservazione precedente
sigma2 = 1.5 # Specifica la varianza degli errori casuali nella serie temporale

# Genera il primo valore della serie temporale utilizzando una distribuzione N(0, sigma2 divi
sa per (1-alpha^2))
x[1] = rnorm(1, 0, (sigma2/(1-alpha^2))^0.5)

# Simula i valori successivi della serie temporale utilizzando il modello AR(1)
for (i in 2:100){
  x[i] = rnorm (1, alpha *x[i-1], sigma2^0.5)
}

par(mfrow = c(1, 3))
plot (x, type = "l", main = "sim", xlab = "Time", ylab = "Sim") # Crea un grafico della serie
temporale simulata
acf(x, main = "ACF") # Calcola e visualizza la funzione di autocorrelazione (ACF) della serie
temporale
pacf(x, main = "p - ACF") # Calcola e visualizza la funzione di autocorrelazione parziale (PA
CF) della serie temporale
```



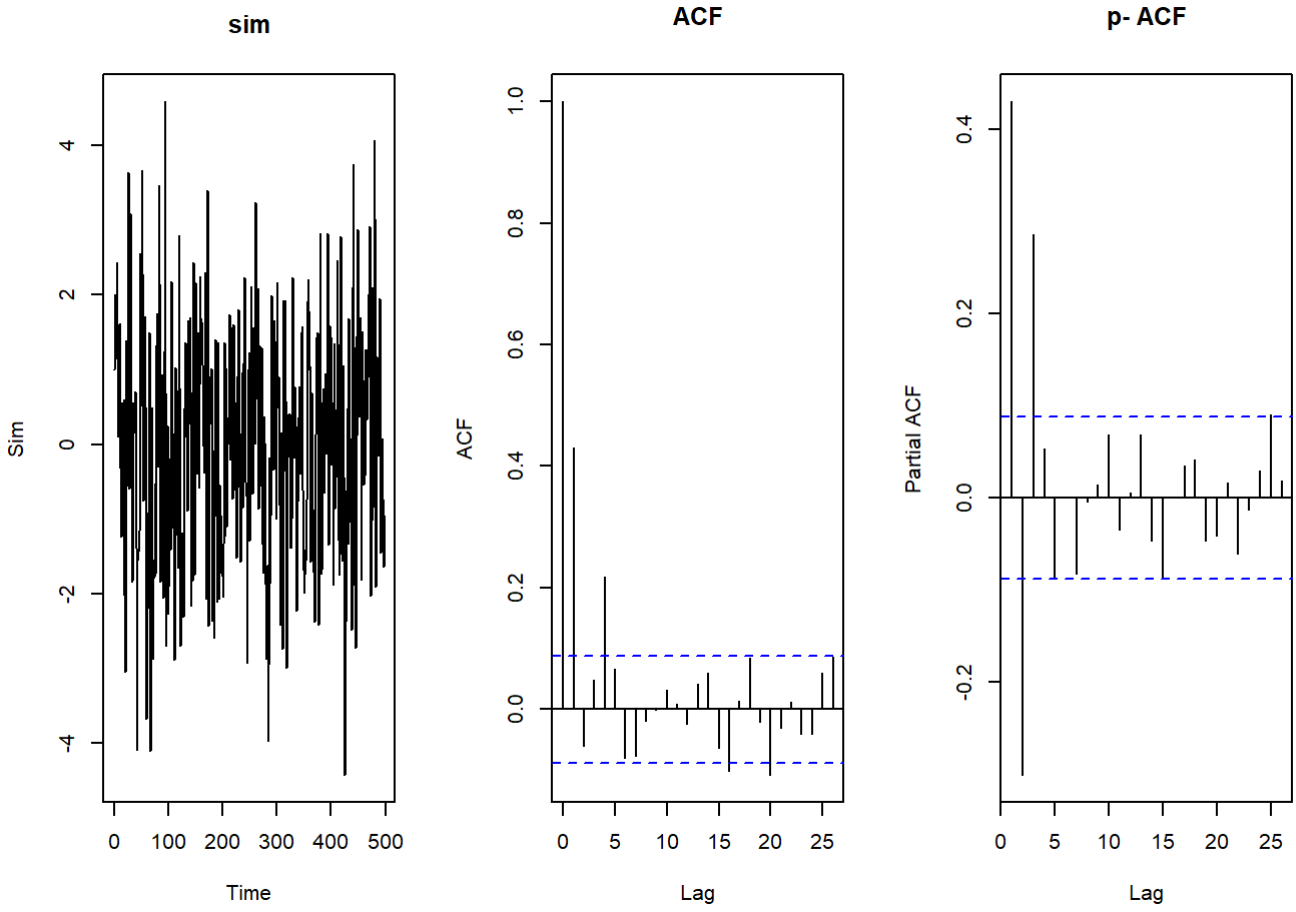
In sintesi, il codice simula una serie temporale AR(1) con errori casuali normalmente distribuiti e visualizza il grafico della serie temporale insieme alla sua funzione di autocorrelazione e autocorrelazione parziale.

Lo script è riutilizzabile variando i valori di α ad esempio uguali a $0.9, -0.3, -0.9$.

SIMULAZIONE DI UN AR(p)

Il codice genera una serie temporale simulata utilizzando un modello autoregressivo di ordine 3 (AR(3)) con errori casuali normalmente distribuiti.

```r
x = c() # Utilizzato per immagazzinare la serie temporale simulata

# Coefficienti del modello AR(3)
alpha1 = 0.7
alpha2 = -0.5
alpha3 = 0.3

sigma2 = 1.5 # Varianza degli errori casuali normalmente distribuiti

# Inizializzazione dei primi valori di x
x[1] = 0
x[2] = 1
x[3] = 2

# Ogni valore successivo di x viene generato utilizzando la funzione rnorm che genera un nume
ro casuale da una distribuzione normale con media e deviazione standard calcolate in base ai
valori precedenti della serie temporale e ai coefficienti del modello AR(3)
for(i in 4:500){
  x[i] = rnorm(1, alpha1*x[i -1]+alpha2*x[i -2]+alpha3*x[i -3], sigma2^0.5)
}

# Viene eliminato il primo e l'ultimo valore di x (i valori 1 e 400)
x = x[-c(1 ,400)]

par( mfrow =c(1 ,3))
plot(x, type ="l", main = "sim", xlab = "Time", ylab = "Sim") # Grafico della serie temporale
acf(x, main ="ACF") # Grafico della funzione di autocorrelazione
pacf(x, main = "p- ACF") # Grafico della funzione di autocorrelazione parziale
```

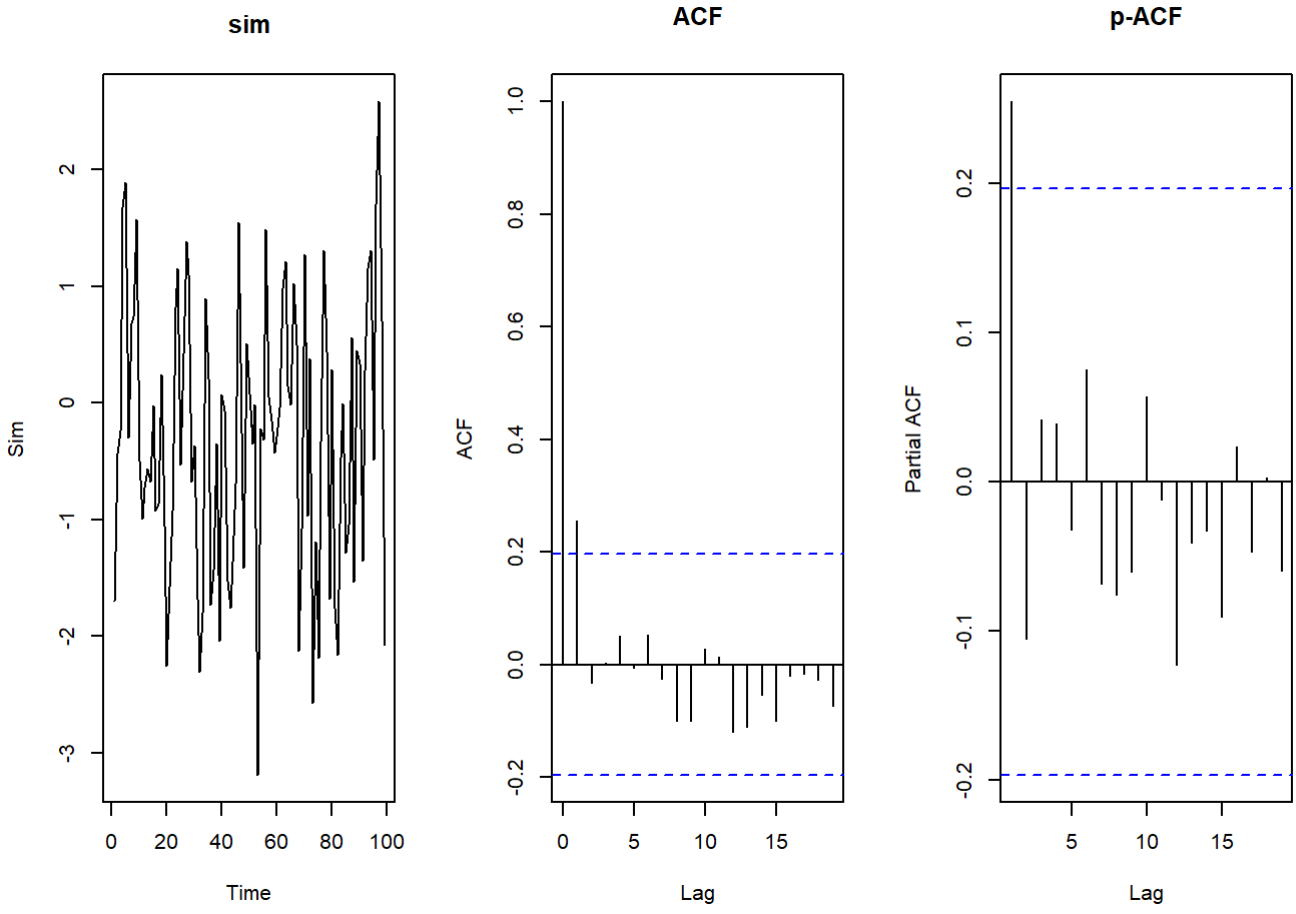


In breve, il codice simula una serie temporale AR(3) con errori casuali normalmente distribuiti e visualizza la serie temporale insieme alle sue funzioni di autocorrelazione e autocorrelazione parziale.

SIMULAZIONE DI UN MA(1) CON $\beta = 0.3$

Il codice genera una serie temporale simulata utilizzando un modello di moving average di ordine 1 (MA(1)) con un termine di errore casuale normalmente distribuito.

```
x = c() # Utilizzato per immagazzinare la serie temporale simulata

# Rappresenta il coefficiente associato al termine di ritardo nella moving average MA(1)
beta_ = 0.3
sigma2 = 1.5 # Varianza del termine di errore casuale normalmente distribuito
w = rnorm(100, 0, sigma2^0.5) # Viene generato un vettore di errori casuali normalmente distr
ibuiti

x1 = 0

# Viene eseguito un ciclo for per generare la serie temporale. Ogni valore successivo di x è
la somma del termine di errore corrente w[i] e del prodotto del termine di errore precedente
w[i-1] per il coefficiente beta_ associato al termine di ritardo nella moving average
for(i in 2:100){
  x[i] = w[i]+ beta_*w[i-1]
}
x = x[-1] # Viene eliminato il primo valore di x
par(mfrow = c(1,3))
plot(x, type="l", main="sim", xlab="Time", ylab="Sim") # Grafico della serie temporale
acf(x, main="ACF") # Grafico della funzione di autocorrelazione
pacf(x, main ="p-ACF") # Grafico della funzione di autocorrelazione parziale
```



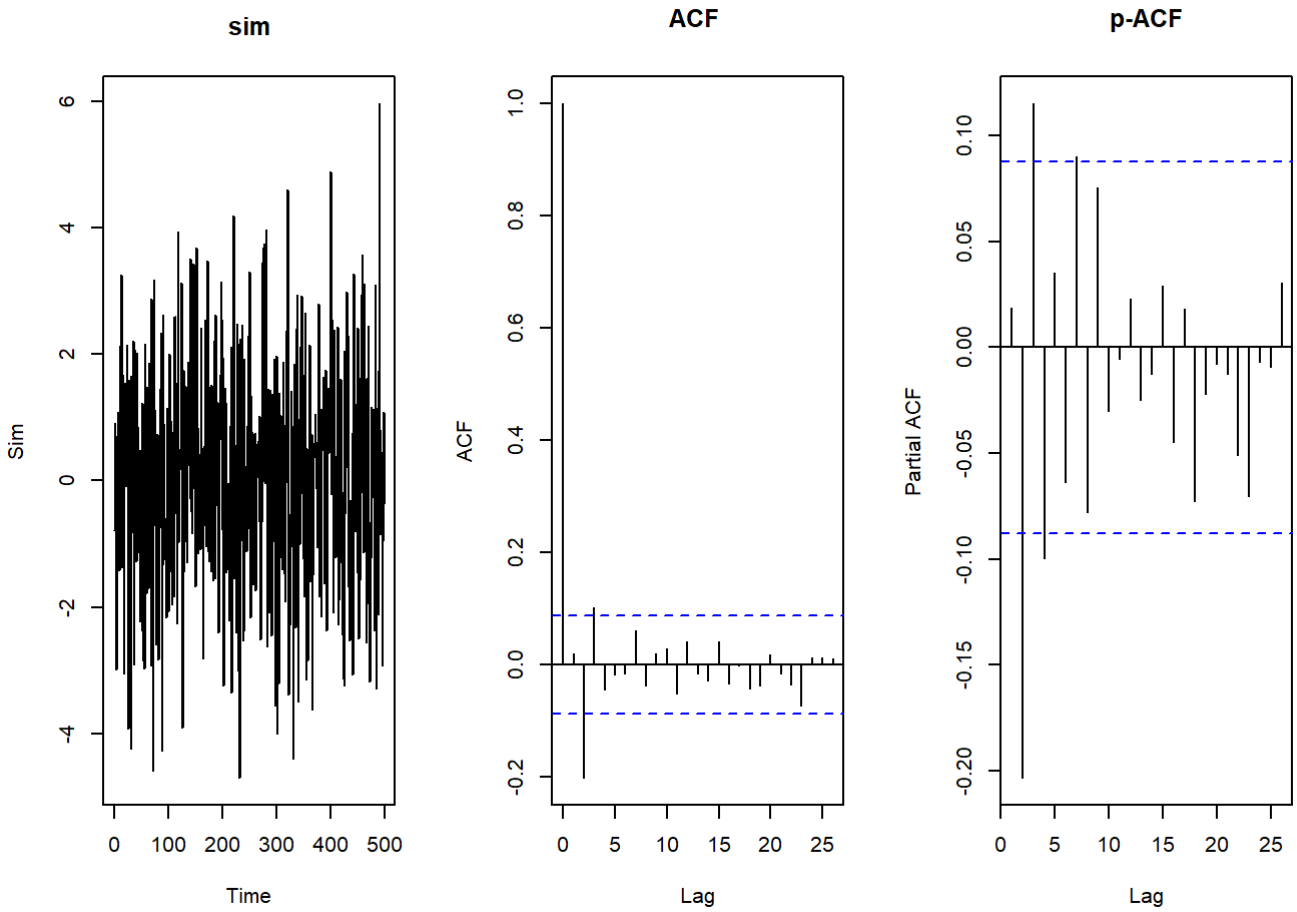
In sintesi, il codice simula una serie temporale MA(1) con un termine di errore casuale normalmente distribuito e visualizza la serie temporale insieme alle sue funzioni di autocorrelazione e autocorrelazione parziale.

Lo script è riutilizzabile variando i valori di α ad esempio uguali a 0.9, -0.3.

SIMULAZIONE DI UN MA(3) CON $ _1=0.8, _2=-0.5, _3=0.3 $

Il codice simula una serie temporale (time series) utilizzando un modello autoregressivo (AR) di ordine 3 con rumore bianco gaussiano.

```
x = c() # Verrà utilizzato per immagazzinare la serie temporale simulata

# Coefficienti del modello autoregressivo
beta1 = 0.8
beta2 = -0.5
beta3 = 0.3

sigma2 = 1.5 # Variabilità del rumore bianco gaussiano
w = rnorm(1000, 0, sigma2^0.5) # Genera un vettore w di lunghezza 1000 contenente valori casu
ali estratti da una distribuzione N(0, sigma)

# Calcola le osservazioni successive della serie temporale x utilizzando il modello autoregre
ssivo di ordine 3 con il rumore bianco w. Ogni osservazione di x è la somma del valore corris
pondente di w e delle tre osservazioni precedenti pesate dai coefficienti beta1, beta2, beta3
x1 = 0
for (i in 4:1000){
  x[i] = w[i] + beta1*w[i-1] + beta2*w[i-2] + beta3*w[i-3]
}

# Rimuove le prime 500 osservazioni per ridurre l'effetto dei valori iniziali influenzati dal
l'inizializzazione a zero
x = x[-c(1:500)]

par(mfrow = c(1,3))
plot(x, type ="l", main="sim", xlab="Time", ylab="Sim") # Plotta la serie temporale simulata
x
acf(x, main="ACF ") # Plotta la funzione di autocorrelazione (ACF) della serie temporale
pacf(x, main="p-ACF") # Plotta la funzione di autocorrelazione parziale (p-ACF) della serie t
emporale
```

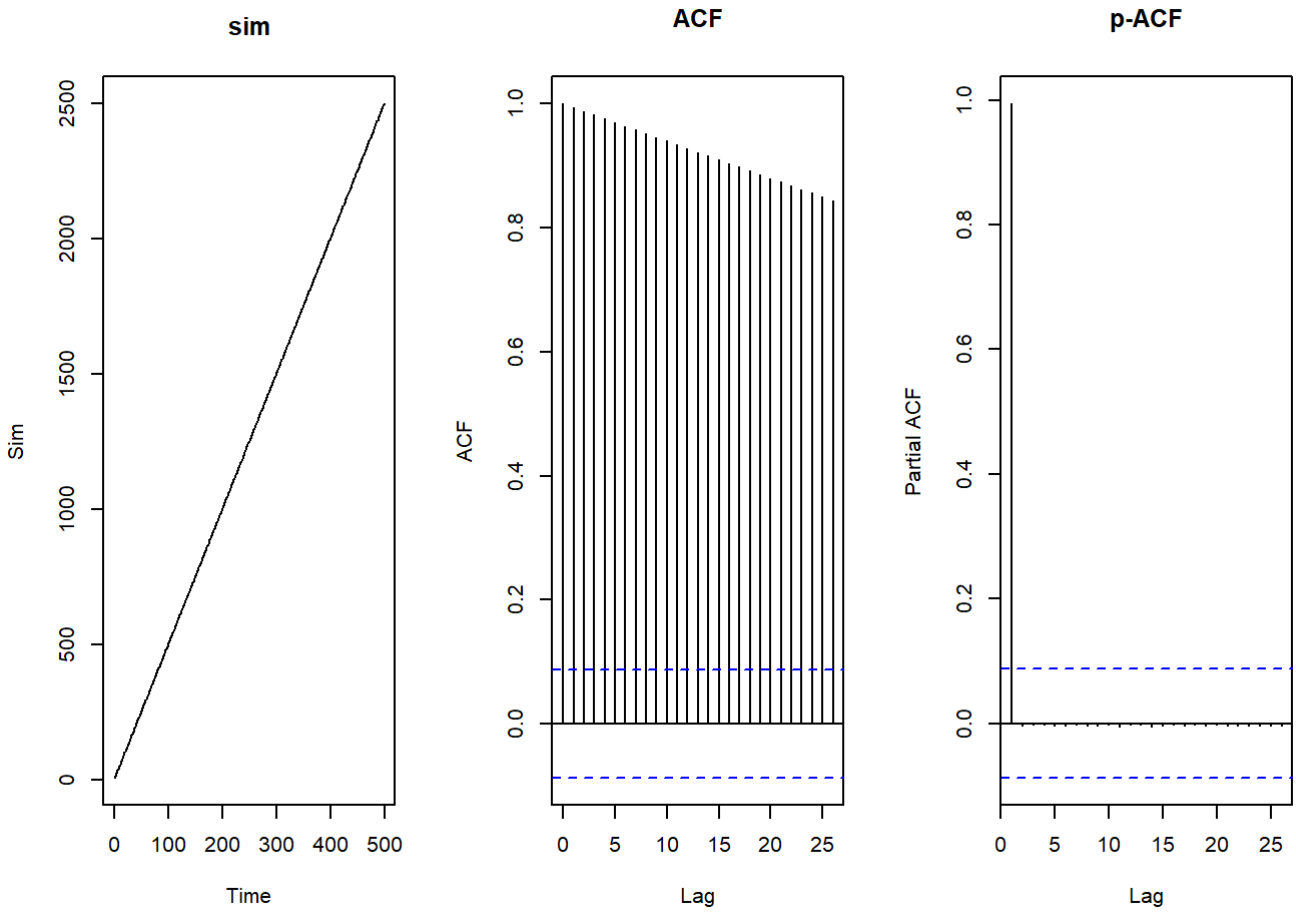


In sintesi, il codice simula una serie temporale AR(3) con rumore bianco gaussiano e visualizza il risultato attraverso un grafico della serie temporale, l'ACF e la p-ACF.

SIMULAZIONE DI UN RUMORE BIANCO CON TREND

Il codice simula una serie temporale utilizzando un processo autoregressivo (AR) di ordine 1 con rumore bianco gaussiano.

```
set.seed (500)
n = 500 # lunghezza della serie temporale
a = 1 # Coefficiente della costante nell'equazione autoregressiva
b = 5 # Coefficiente associato alla variabile temporale nel processo autoregressivo

# Simula la serie temporale utilizzando un modello AR(1) con rumore bianco gaussiano. La medi
a della serie temporale segue una tendenza lineare crescente con coefficiente b, e il termine
di errore è estratto da una distribuzione normale con deviazione standard 2.5^0.5
x = rnorm(n, a+b*1:n, 2.5^0.5)
par(mfrow=c(1,3))
plot(x, type ="l", main="sim", xlab="Time", ylab="Sim") # Plotta la serie temporale simulata
x
acf(x, main="ACF") # Plotta la funzione di autocorrelazione (ACF) della serie temporale
pacf(x, main="p-ACF") # Plotta la funzione di autocorrelazione parziale (p-ACF) della serie t
emporale
```

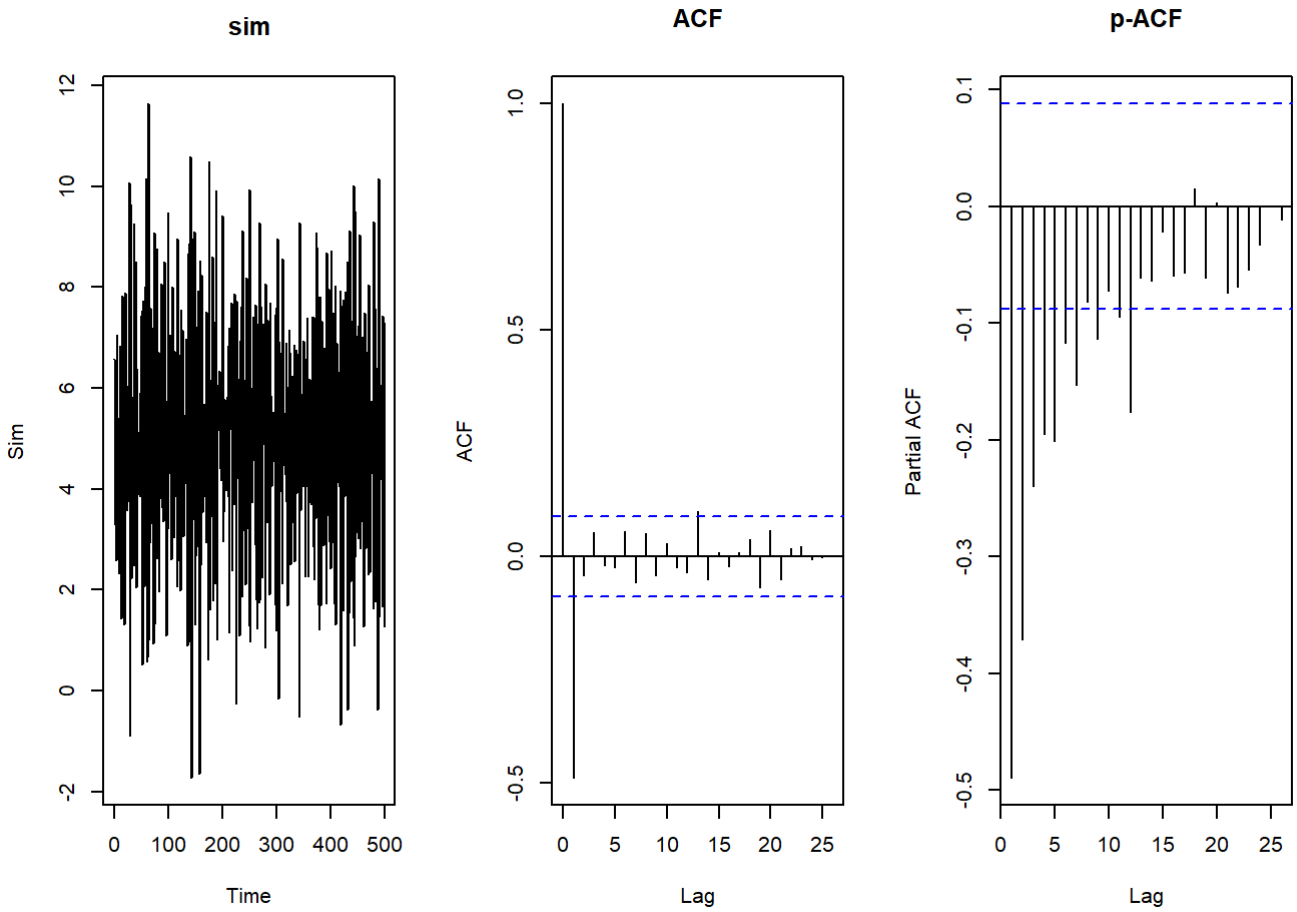


In sintesi, il codice simula una serie temporale AR(1) con rumore bianco gaussiano e visualizza il risultato attraverso un grafico della serie temporale, l'ACF e la p-ACF. La tendenza lineare crescente nella serie temporale è dovuta al termine lineare b*1:n.

SIMULAZIONE DI UN RUMORE BIANCO CON TREND – DIFFERENZE PRIME

Il codice è simile al precedente, ma con una piccola differenza: ora la serie temporale x viene differenziata prima di essere plottata e analizzata per autocorrelazione e autocorrelazione parziale. La differenziazione viene effettuata utilizzando la funzione diff(x).

```
set.seed (500)
n = 500 # lunghezza della serie temporale
a = 1 # Coefficiente della costante nell'equazione autoregressiva
b = 5 # Coefficiente associato alla variabile temporale nel processo autoregressivo

# Simula la serie temporale utilizzando un modello AR(1) con rumore bianco gaussiano. La medi
a della serie temporale segue una tendenza lineare crescente con coefficiente b, e il termine
di errore è estratto da una distribuzione normale con deviazione standard 2.5^0.5
x = rnorm(n, a+b*1:n, 2.5^0.5)
par(mfrow=c(1,3))
plot(diff(x), type ="l", main="sim", xlab="Time", ylab="Sim") # Plotta la serie temporale dif
ferenziata simulata
acf(diff(x), main="ACF") # Plotta la funzione di autocorrelazione (ACF) della serie temporale
differenziata
pacf(diff(x), main="p-ACF") # Plotta la funzione di autocorrelazione parziale (p-ACF) della s
erie temporale differenziata
```

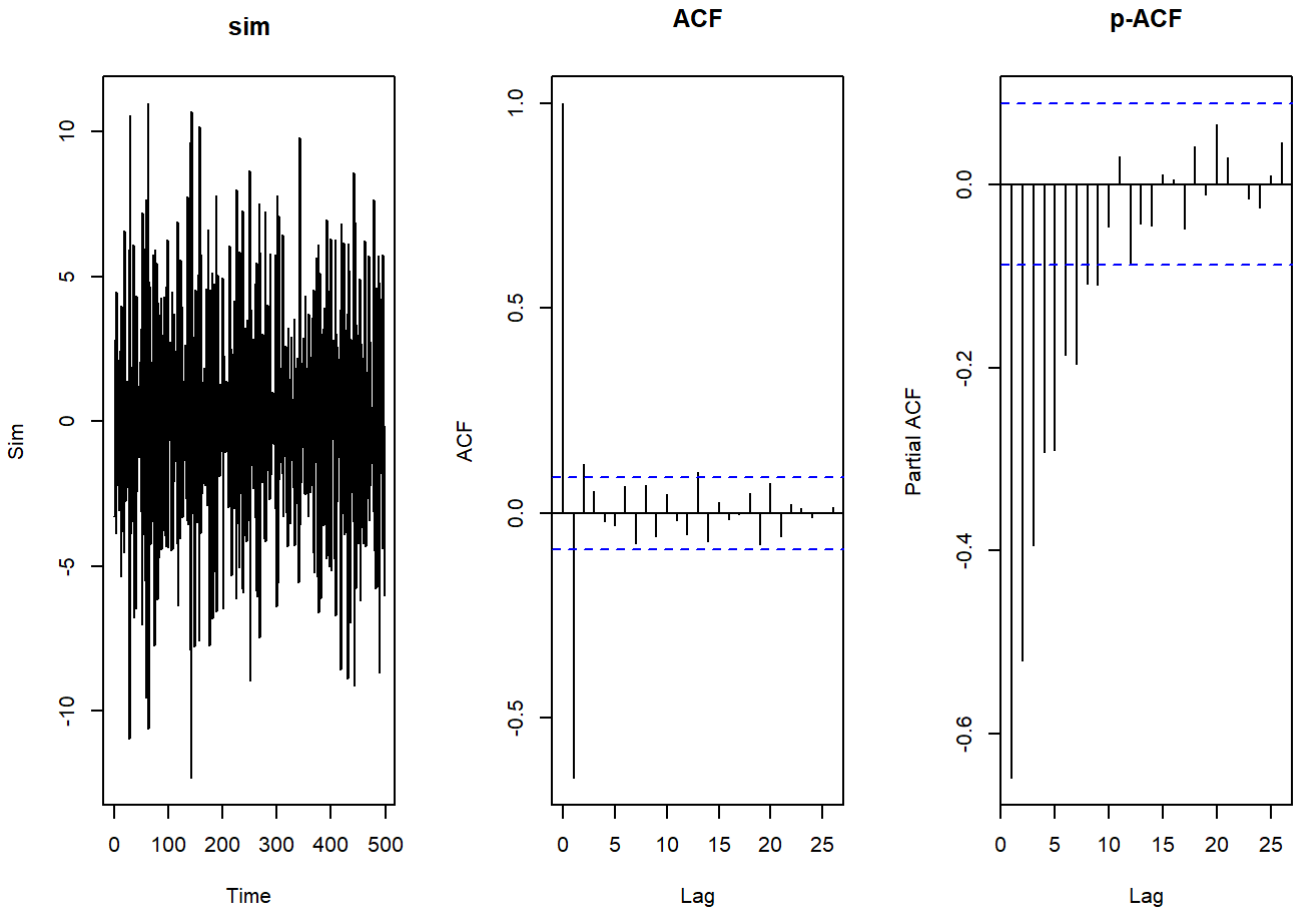


La differenziazione può essere utile per rendere la serie temporale più stazionaria, ovvero per rimuovere tendenze o stagionalità che potrebbero influenzare l'analisi dell'autocorrelazione.

SIMULAZIONE DI UN RUMORE BIANCO CON TREND – DIFFERENZE SECONDE

Il codice effettua una doppia differenziazione sulla serie temporale simulata x.

```
set.seed (500)
n = 500 # Lunghezza della serie temporale
a = 1 # Coefficiente della costante nell'equazione autoregressiva
b = 5 # Coefficiente associato alla variabile temporale nel processo autoregressivo

# Simula la serie temporale utilizzando un modello AR(1) con rumore bianco gaussiano. La medi
a della serie temporale segue una tendenza lineare crescente con coefficiente b, e il termine
di errore è estratto da una distribuzione normale con deviazione standard 2.5^0.5
x = rnorm(n, a+b*1:n, 2.5^0.5)
par(mfrow=c(1,3))
# Plotta la serie temporale che ha subito una doppia differenziazione. La doppia differenziaz
ione può essere utilizzata per rendere la serie ancora più stazionaria, se necessario. La fun
zione diff() calcola la differenza tra valori successivi, e applicarla due volte equivale a c
alcolare la differenza tra valori successivi delle differenze
plot(diff(diff(x)), type ="l", main="sim", xlab="Time", ylab="Sim")
acf(diff(diff(x)), main="ACF") # Plotta la funzione di autocorrelazione (ACF) della serie tem
porale differenziata due volte
pacf(diff(diff(x)), main="p-ACF") # Plotta la funzione di autocorrelazione parziale (p-ACF) d
ella serie temporale differenziata due volte
```

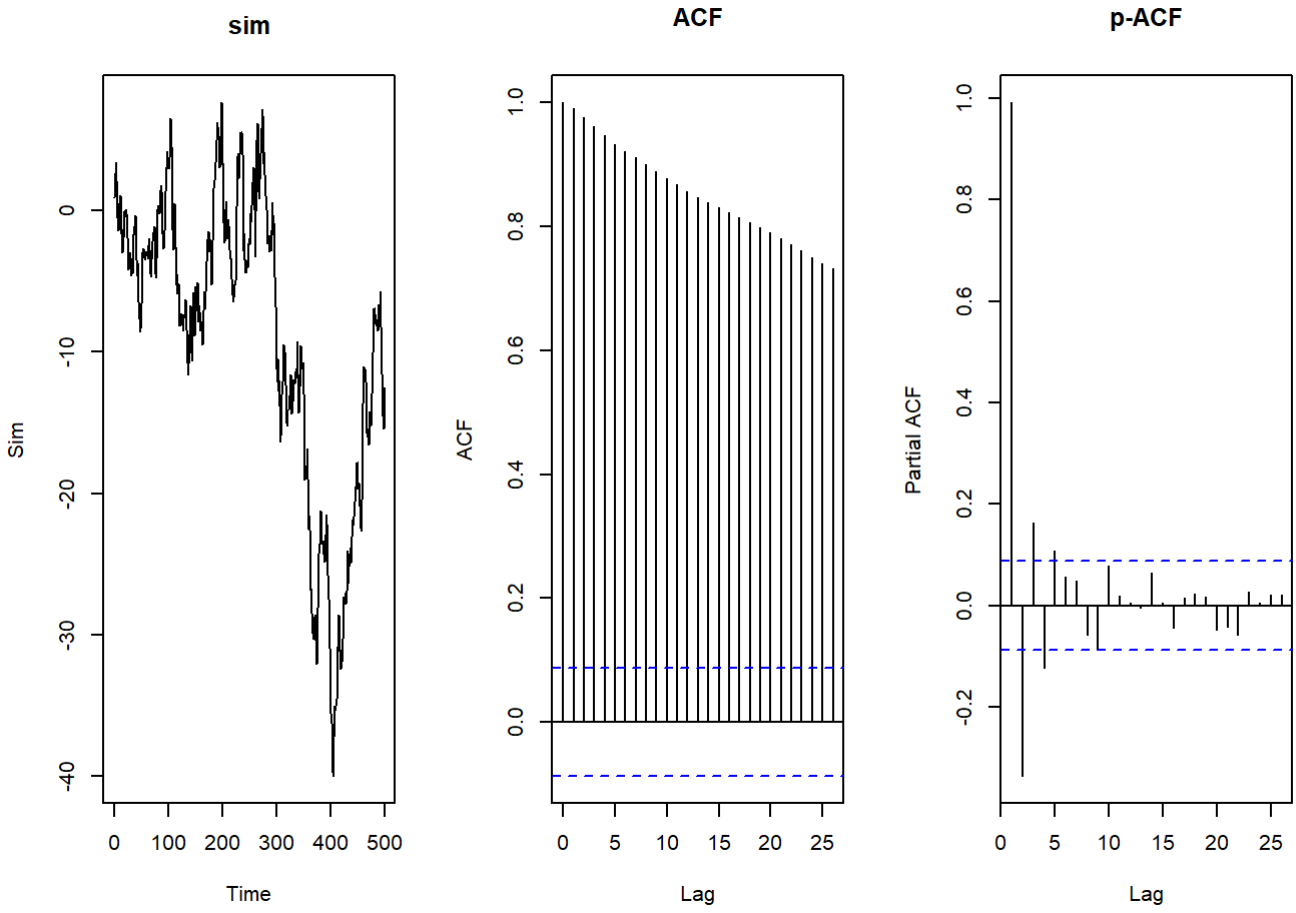


La doppia differenziazione può essere utile quando ci sono ancora tendenze o stagionalità rilevanti nella serie temporale differenziata una sola volta. Tuttavia, è importante notare che la differenziazione eccessiva può rendere la serie troppo volatile o causare la perdita di informazioni significative. Pertanto, la scelta del numero di differenziazioni dipende dalle caratteristiche specifiche della serie temporale e degli obiettivi analitici.

SIMULAZIONE ARIMA(0,1,1)

Il codice simula una serie temporale.

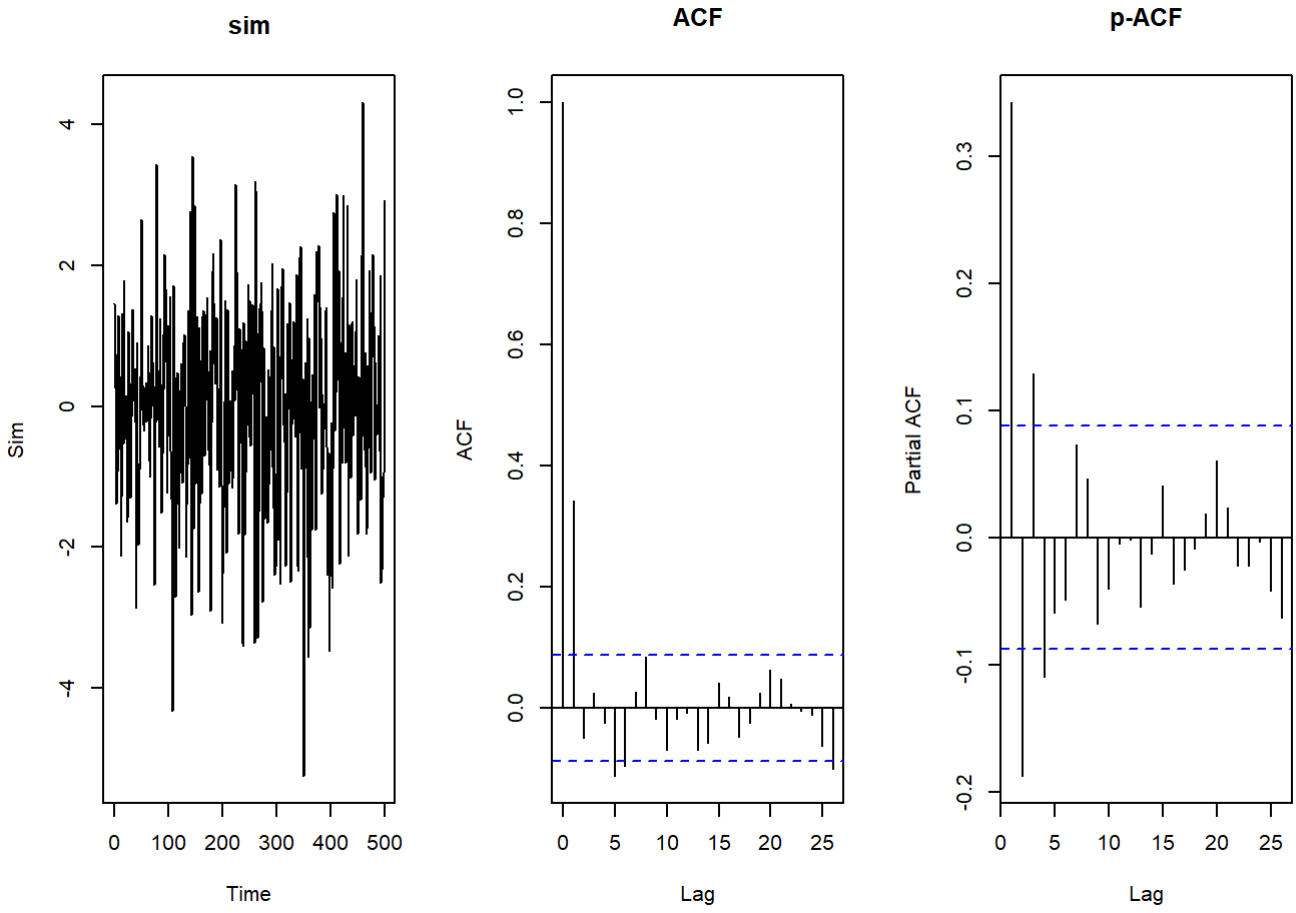
```r
set.seed(100)
x = c() # Serie temporale simulata
beta1 = 0.5 # Coefficiente autoregressivo associato alla precedente osservazione
sigma2 = 1.5 # Variabilità del rumore bianco gaussiano
w = rnorm(1000, 0, sigma2^0.5)

x [1:4] = 0 # Inizializza i primi quattro valori della serie temporale a zero
# Calcola le osservazioni successive della serie temporale x utilizzando un modello autoregre
ssivo di ordine 1 con il rumore bianco w. Ogni osservazione di x è la somma del valore corris
pondente di w, della precedente osservazione di x e del prodotto tra beta1 e la precedente os
servazione di w
for (i in 4:600){
  x[i] = x[i-1]+w[i]+beta1*w[i-1]
}

x = x[-c (1:100)] # Rimuove le prime 100 osservazioni per ridurre l'effetto dei valori inizia
li influenzati dall'inizializzazione a zero

par(mfrow=c(1,3))
plot(x, type ="l", main="sim", xlab="Time", ylab="Sim") # Plotta la serie temporale simulata
x
acf(x, main="ACF") # Plotta la funzione di autocorrelazione (ACF) della serie temporale
pacf(x, main="p-ACF") # Plotta la funzione di autocorrelazione parziale (p-ACF) della serie t
emporale
```



SIMULAZIONE ARIMA(0,1,1) – DIFFERENZE PRIME

Il codice plotta la simulazione ARIMA differenziata.

```r
set.seed(100)
x = c() # Serie temporale simulata
beta1 = 0.5 # Coefficiente autoregressivo associato alla precedente osservazione
sigma2 = 1.5 # Variabilità del rumore bianco gaussiano
w = rnorm(1000, 0, sigma2^0.5)

x [1:4] = 0 # Inizializza i primi quattro valori della serie temporale a zero
# Calcola le osservazioni successive della serie temporale x utilizzando un modello autoregre
ssivo di ordine 1 con il rumore bianco w. Ogni osservazione di x è la somma del valore corris
pondente di w, della precedente osservazione di x e del prodotto tra beta1 e la precedente os
servazione di w
for (i in 4:600){
  x[i] = x[i-1]+w[i]+beta1*w[i-1]
}

x = x[-c (1:100)] # Rimuove le prime 100 osservazioni per ridurre l'effetto dei valori inizia
li influenzati dall'inizializzazione a zero

par(mfrow=c(1,3))
plot(diff(x), type ="l", main="sim", xlab="Time", ylab="Sim") # Plotta la serie temporale sim
ulata x
acf(diff(x), main="ACF") # Plotta la funzione di autocorrelazione (ACF) della serie temporale
pacf(diff(x), main="p-ACF") # Plotta la funzione di autocorrelazione parziale (p-ACF) della s
erie temporale
```

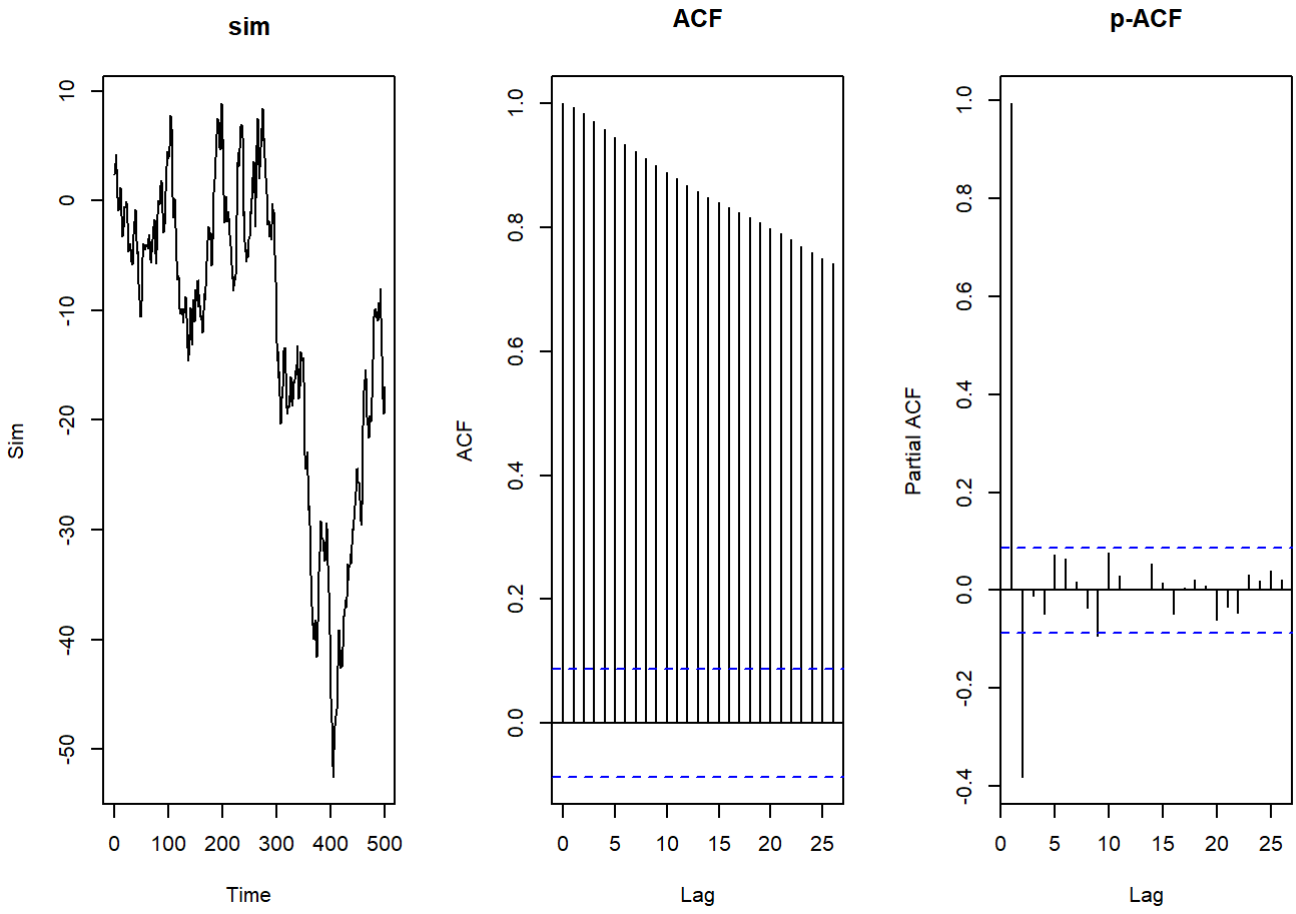


C

SIMULAZIONE ARIMA(1,1,0)

Il codice simula una serie temporale utilizzando un modello autoregressivo ARIMA(1,1,0) con un termine di errore gaussiano.

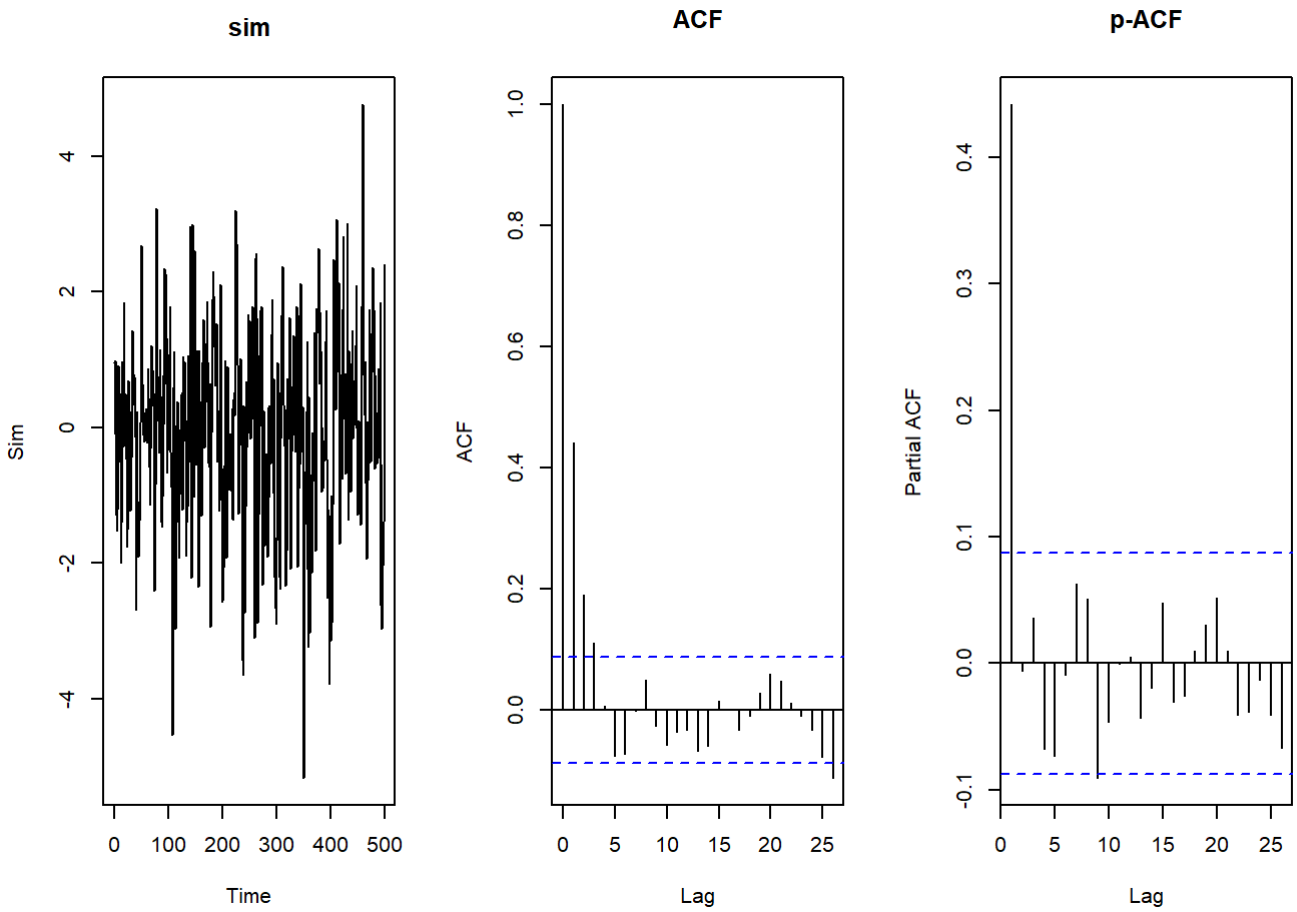
```
set.seed (100)
x = c() # Serie temporale simulata
alpha = 0.5 # Coefficiente autoregressivo associato alla precedente osservazione
sigma2 = 1.5 # Variabilità del rumore bianco gaussiano
w = rnorm(1000, 0, sigma2^0.5)
x [1:4] = 0
# Calcola le osservazioni successive della serie temporale x
for (i in 4:600){
  x[i] = alpha*x[i-1]+x[i-1]-alpha*x[i-2]+ w[i]
}
x = x[-c(1:100)] #  Rimuove le prime 100 osservazioni per ridurre l'effetto dei valori inizia
li influenzati dall'inizializzazione a zero
par(mfrow=c(1,3))
plot(x, type="l", main="sim", xlab="Time", ylab="Sim") # Plotta la serie temporale simulata x
acf(x, main="ACF") # Plotta la funzione di autocorrelazione (ACF) della serie temporale
pacf(x, main="p-ACF") # Plotta la funzione di autocorrelazione parziale (p-ACF) della serie t
emporale
```



SIMULAZIONE ARIMA(1,1,0) – DIFFERENZE PRIME

Il codice effettua una differenziazione della serie temporale x prima di plottarla e analizzarla per autocorrelazione e autocorrelazione parziale.

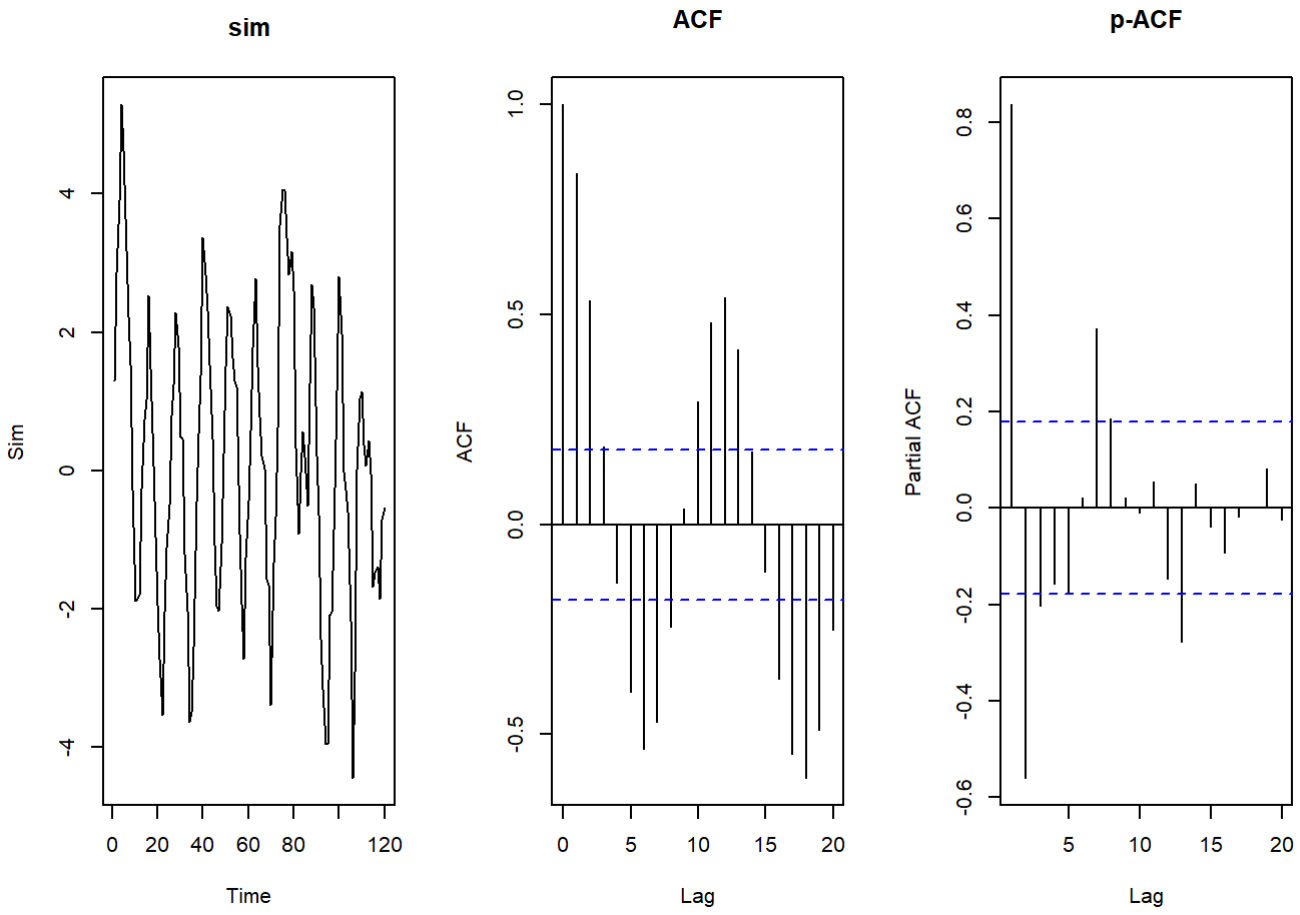
```r
set.seed (100)
x = c() # Serie temporale simulata
alpha = 0.5 # Coefficiente autoregressivo associato alla precedente osservazione
sigma2 = 1.5 # Variabilità del rumore bianco gaussiano
w = rnorm(1000, 0, sigma2^0.5)
x [1:4] = 0
# Calcola le osservazioni successive della serie temporale x
for (i in 4:600){
  x[i] = alpha*x[i-1]+x[i-1]-alpha*x[i-2]+ w[i]
}
x = x[-c(1:100)] #  Rimuove le prime 100 osservazioni per ridurre l'effetto dei valori iniziali influenzati dall'inizializzazione a zero
par(mfrow=c(1,3))
plot(diff(x), type="l", main="sim", xlab="Time", ylab="Sim") # Plotta la serie temporale differenziata simulata x
acf(diff(x), main="ACF") # Plotta la funzione di autocorrelazione (ACF) della serie temporale differenziata
pacf(diff(x), main="p-ACF") # Plotta la funzione di autocorrelazione parziale (p-ACF) della serie temporale differenziata
```



SIMULAZIONE DI UN AR(1) CON EFFETTO STAGIONALE DI LUNGHEZZA 12

Il codice simula una serie temporale y con una componente autoregressiva e un termine di errore.

```
x = c() # Componente autoregressiva
alpha = 0.8 # Coefficiente autoregressivo
sigma2 = 0.5 # Varianza del termine di errore
x[1] = rnorm(1, 0, (sigma2/(1-alpha^2))^0.5)

# Crea un vettore beta con un pattern ciclico di valori. Questo sarà il vettore di "offset" c
he verrà sommato alla componente autoregressiva per formare la serie temporale finale y
beta = rep(c(0,1,2,3,2,1,0,-1,-2,-3,-2,-1), times=10)

# Calcola i successivi valori della componente autoregressiva x utilizzando un modello AR(1)
con il termine di errore estratto da una distribuzione normale
for(i in 2:120){
  x[i] = rnorm(1, alpha*x[i-1], sigma2^0.5)
}

# Calcola la serie temporale finale y sommando la componente autoregressiva x al vettore di
"offset" beta.
y = x + beta

par(mfrow=c(1,3))
plot(y, type="l", main="sim", xlab="Time", ylab="Sim") # Plotta la serie temporale simulata y
acf(y, main="ACF") # Plotta la funzione di autocorrelazione (ACF) della serie temporale
pacf(y, main="p-ACF") # Plotta la funzione di autocorrelazione parziale (p-ACF) della serie t
emporale
```



In sintesi, il codice simula una serie temporale y che ha una componente autoregressiva AR(1) con il termine di errore, e viene aggiunto un vettore ciclico beta per creare una struttura più complessa. La varianza totale della serie è controllata da sigma2.

ANALISI DI UNA SERIE STORICA

```
library(astsa)
library(forecast)
```

```
## Warning: il pacchetto 'forecast' è stato creato con R versione 4.2.3
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
##
## Caricamento pacchetto: 'forecast'
```

```
## Il seguente oggetto è mascherato da 'package:astsa':
##
##     gas
```

```
library("TTR")
```

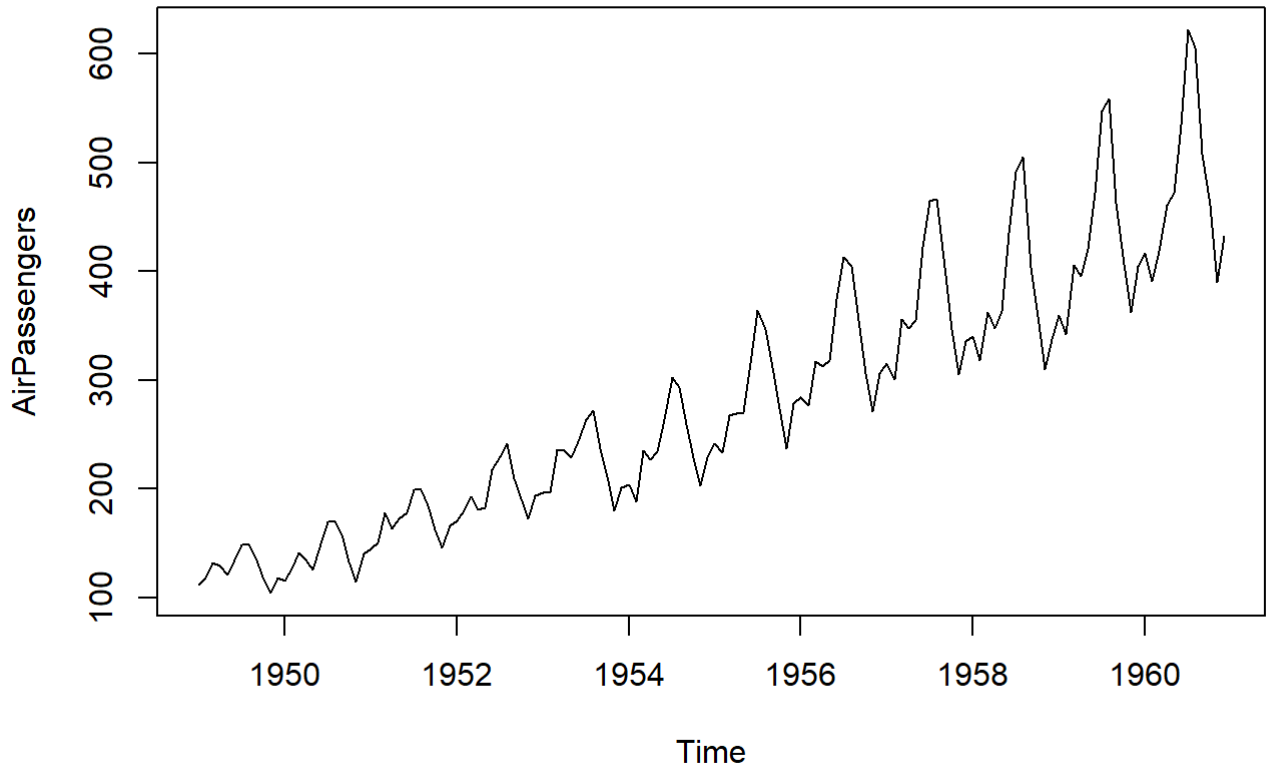
```
## Warning: il pacchetto 'TTR' è stato creato con R versione 4.2.3
```

Il codice utilizza il dataset predefinito AirPassengers per visualizzare un riepilogo statistico e alcuni grafici relativi al numero mensile di passeggeri di compagnie aeree internazionali, registrato da gennaio 1949 a dicembre 1960

```
# Carica il dataset predefinito AirPassengers nel tuo ambiente R. Questo dataset contiene il numero mensile di passeggeri di compagnie aeree internazionali
data("AirPassengers")

# Fornisce un riepilogo statistico del dataset, che include il conteggio, la media, la median a, i quartili e altri parametri statistici per la variabile AirPassengers
summary(AirPassengers)
```

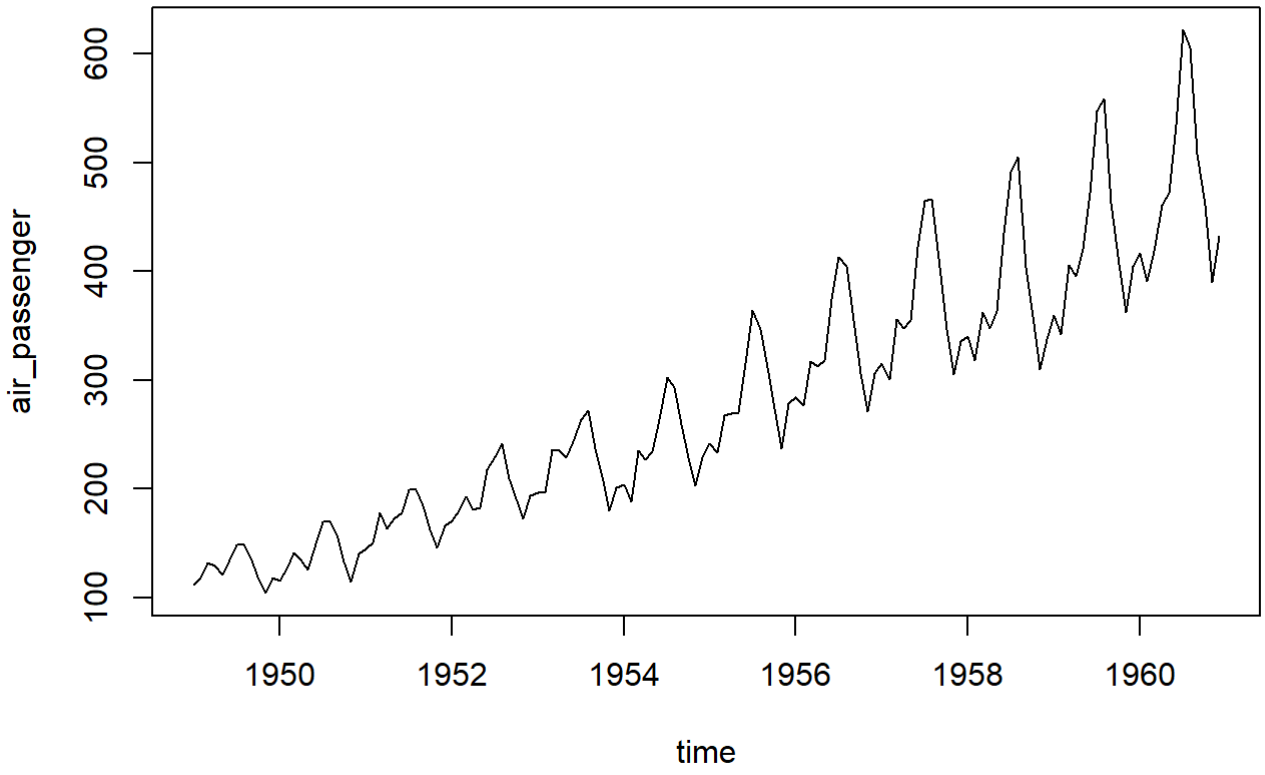
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   104.0   180.0   265.5   280.3   360.5   622.0
```

```
# Plotta il numero mensile di passeggeri nel tempo. Il grafico mostra una tendenza di crescit a generale con alcune evidenti stagionalità annue
plot(AirPassengers)
```



Il codice ha creato un grafico della serie temporale dei passeggeri aerei internazionali, con il tempo espresso in anni e mesi.

```
# Trasforma la matrice AirPassengers in un vettore con la funzione t() (trasposizione) e c() (concatenazione). In questo modo, air_passenger contiene tutti i dati mensili di passeggeri aerei internazionali in un unico vettore
air_passenger = c(t(AirPassengers))

# Crea un vettore di sequenza time che rappresenta il tempo da gennaio 1949 a dicembre 1960 con incrementi mensili
time = seq(1949, 1961, by = 1/12)
time = time[-length(time)] # Rimuove l'ultimo elemento da time, che era stato aggiunto in eccesso nel caso in cui il vettore rappresentasse il tempo fino a gennaio 1961

# Plotta la serie temporale dei passeggeri aerei internazionali nel tempo. L'asse x rappresenta l'anno e il mese, mentre l'asse y rappresenta il numero di passeggeri
plot(time,air_passenger, type="l" )
```



Questo grafico fornisce una visualizzazione della serie temporale dei passeggeri aerei internazionali nel periodo considerato, mostrando la tendenza generale e la stagionalità dei dati.

ELIMINAZIONE DEL TREND

Immaginimao che X_t sia la serie. Io voglio lavorare con

$$Y_t = X_t - E(X_t)$$

Y_t è simile al residuo di una regressione. Per togliere il trend possiamo usare un modello lineare, del tipo

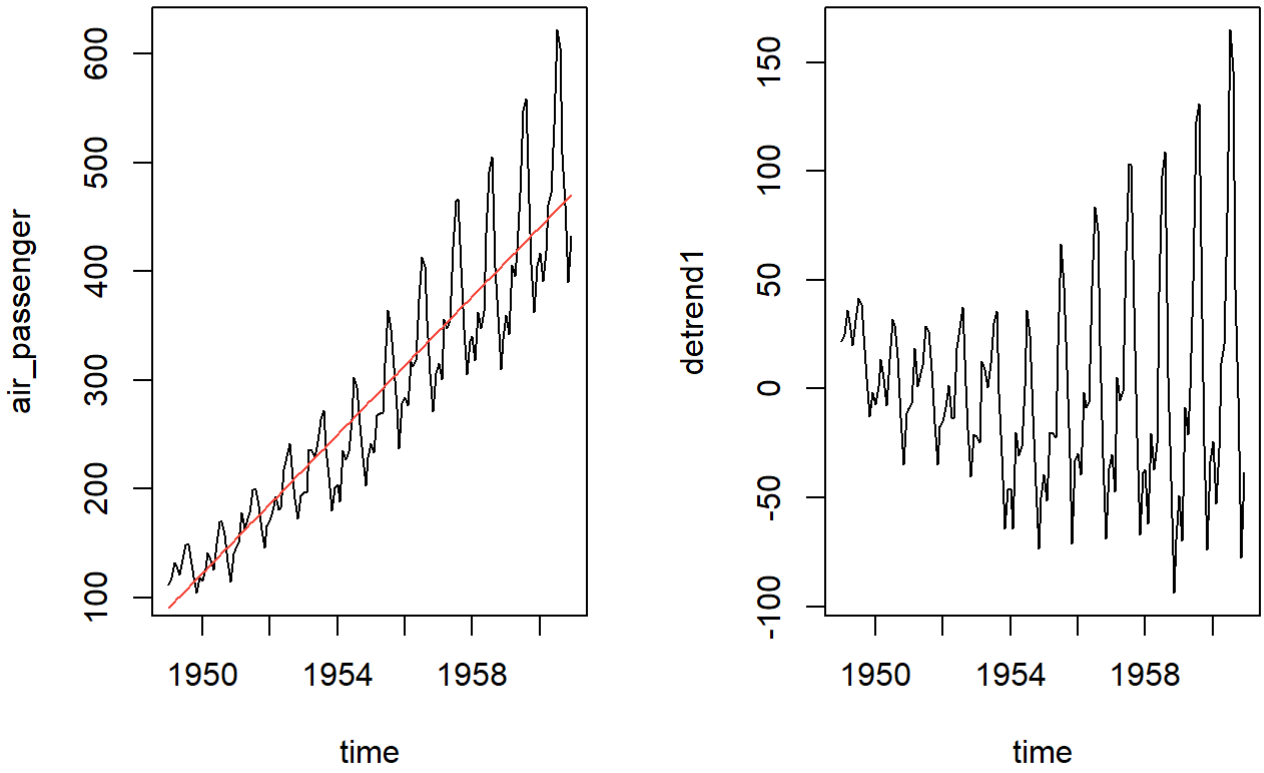
$$X_t = \beta_0 + \beta_1 z_{t,1} + \beta_2 z_{t,2} + \cdots + \beta_p z_{t,p}$$

Testiamo trend in funzione del tempo.

1. **Trend lineare**

Il codice esegue una regressione lineare della serie temporale air_passenger rispetto al tempo (time). Successivamente, viene plottato il grafico originale e il grafico dei residui, fornendo una visualizzazione della componente trend della serie temporale.

```
# Esegue una regressione lineare della serie temporale air_passenger rispetto al tempo time. Il modello di regressione cerca di spiegare la variabilità nella serie temporale attraverso una relazione lineare con il tempo. I coefficienti della regressione sono memorizzati in reg$coefficients
reg = lm(air_passenger ~ time)
# Estrae i residui della regressione lineare e li memorizza nella variabile detrend1. I residui rappresentano la parte della serie temporale che non è spiegata dalla componente lineare del tempo
detrend1 = reg$residuals

par(mfrow=c(1,2))
plot(time, air_passenger, type="l") #  Plotta la serie temporale originale dei passeggeri aerei internazionali nel tempo
lines(time, reg$coefficients[1]+ reg$coefficients[2]*time, col=2) # Sovrapponi al grafico originale la retta rossa di regressione lineare ottenuta dal modello.
plot(time, detrend1, type="l") # Plotta i residui della regressione (detrended series), mostrando la parte della serie temporale che è stata rimossa attraverso la regressione lineare.
```

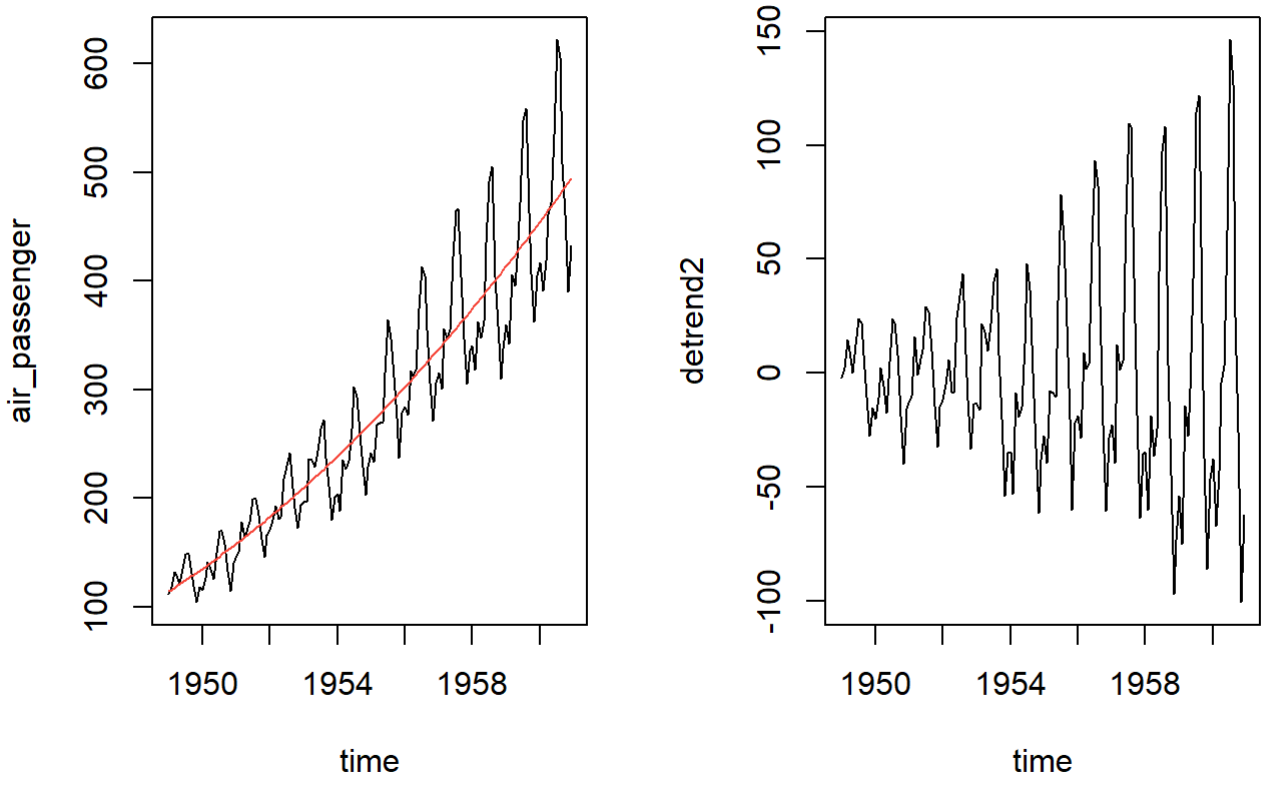


In sintesi, il codice mostra due grafici. Il primo mostra la serie temporale originale dei passeggeri aerei internazionali con la retta di regressione lineare sovrapposta, mentre il secondo mostra la componente detrended della serie temporale, che rappresenta la parte della variabilità non spiegata dalla componente lineare del tempo.

2. **Trend quandratico**

Il codice esegue una regressione quadratica della serie temporale air_passenger rispetto al tempo (time). Successivamente, viene plottato il grafico originale e il grafico dei residui della regressione quadratica, fornendo una visualizzazione della componente trend della serie temporale.

```
# Esegue una regressione quadratica della serie temporale air_passenger rispetto al tempo time. Il termine I(time^2) è utilizzato per rappresentare il termine quadratico nella regression e. I coefficienti della regressione sono memorizzati in reg$coefficients
reg = lm(air_passenger ~ time + I(time^2))

# Estrae i residui della regressione quadratica e li memorizza nella variabile detrend2. I re sidui rappresentano la parte della serie temporale che non è spiegata dalla componente quadra tica del tempo.
detrend2 = reg$residuals

par(mfrow=c(1,2))
plot(time,air_passenger, type="l") #  Plotta la serie temporale originale dei passeggeri aere i internazionali nel tempo
lines(time, reg$coefficients[1]+ reg$coefficients[2]*time + reg$coefficients[3]*time^2, col= 2) #  Sovrapponi al grafico originale la curva di regressione quadratica ottenuta dal modello in rosso
plot(time, detrend2, type="l") # Plotta i residui della regressione quadratica (detrended ser ies), mostrando la parte della serie temporale che è stata rimossa attraverso la regressione quadratica
```

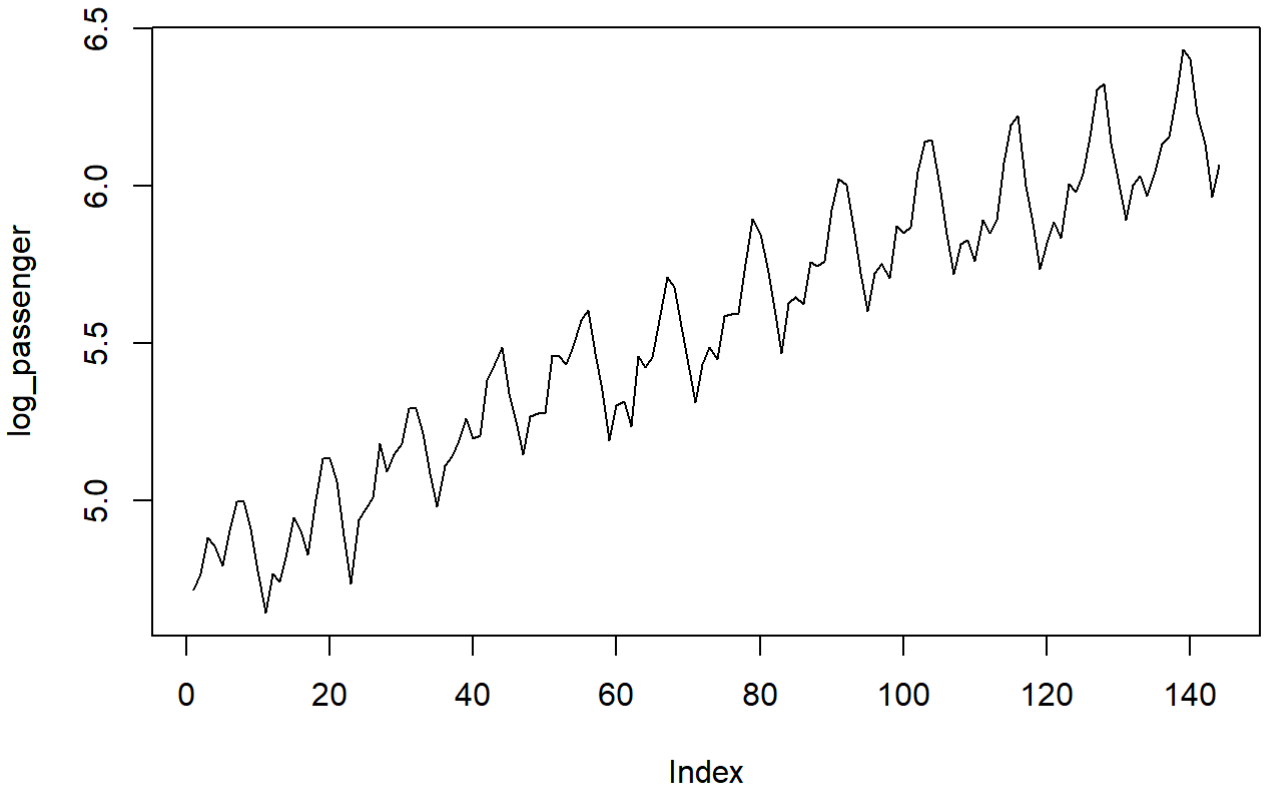


In sintesi, il codice mostra due grafici. Il primo mostra la serie temporale originale dei passeggeri aerei internazionali con la curva di regressione quadratica sovrapposta, mentre il secondo mostra la componente detrended della serie temporale, che rappresenta la parte della variabilità non spiegata dalla componente quadratica del tempo.

Continauiamo ad avere il problema della varianza, e per questo motivo detrendizziamo utilizzando la serie in scala logaritmica.

Il codice calcola il logaritmo naturale (logaritmo in base e) della serie temporale air_passenger e plotta il risultato.

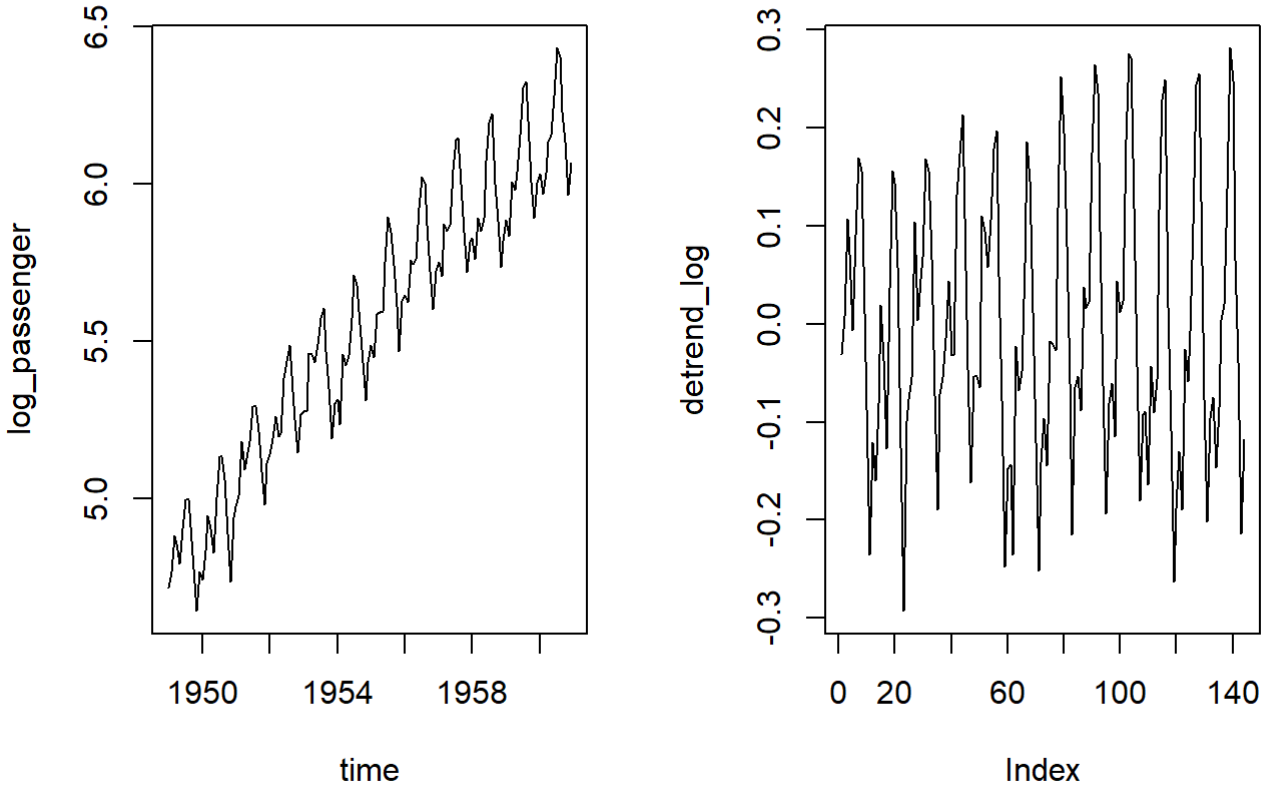
```
log_passenger = log(air_passenger) #  La trasformazione logaritmica può essere utile per stab
ilizzare la varianza e rendere la serie temporale più interpretabile
plot(log_passenger, type="l") # Plotta la serie temporale trasformata attraverso il logaritmo
naturale
```



Questo grafico mostra la serie temporale dei logaritmi naturali del numero mensile di passeggeri aerei internazionali. La trasformazione logaritmica è spesso utilizzata per stabilizzare la varianza e rendere più chiare le tendenze relative a tassi di crescita percentuale.

Il codice esegue una regressione quadratica del logaritmo naturale della serie temporale air_passenger rispetto al tempo (time). Successivamente, vengono plottati il grafico originale e il grafico dei residui della regressione quadratica trasformata.

```
reg = lm(log_passenger ~ time + I(time^2)) # Esegue una regressione quadratica del logaritmo naturale
detrend_log = reg$residuals # Estrae i residui della regressione quadratica trasformata e li memorizza nella variabile detrend_log. I residui rappresentano la parte della serie temporale che non è spiegata dalla componente quadratica del tempo.
par(mfrow=c(1,2))
plot(time,log_passenger, type="l") #  Plotta la serie temporale del logaritmo naturale dei passeggeri aerei internazionali nel tempo
plot(detrend_log , type="l") # Plotta i residui della regressione quadratica trasformata (detrended series), mostrando la parte della serie temporale che è stata rimossa attraverso la regressione quadratica
```



In sintesi, il codice mostra due grafici. Il primo mostra la serie temporale del logaritmo naturale dei passeggeri aerei internazionali con la curva di regressione quadratica sovrapposta, mentre il secondo mostra la componente detrended della serie temporale trasformata, che rappresenta la parte della variabilità non spiegata dalla componente quadratica del tempo. La trasformazione logaritmica può essere utile per stabilizzare la varianza e semplificare l'interpretazione delle tendenze in termini di tassi di crescita percentuale.

MODELLI ARIMA

Adesso vogliamo modellizzare utilizzando un modello ARIMA stagionale (detto anche SARIMA)

$$\Theta_P(B^s)\theta_p(B)(1-B^s)^D(1-B)^d y_t = \Phi_Q(B^s)\phi_q(B)w_t$$

dove in questo caso y_t è il residuo (cioè, quello che vogliamo modellizzare) Il modello si definisce come $\text{ARIMA}(p,d,q)(P,D,Q)_s$

```
y = detrend_log # Residui della regressione quadratica trasformata
```

Facciamo prima un $AR(1)$.

Il codice è progettato per adattare un modello SARIMA (Seasonal AutoRegressive Integrated Moving Average) con i parametri specificati.

```
# Definisci i parametri del modello SARIMA
p = 1
d = 0
q = 0
P = 0
D = 0
Q = 0
s = 12

model_ar1 = sarima(y,p,d,q,P,D,Q,s, details=F) # Applico il modello
model_ar1$ttable # # Visualizza la tabella dei coefficienti del modello
```

```
##       Estimate     SE t.value p.value
## ar1     0.6842 0.0601 11.3788  0.0000
## xmean  -0.0022 0.0253 -0.0862  0.9315
```

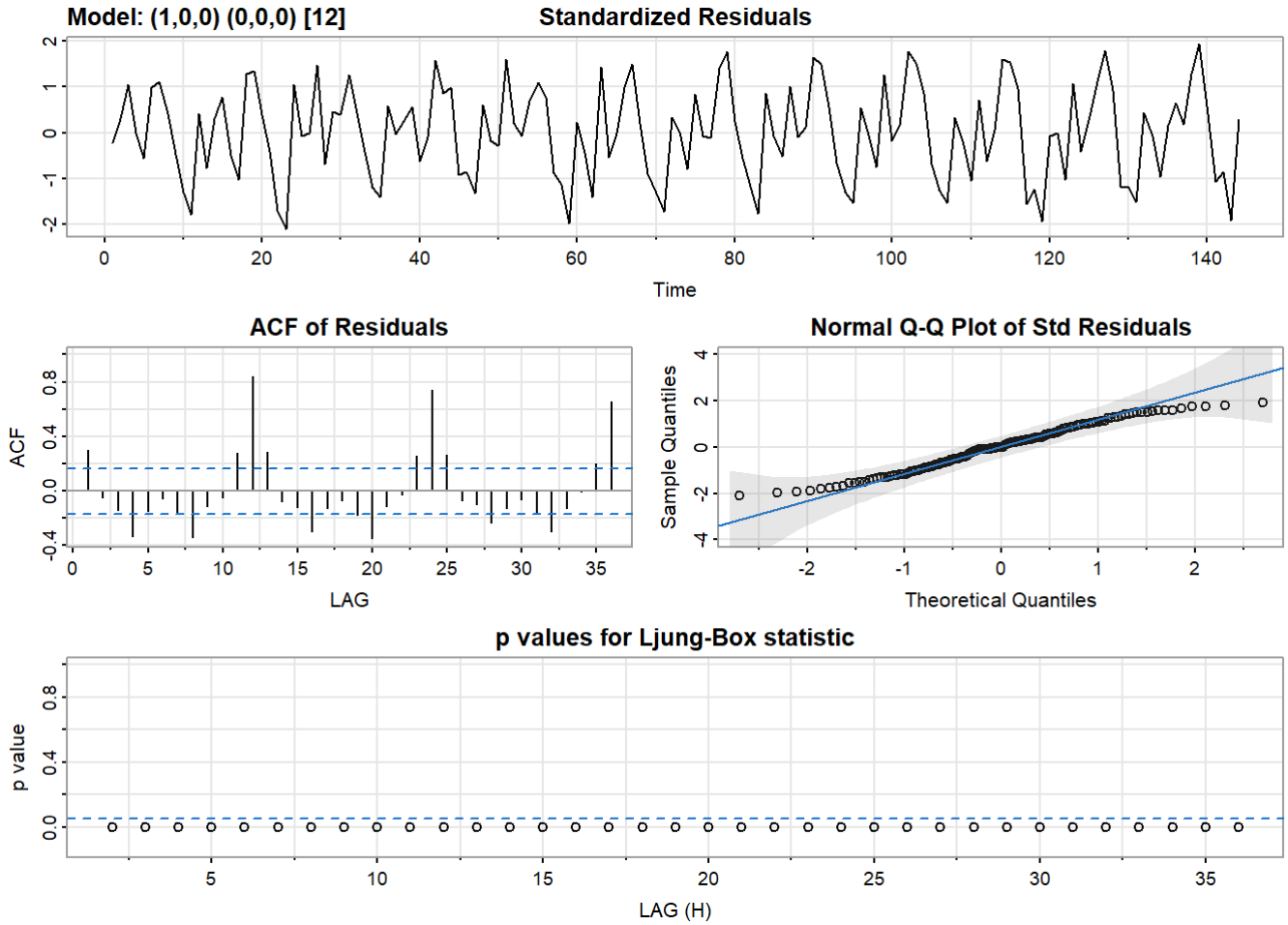
Se non specificate diversamente, il modello stimato è

$$\Theta_P(B^s)\theta_p(B)(1-B^s)^D(1-B)^d(y_t-\mu)=\Phi_Q(B^s)\phi_q(B)w_t$$

dove μ non dipende dal tempo

```
model_ar1 = sarima(y,p,d,q,P,D,Q,s, details=T)
```

```
## initial  value -2.007723
## iter   2 value -2.326873
## iter   3 value -2.326877
## iter   4 value -2.326887
## iter   5 value -2.326889
## iter   6 value -2.326892
## iter   7 value -2.326892
## iter   8 value -2.326892
## iter   8 value -2.326892
## iter   8 value -2.326892
## final  value -2.326892
## converged
## initial  value -2.327971
## iter   2 value -2.327988
## iter   3 value -2.327994
## iter   4 value -2.327995
## iter   5 value -2.327997
## iter   5 value -2.327997
## iter   5 value -2.327997
## final  value -2.327997
## converged
```



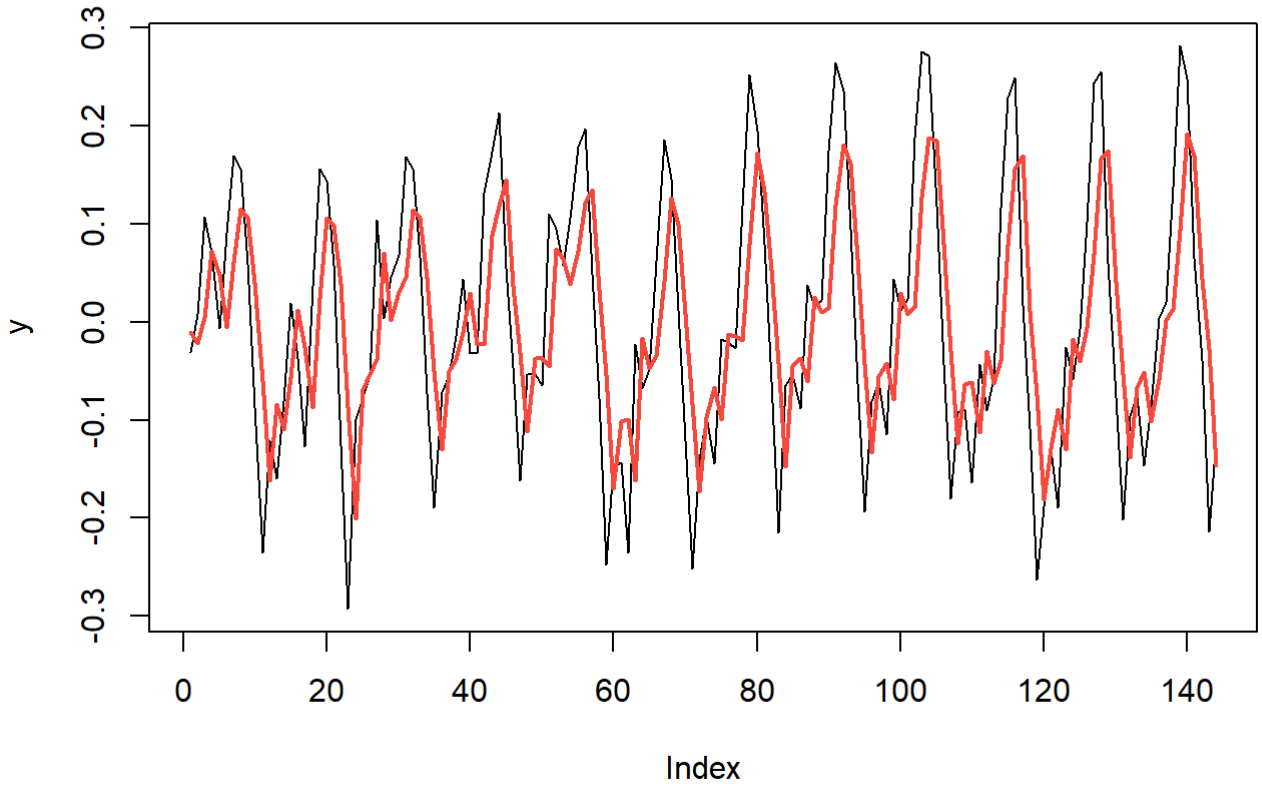
Anche se il modello non è perfetto, è comuqnue interessante vedere i suoi residui ($\hat{w}_t$) o

$$\hat{y}_t = y_t - \hat{w}_t$$

.

Il codice vuole sovrapporre al grafico della serie temporale y la curva ottenuta sottraendo i residui del modello SARIMA.

```
# Calcola la curva fit_ar1 sottraendo i residui del modello SARIMA adattato (model_ar1$fit) d
alla serie temporale originale y. Questo è un modo per ottenere la parte del modello che è st
ata "spiegata" dalla SARIMA, ovvero la serie temporale privata della componente stagionale e
di tendenza identificata dal modello
fit_ar1 = y - residuals(model_ar1$fit)
plot(y, type="l") # Plotta la serie temporale originale y
lines(fit_ar1, col=2, lwd=2)
```

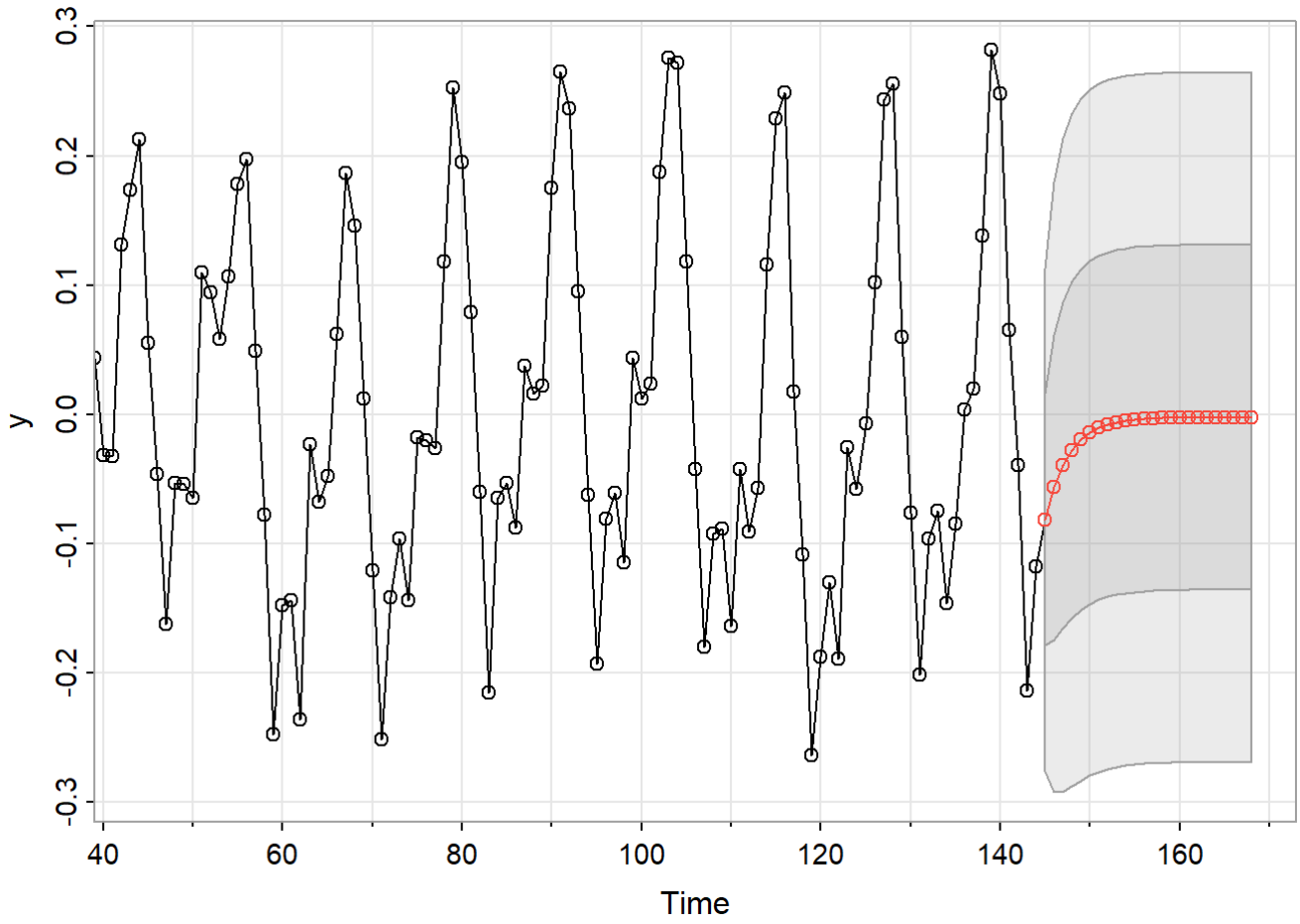


Questo grafico mostra la serie temporale originale e la parte “spiegata” dal modello SARIMA (senza la componente stagionale e di tendenza), consentendo di visualizzare come il modello si adatta ai dati.

Con sarima possiamo fare previsione facilmente.

Il codice utilizza la funzione sarima.for per effettuare previsioni future sulla serie temporale y utilizzando un modello SARIMA

```
# 24 è il numero di periodi di previsione futura, cioè 24 mesi
prev_ar1 = sarima.for(y,24,p,d,q,P,D,Q,s)
```



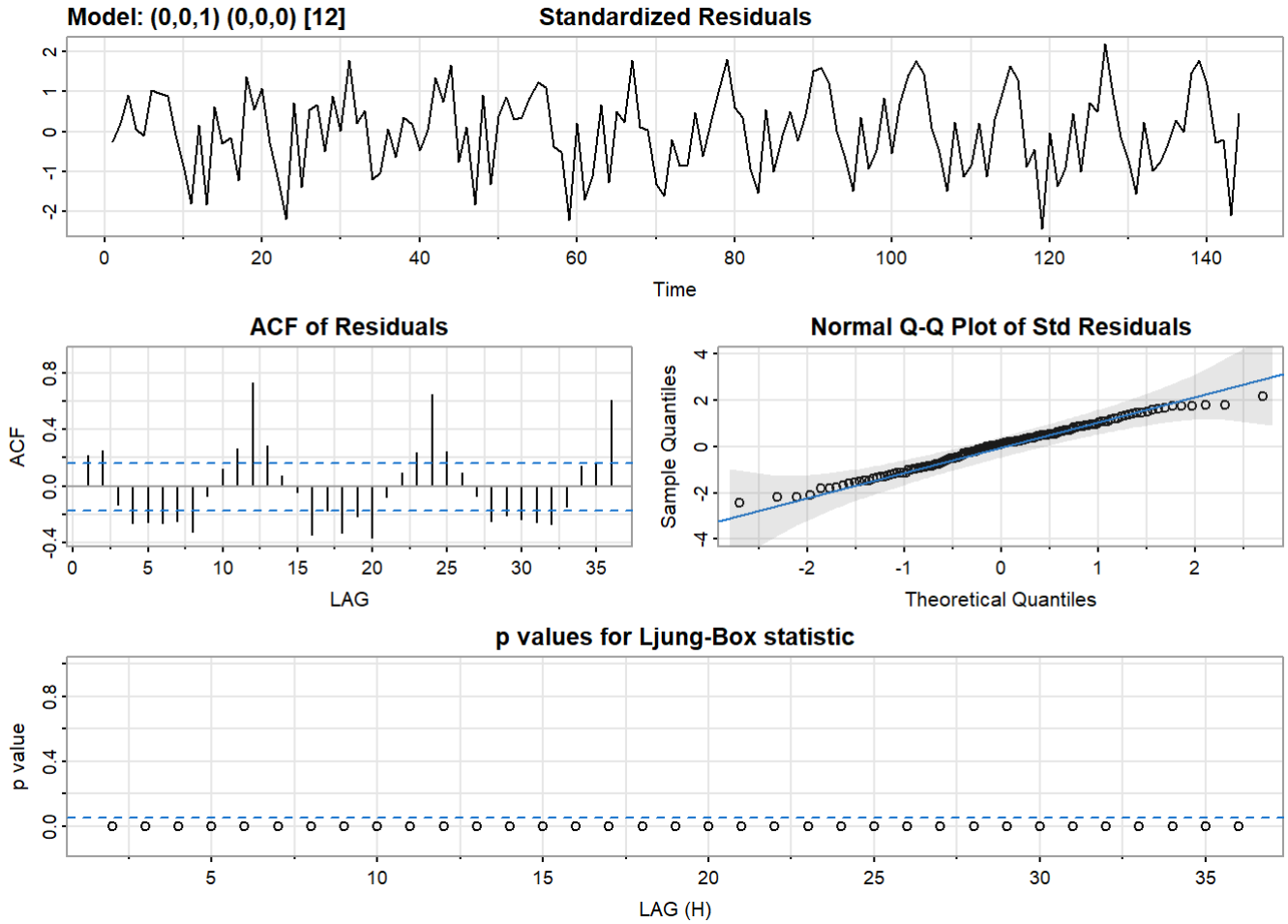
Il risultato della funzione sarima.for (prev_ar1) contine le previsioni future per i prossimi 24 periodi, basate sul modello SARIMA adattato ai dati storici. Puoi esplorare il contenuto di prev_ar1 per ottenere le previsioni e altre informazioni.

Proviamo un modello MA(1).

Il codice adatta un modello SARIMA con un termine di media mobile di ordine 1 (MA(1)) sulla serie temporale y.

```
# Questi sono i parametri specificati per il modello SARIMA. In particolare, q è impostato su 1, indicando che si sta adattando un termine di media mobile (MA) di ordine 1
p = 0
d = 0
q = 1
P = 0
D = 0
Q = 0
s = 12

model_ma1 = sarima(y,p,d,q,P,D,Q,s)
```

```
## initial  value -2.011020
## iter   2 value -2.340957
## iter   3 value -2.354440
## iter   4 value -2.355055
## iter   5 value -2.358139
## iter   6 value -2.358450
## iter   7 value -2.358472
## iter   8 value -2.358476
## iter   9 value -2.358476
## iter   9 value -2.358476
## iter   9 value -2.358476
## final  value -2.358476
## converged
## initial  value -2.355355
## iter   2 value -2.355355
## iter   3 value -2.355364
## iter   4 value -2.355364
## iter   4 value -2.355364
## iter   4 value -2.355364
## final  value -2.355364
## converged
```



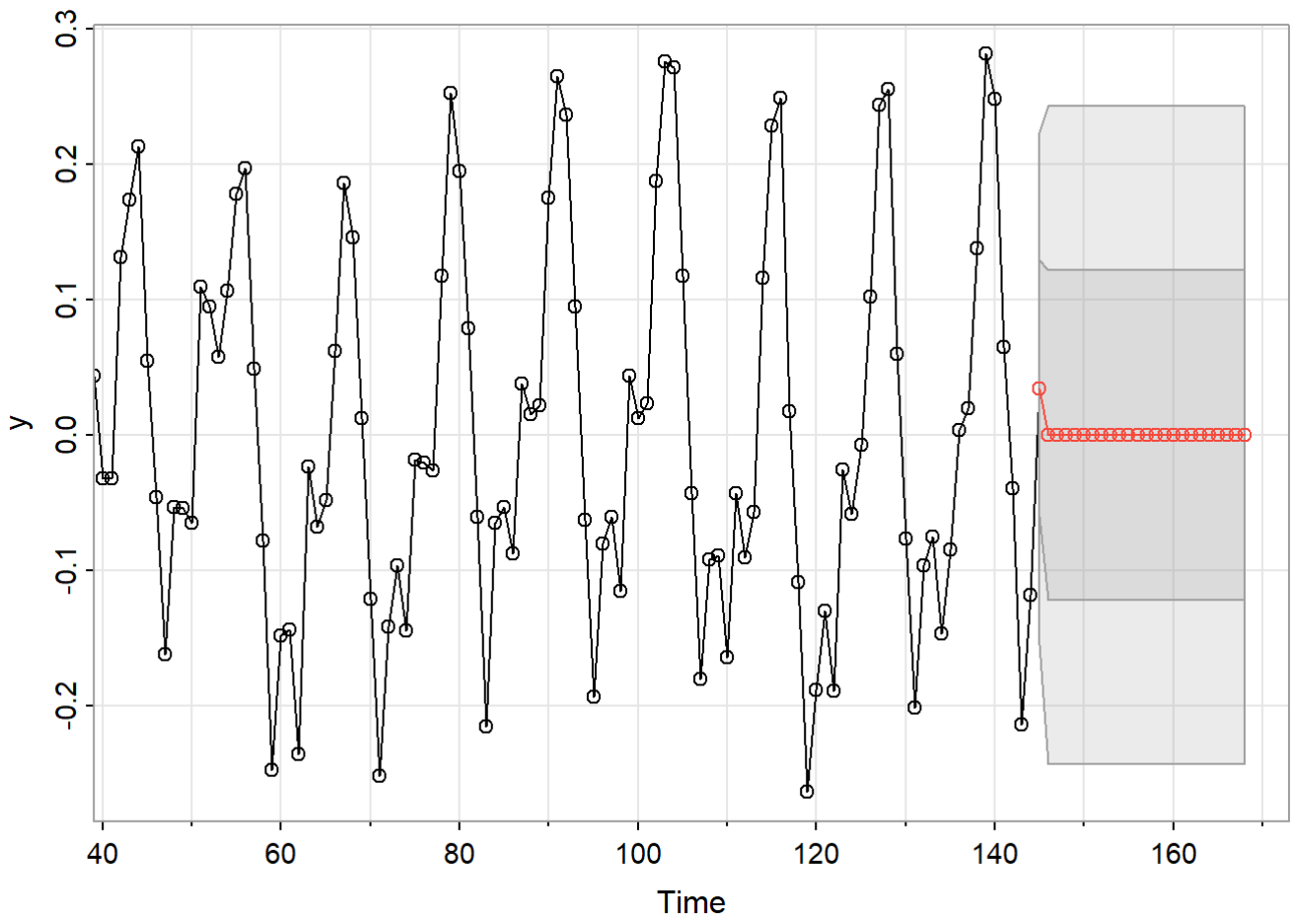

```
model_ma1$ttable #  La tabella dei coefficienti del modello adattato
```

```
##       Estimate     SE t.value p.value
## ma1     0.8099 0.0478 16.9352  0.0000
## xmean   0.0001 0.0142  0.0055  0.9956
```

Vediamo la predizione.

Il codice utilizza la funzione sarima.for per effettuare previsioni future sulla serie temporale y utilizzando un modello SARIMA con un termine di media mobile di ordine 1 (MA(1)).

```
# 24: Specifica il numero di periodi di previsione futura (nel tuo caso, 24 periodi, che potr
ebbero rappresentare 24 mesi, ad esempio)
prev_ma1 = sarima.for(y,24,p,d,q,P,D,Q,s)
```



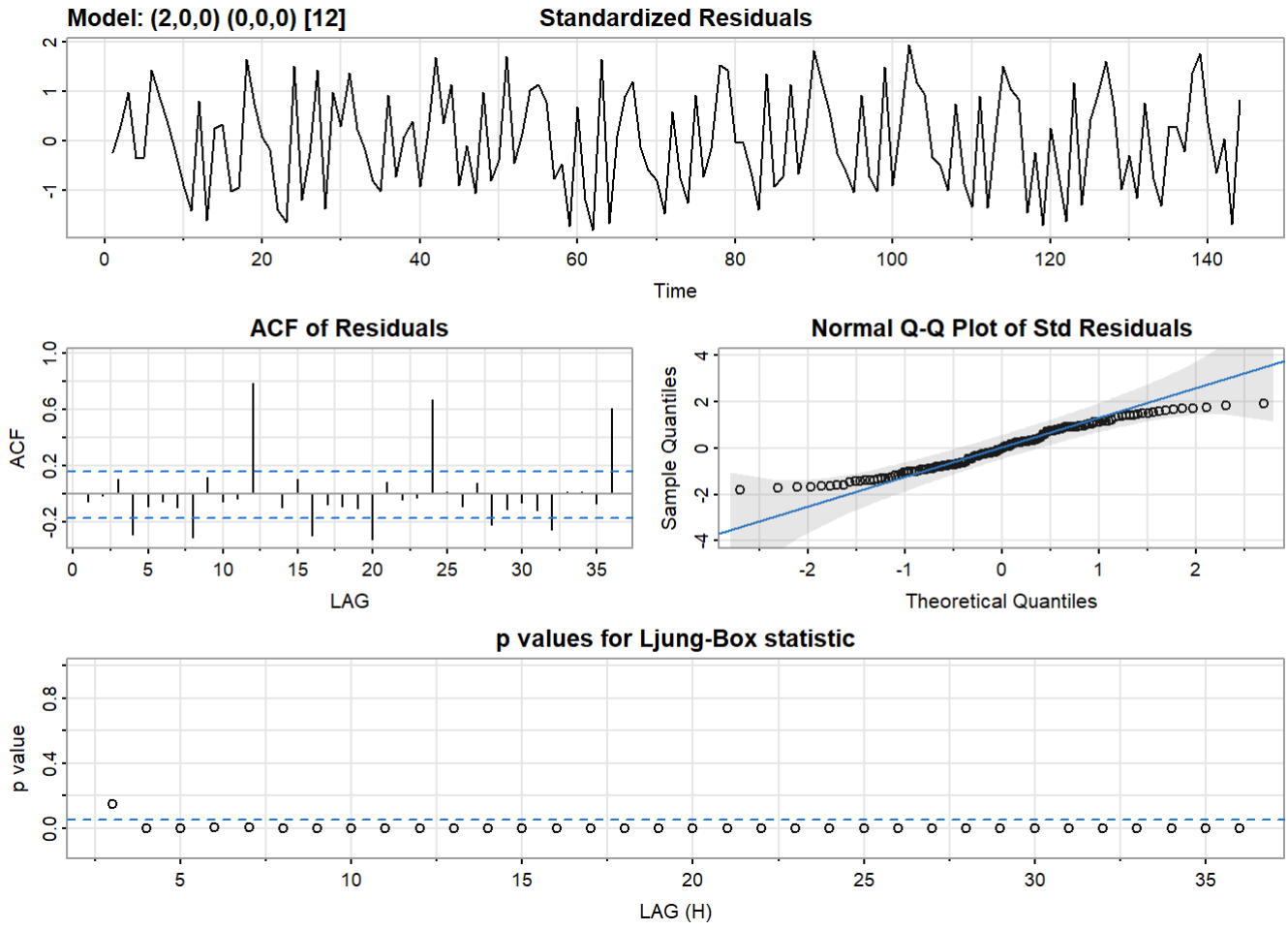
Il risultato della funzione sarima.for (prev_ma1) contiene le previsioni future per i prossimi 24 periodi, basate sul modello SARIMA con un termine di media mobile MA(1) adattato ai dati storici.

Facciamo adesso diversi AR(q)

Vengono adattati due modelli AR(2) e AR(3) sulla stessa serie temporale y.

```
# Primo Modello AR(2)
p = 2 # Specifica l'ordine di autoregressione (AR) pari a 2
d = 0 # Specifica l'ordine di differenziazione (d) pari a 0
q = 0 # Specifica l'ordine di media mobile (MA) pari a 0
# Specificano gli ordini stagionali
P = 0
D = 0
Q = 0
s = 12

model_ar2 = sarima(y,p,d,q,P,D,Q,s)
```

```
## initial  value -2.004227
## iter   2 value -2.188394
## iter   3 value -2.362687
## iter   4 value -2.384549
## iter   5 value -2.424361
## iter   6 value -2.424506
## iter   7 value -2.424515
## iter   8 value -2.424515
## iter   8 value -2.424515
## final  value -2.424515
## converged
## initial  value -2.427370
## iter   2 value -2.427396
## iter   3 value -2.427398
## iter   4 value -2.427398
## iter   5 value -2.427398
## iter   5 value -2.427398
## iter   5 value -2.427398
## final  value -2.427398
## converged
```

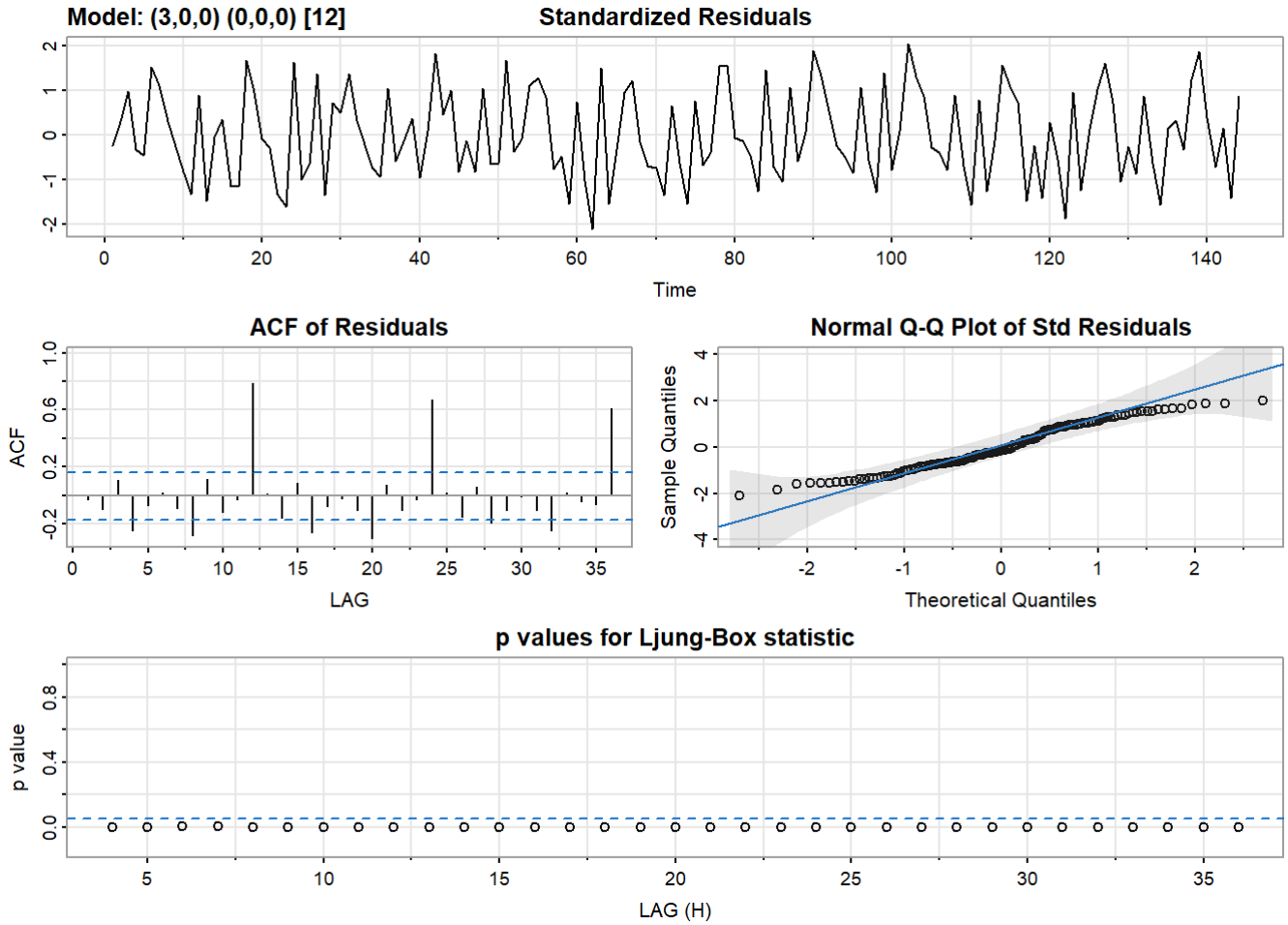



```
model_ar2$ttable
```

```
##       Estimate     SE t.value p.value
## ar1     0.9752 0.0750 13.0044  0.0000
## ar2    -0.4255 0.0754 -5.6464  0.0000
## xmean   0.0001 0.0162  0.0052  0.9959
```

```
# Secondo Modello AR(3)
p = 3 # Specifica l'ordine di autoregressione (AR) pari a 3
d = 0 # Specifica l'ordine di differenziazione (d) pari a 0
q = 0 # Specifica l'ordine di media mobile (MA) pari a 0
# Specificano gli ordini stagionali
P = 0
D = 0
Q = 0
s = 12

model_ar3 = sarima(y,p,d,q,P,D,Q,s)
```

```
## initial  value -2.002916
## iter   2 value -2.257135
## iter   3 value -2.336517
## iter   4 value -2.407486
## iter   5 value -2.423018
## iter   6 value -2.432759
## iter   7 value -2.435653
## iter   8 value -2.435902
## iter   9 value -2.435904
## iter   9 value -2.435904
## iter   9 value -2.435904
## final  value -2.435904
## converged
## initial  value -2.438720
## iter   2 value -2.438744
## iter   3 value -2.438751
## iter   4 value -2.438752
## iter   5 value -2.438752
## iter   5 value -2.438752
## iter   5 value -2.438752
## final  value -2.438752
## converged
```



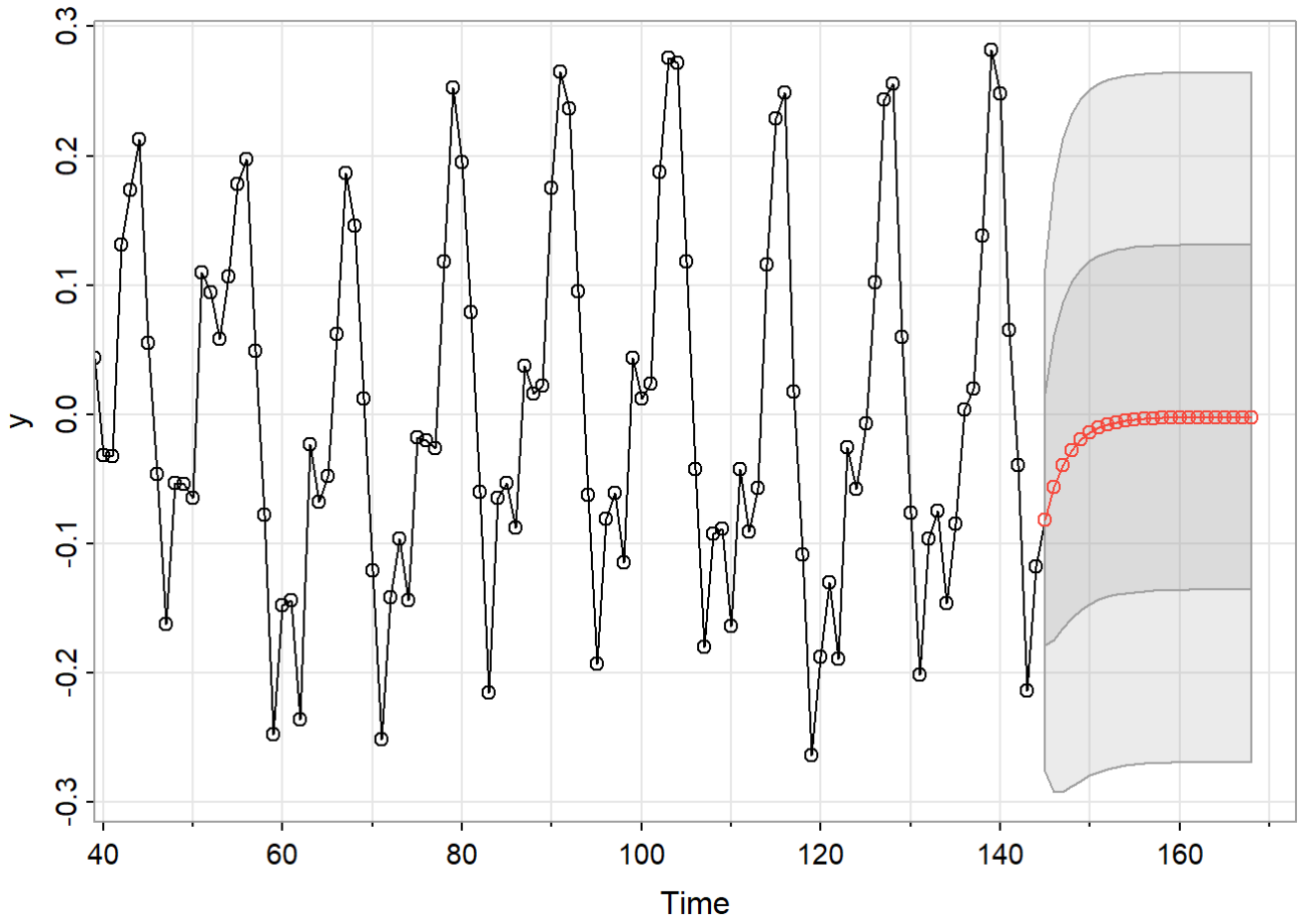

```
model_ar3$ttable
```

```
##       Estimate     SE t.value p.value
## ar1     0.9103 0.0822 11.0744  0.0000
## ar2    -0.2763 0.1108 -2.4928  0.0138
## ar3    -0.1512 0.0831 -1.8194  0.0710
## xmean   0.0001 0.0140  0.0054  0.9957
```

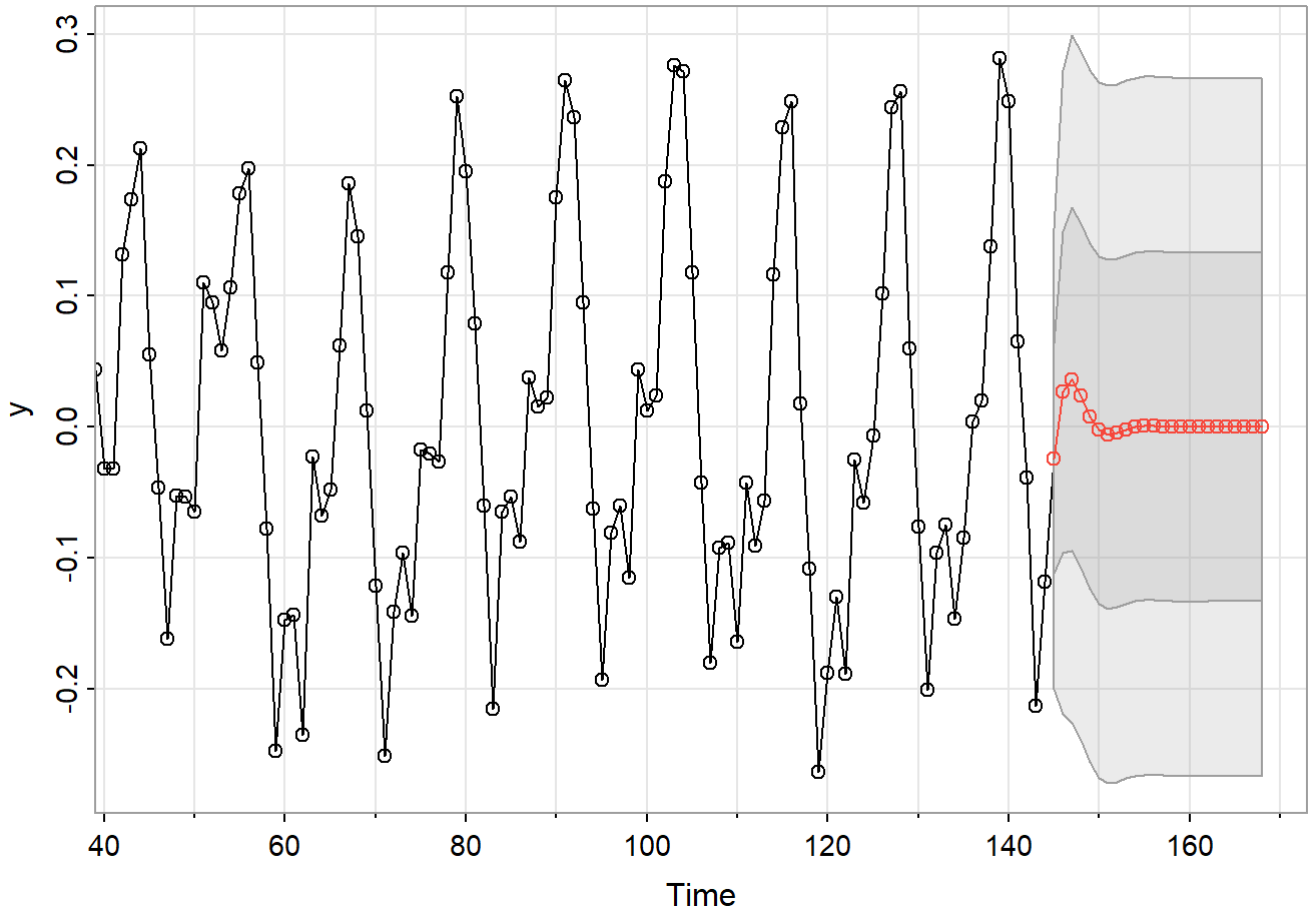
Vediamo tutte le previsione dell AR(p), con p = 1,2,3.

Il codice vuole fare previsioni future per 24 periodi utilizzando tre diversi modelli SARIMA con ordini di autoregressione crescenti (AR(1), AR(2), AR(3)).

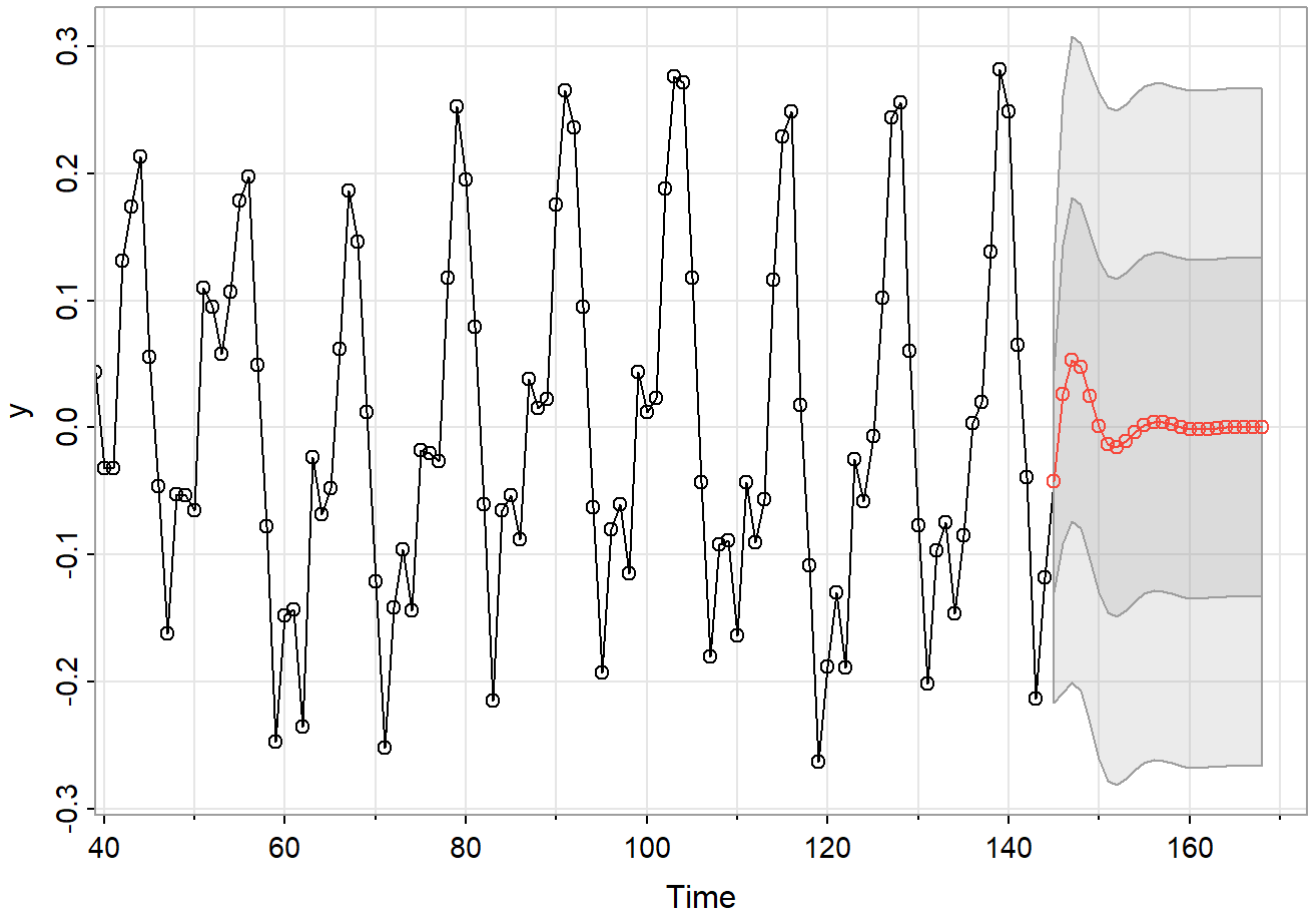
```
# Previsioni future per AR(1)
prev_ar1 = sarima.for(y,24,1,d,q,P,D,Q,s)
```




```
# Previsioni future per AR(2)
prev_ar2 = sarima.for(y,24,2,d,q,P,D,Q,s)
```



```
# Previsioni future per AR(3)
prev_ar3 = sarima.for(y,24,3,d,q,P,D,Q,s)
```



Se vogliamo vedere come il modello si adatta ai dati, possiamo usare l'AIC, che è un indice del tipo

$$AIC = -2ln(\hat{L}) + g(k)$$

dove $\hat{L}$ è il massimo della varosimiglianza, e $g(k) > 0$ è una funzione del numero dei parametri.

Il criterio di informazione di Akaike (AIC) è una misura che valuta la bontà di adattamento di un modello ai dati, tenendo conto della complessità del modello. Un valore più basso di AIC indica generalmente un modello migliore.

```
model_ar1$AIC
```

```
## [1] -1.77645
```

```
model_ar2$AIC
```

```
## [1] -1.961364
```

```
model_ar3$AIC
```

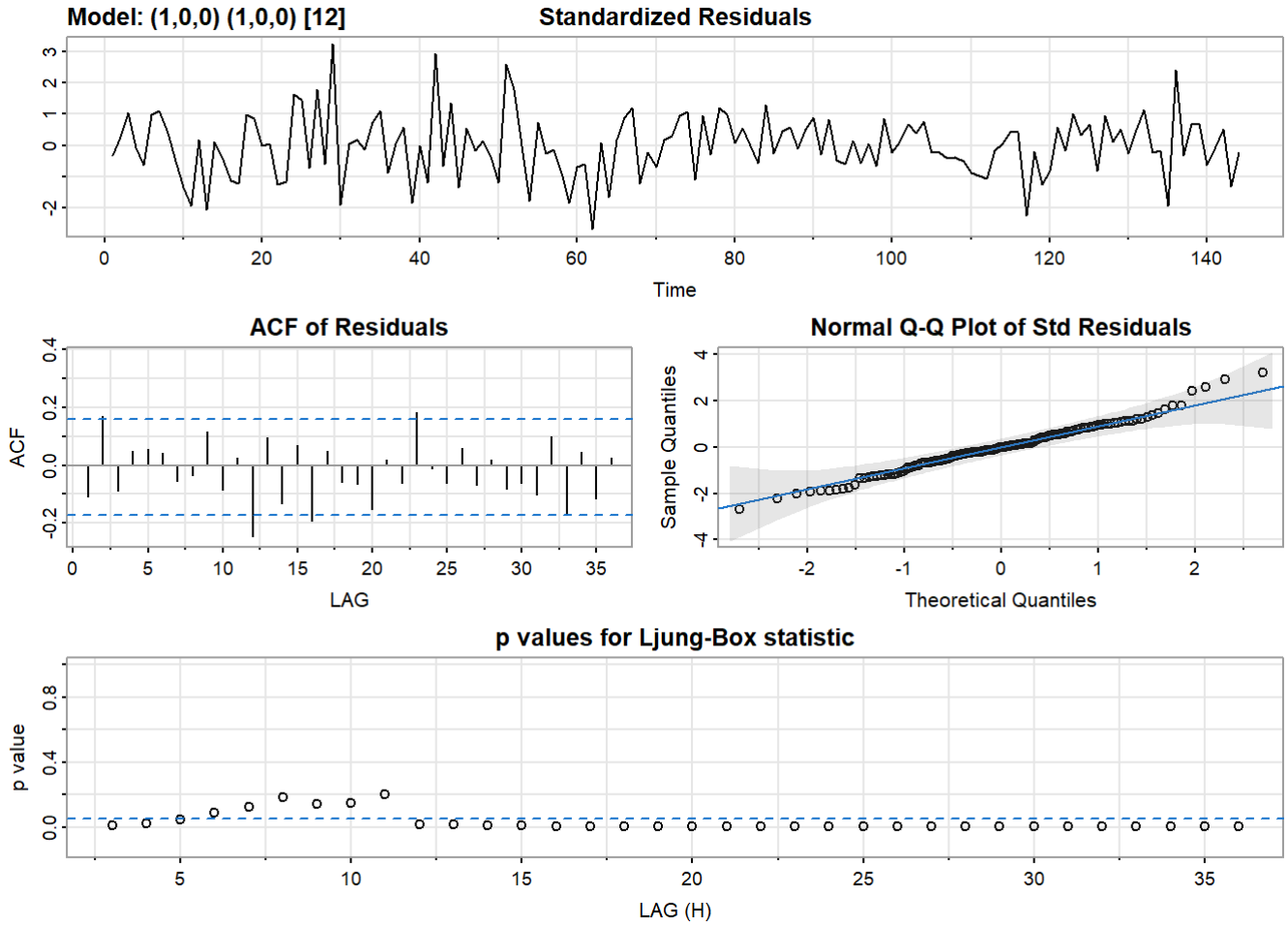
```
## [1] -1.970182
```

MODELLI STAGIONALI

Modello SARIMA(1,0,0)(1,0,0)12

```
p = 1 # Specifica l'ordine di autoregressione non stagionale (AR)
d = 0 # Specifica l'ordine di differenziazione non stagionale (I)
q = 0 # Specifica l'ordine di media mobile non stagionale (MA)
P = 1 # Specifica l'ordine di autoregressione stagionale
D = 0 # Specifica l'ordine di differenziazione stagionale
Q = 0 # Specifica l'ordine di media mobile stagionale
s = 12 # Specifica la stagionalità, indicando che la serie temporale ha una stagionalità di 1
2 periodi (ad esempio, mensile)
model_ar1_ars1 = sarima(y,p,d,q,P,D,Q,s) # Modello
```

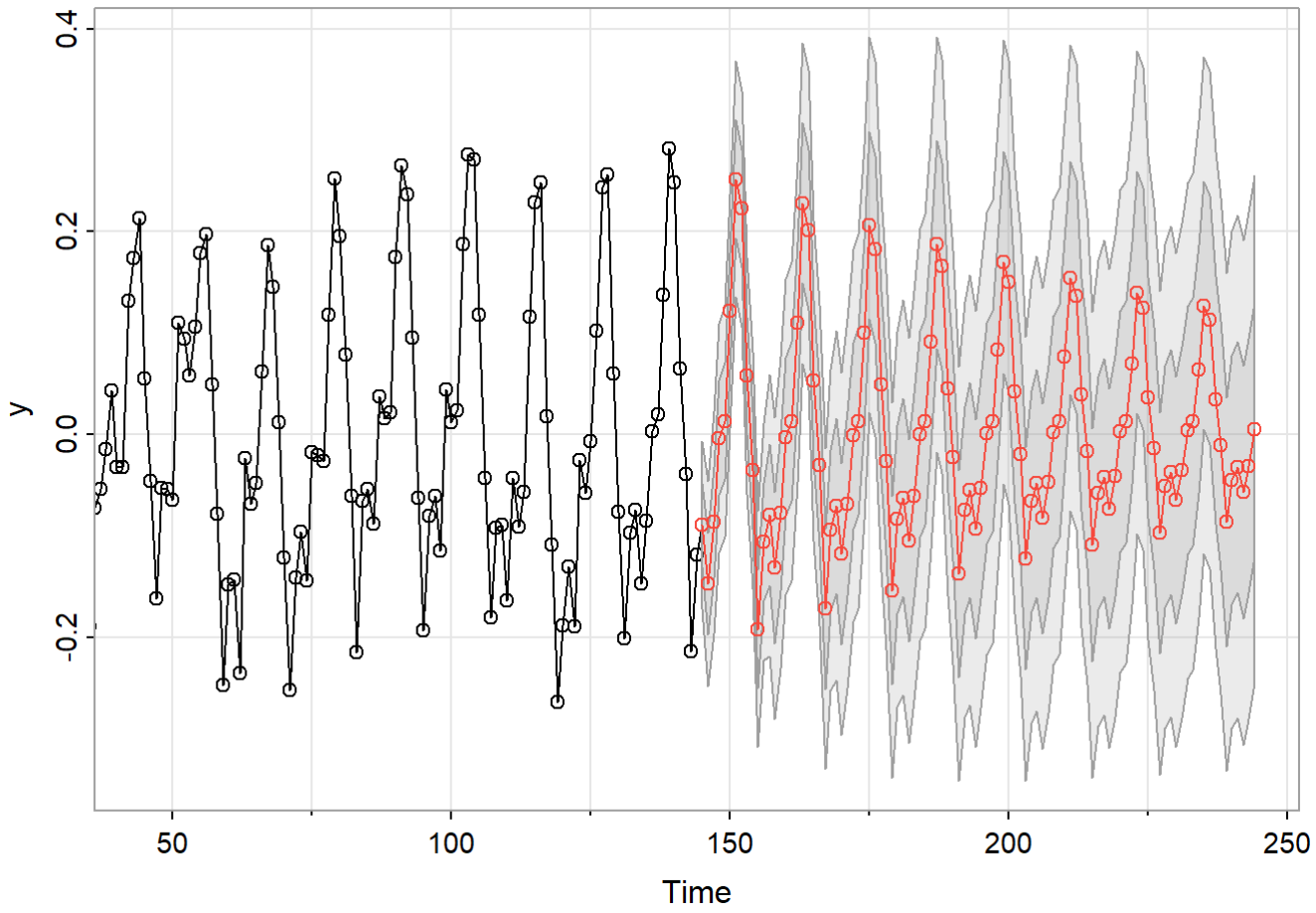
```
## initial  value -2.000973
## iter   2 value -3.191509
## iter   3 value -3.193622
## iter   4 value -3.193875
## iter   5 value -3.193876
## iter   6 value -3.193878
## iter   7 value -3.193887
## iter   8 value -3.193903
## iter   9 value -3.193909
## iter  10 value -3.193910
## iter  11 value -3.193912
## iter  12 value -3.193913
## iter  13 value -3.193916
## iter  14 value -3.193922
## iter  15 value -3.193935
## iter  16 value -3.193942
## iter  17 value -3.193943
## iter  18 value -3.193944
## iter  19 value -3.193944
## iter  20 value -3.193947
## iter  21 value -3.193953
## iter  22 value -3.193963
## iter  23 value -3.193968
## iter  24 value -3.193970
## iter  25 value -3.193971
## iter  26 value -3.193971
## iter  27 value -3.193972
## iter  28 value -3.193973
## iter  29 value -3.193976
## iter  30 value -3.193978
## iter  30 value -3.193978
## iter  30 value -3.193978
## final  value -3.193978
## converged
## initial  value -3.114908
## iter   2 value -3.115835
## iter   3 value -3.116738
## iter   4 value -3.116773
## iter   5 value -3.116794
## iter   6 value -3.116795
## iter   7 value -3.116830
## iter   7 value -3.116830
## iter   7 value -3.116830
## final  value -3.116830
## converged
```



Facciamo predizione.

Il codice effettua previsioni future per 100 periodi utilizzando un modello SARIMA (Seasonal AutoRegressive Integrated Moving Average) precedentemente adattato con specifici ordini di autoregressione e stagionalità sulla serie temporale y.

```
pred_ar1_ars1 = sarima.for(y,100,p,d,q,P,D,Q,s )
```

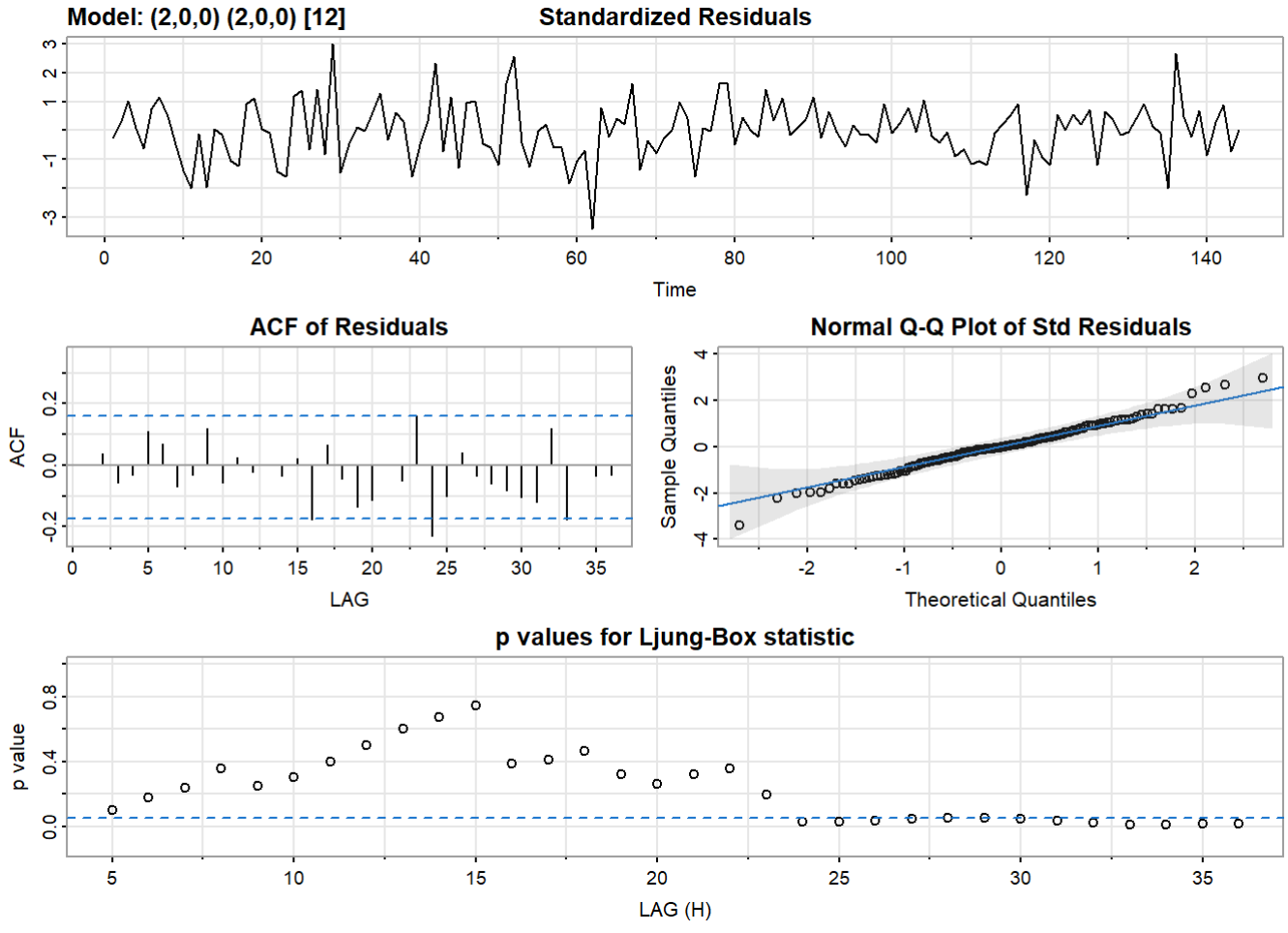


Possiamo provare diversi modelli.

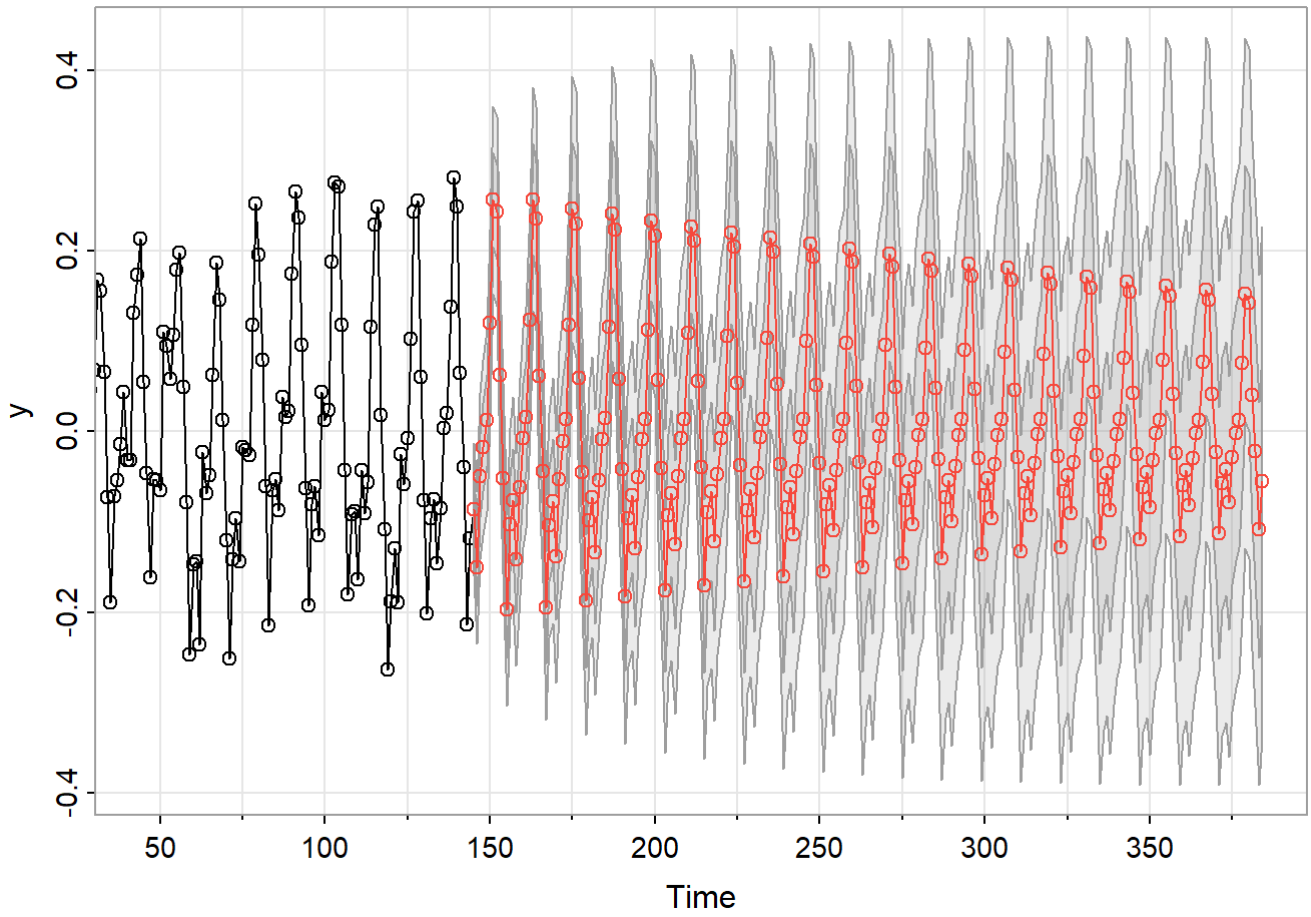
```
p = 2 # Specifica l'ordine di autoregressione non stagionale (AR)
d = 0 # Specifica l'ordine di differenziazione non stagionale (I)
q = 0 # Specifica l'ordine di media mobile non stagionale (MA)
P = 2 # Specifica l'ordine di autoregressione stagionale
D = 0 # Specifica l'ordine di differenziazione stagionale
Q = 0 # Specifica l'ordine di media mobile stagionale
s = 12 # Specifica la stagionalità, indicando che la serie temporale ha una stagionalità di 1
2 periodi (ad esempio, mensile)

# Adatta il modello SARIMA
model_ar2_ars2 = sarima(y, p, d, q, P, D, Q, s)
```

```
## initial  value -1.991065
## iter   2 value -2.344310
## iter   3 value -3.172378
## iter   4 value -3.242332
## iter   5 value -3.288178
## iter   6 value -3.295696
## iter   7 value -3.315253
## iter   8 value -3.316507
## iter   9 value -3.316709
## iter  10 value -3.316712
## iter  11 value -3.316713
## iter  12 value -3.316713
## iter  13 value -3.316714
## iter  14 value -3.316714
## iter  15 value -3.316715
## iter  16 value -3.316716
## iter  17 value -3.316724
## iter  18 value -3.316739
## iter  19 value -3.316778
## iter  20 value -3.316850
## iter  21 value -3.316941
## iter  22 value -3.317011
## iter  23 value -3.317030
## iter  24 value -3.317030
## iter  25 value -3.317031
## iter  26 value -3.317031
## iter  26 value -3.317031
## iter  26 value -3.317031
## final  value -3.317031
## converged
## initial  value -3.200537
## iter   2 value -3.203488
## iter   3 value -3.203824
## iter   4 value -3.204116
## iter   5 value -3.204218
## iter   6 value -3.204294
## iter   7 value -3.204375
## iter   8 value -3.204492
## iter   9 value -3.204646
## iter  10 value -3.204886
## iter  11 value -3.205297
## iter  12 value -3.205892
## iter  13 value -3.206255
## iter  14 value -3.206488
## iter  15 value -3.206528
## iter  16 value -3.206549
## iter  17 value -3.206566
## iter  18 value -3.206580
## iter  19 value -3.206593
## iter  20 value -3.206613
## iter  21 value -3.206652
## iter  22 value -3.206724
## iter  23 value -3.206821
## iter  24 value -3.206866
## iter  25 value -3.206886
```

```
## iter  26 value -3.206889
## iter  27 value -3.206890
## iter  28 value -3.206891
## iter  29 value -3.206892
## iter  30 value -3.206893
## iter  31 value -3.206895
## iter  32 value -3.206897
## iter  33 value -3.206900
## iter  34 value -3.206905
## iter  35 value -3.206907
## iter  36 value -3.206908
## iter  37 value -3.206909
## iter  38 value -3.206909
## iter  39 value -3.206909
## iter  40 value -3.206909
## iter  41 value -3.206909
## iter  42 value -3.206909
## iter  43 value -3.206909
## iter  44 value -3.206910
## iter  45 value -3.206910
## iter  46 value -3.206910
## iter  47 value -3.206910
## iter  47 value -3.206910
## iter  47 value -3.206910
## final  value -3.206910
## converged
```



```
# Effettua previsioni future per 10 anni (24*10 periodi)
pred_ar2_ars2 = sarima.for(y, 24*10, p, d, q, P, D, Q, s)
```



Abbiamo fatto la predizione, ma nella serie dei residui della regressione. Generalmente siamo interessati alla predizione della serie originale.

La funzione str() in R è utilizzata per visualizzare la struttura di un oggetto. Quando applicata a un oggetto, restituirà una rappresentazione strutturata delle sue componenti interne. Nell'esempio delle previsioni future pred_ar2_ars2, l'output di str() fornirà informazioni sulla struttura dell'oggetto, inclusi i componenti principali.

```
str(pred_ar2_ars2)
```

```
## List of 2
##  $ pred: Time-Series [1:240] from 145 to 384: -0.0862 -0.1505 -0.0494 -0.0175 0.012 ...
##  $ se  : Time-Series [1:240] from 145 to 384: 0.0366 0.042 0.0462 0.0486 0.0501 ...
```

L'oggetto della predizione contiene il valore atteso, e una stima della radice della varianza. Qiuando facciamo i plot della predizione, quello che ci mostra è l'intervallo

$$[\hat{y}_t - z_{\alpha/2}\sqrt{\hat{var}(y_t)}, \hat{y}_t + z_{\alpha/2}\sqrt{\hat{var}(y_t)}]$$

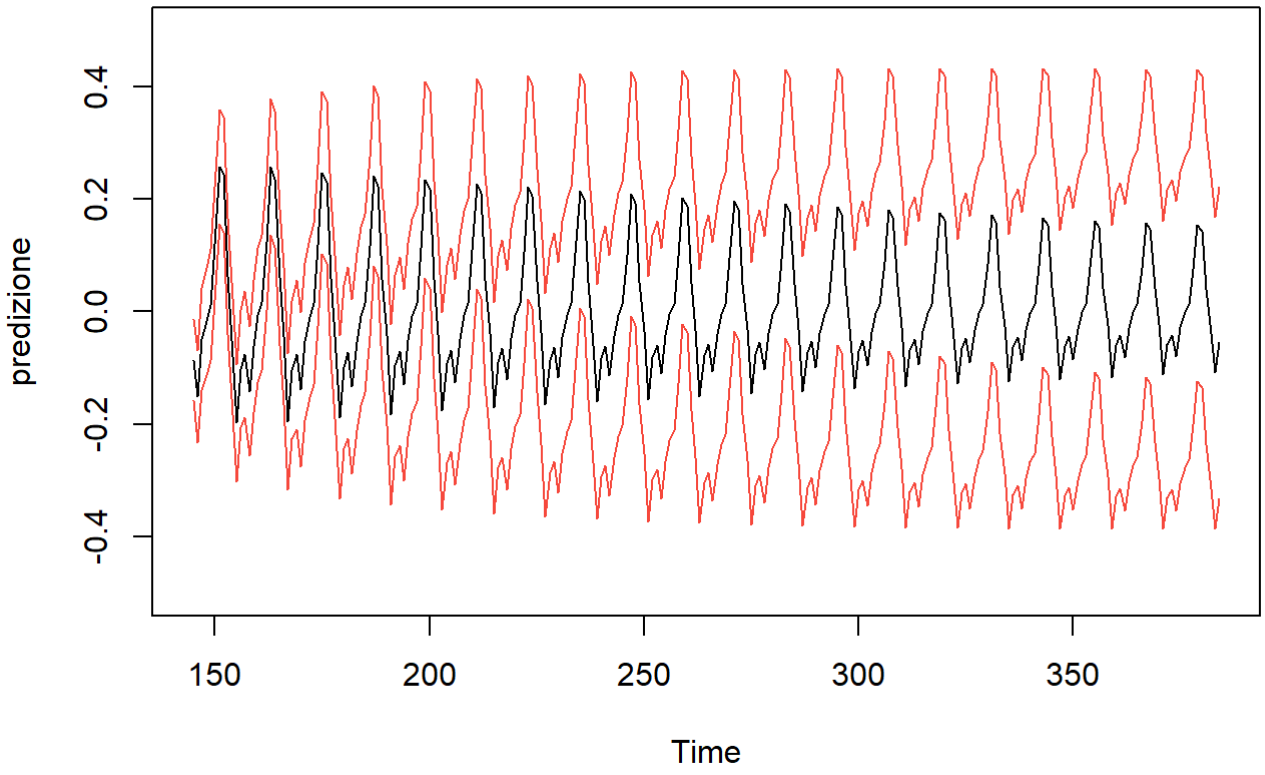
oppure

$$[\hat{y}_t - t_{T-1,\alpha/2}\sqrt{\hat{var}(y_t)}, \hat{y}_t + t_{T-1,\alpha/2}\sqrt{\hat{var}(y_t)}]$$

Vediamo la predizione nella serie storica

Il codice crea un grafico della previsione insieme agli intervalli di confidenza al 95%.

```
# Estrae la previsione dal risultato della previsione
predizione = pred_ar2_ars2$pred # Serie temporale

# Calcola gli estremi degli intervalli di confidenza al 95%
q1_pred = predizione - qnorm(0.05/2, 0, 1, lower.tail = FALSE) * pred_ar2_ars2$se
q2_pred = predizione + qnorm(0.05/2, 0, 1, lower.tail = FALSE) * pred_ar2_ars2$se

# Crea un grafico della previsione con intervalli di confidenza
plot(predizione, ylim = c(-0.5, 0.5))
lines(q1_pred, col = 2)
lines(q2_pred, col = 2)
```



Questo tipo di visualizzazione è comune per mostrare la precisione delle previsioni e fornire una rappresentazione visiva degli intervalli di confidenza.

Ricordiamo che x_t è la serie originale, e che

$$log(x_t) = \beta_0 + \beta_1 t + \beta_2 t^2 + \epsilon_t$$

e dove

$$\epsilon_t$$

è la serie storica. Posso quindi prevedere il logariotmo di x_t.

Il codice estende le previsioni di un modello di regressione polinomiale di secondo grado al futuro.

```
# Adatta un modello di regressione polinomiale di secondo grado ai dati logaritmici
reg = lm(log_passenger ~ time + I(time^2))

# Crea una sequenza di nuovi tempi per le previsioni future
time_pred = max(time) + seq(1, 24*10)/12

# Calcola le previsioni del modello di regressione polinomiale
pred_log_reg = reg$coefficients[1] + reg$coefficients[2]*time_pred + reg$coefficients[3]*time
_pred^2

# Aggiungi le previsioni SARIMA al modello di regressione
pred_log = pred_log_reg + predizione
q1_pred_log = pred_log_reg + q1_pred
q2_pred_log = pred_log_reg + q2_pred
```

In questo modo, stai combinando le previsioni del modello SARIMA con le previsioni di un modello di regressione polinomiale. Puoi utilizzare questi risultati per ottenere previsioni più complesse che tengono conto sia delle tendenze polinomiali che delle componenti stagionali o autoregressive della serie temporale.

Il codice è utilizzato per tracciare i dati osservati insieme alle previsioni del modello di regressione polinomiale e alle previsioni del modello SARIMA, compresi gli intervalli di confidenza.

```
# Crea un grafico dei dati osservati
plot(time, log_passenger, type="l", xlim=c(1948, 1972), ylim=c(4, 7), main="Log(Passenger) an
d Predictions")

# Aggiunge le previsioni del modello di regressione polinomiale (linea tratteggiata)
lines(time_pred, pred_log_reg, lty=2, col="blue", label="Poly. Regression")
```

```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): parametro grafico "label"
## non valido
```

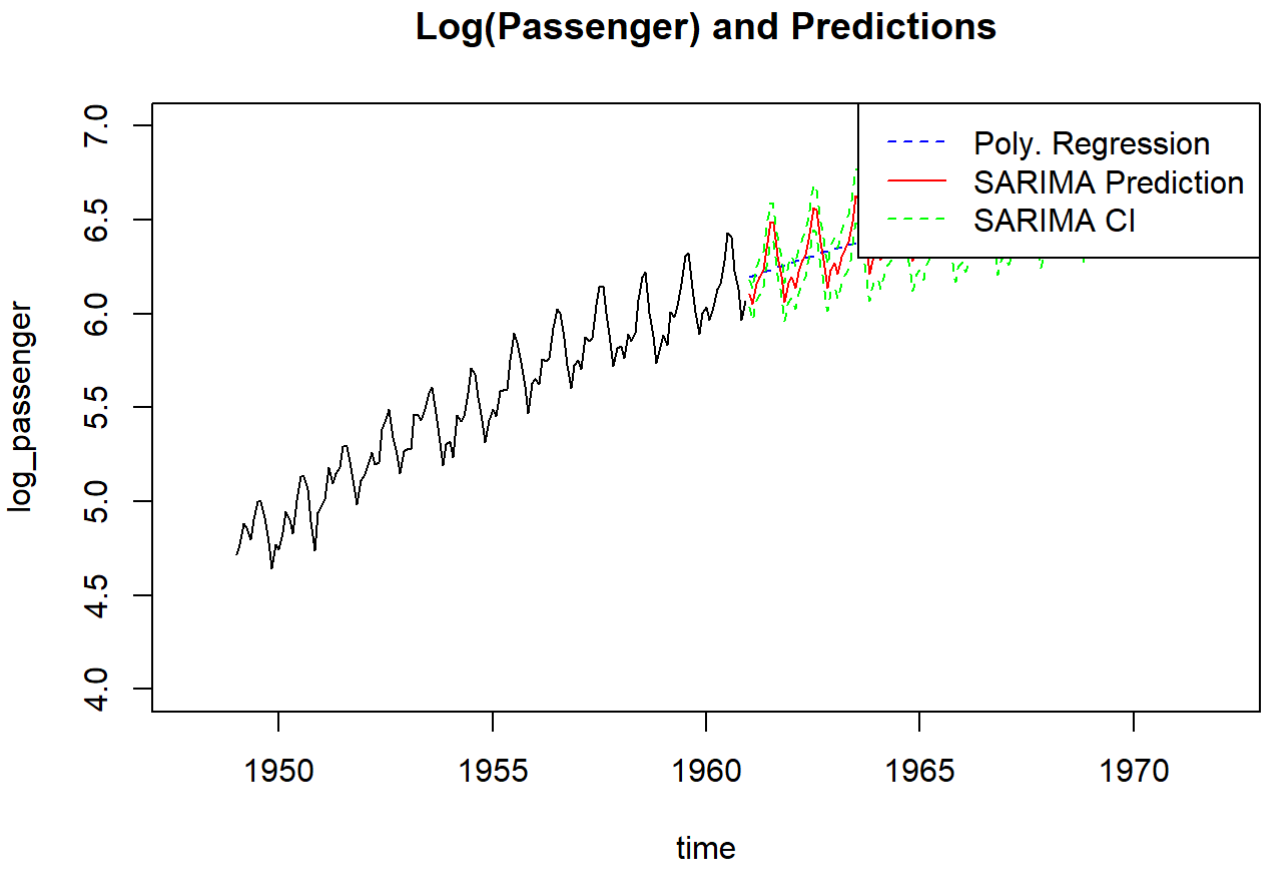
```
# Aggiunge le previsioni del modello SARIMA (linea continua)
lines(time_pred, pred_log, col="red", label="SARIMA Prediction")
```

```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): parametro grafico "label"
## non valido
```

```
# Aggiunge gli intervalli di confidenza del modello SARIMA
lines(time_pred, q1_pred_log, col="green", lty=2, label="SARIMA CI")
```

```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): parametro grafico "label"
## non valido
```

```
lines(time_pred, q2_pred_log, col="green", lty=2)

# Aggiunge una legenda
legend("topright", legend=c("Poly. Regression", "SARIMA Prediction", "SARIMA CI"), col=c("blu
e", "red", "green"), lty=c(2, 1, 2))
```



Vogliamo adesos fare la predizione nella scala originale. Fate atetnzioen che se

$$\log(x_t) \sim N()$$

non è vero che

$$x_t \sim N()$$

Quindi, anche se $x_t = e^{(log(x_t))}$

$$E(x_t) \neq e^{(E(log(x_t)))}$$

ma

$$E(x_t) \approx e^{(E(log(x_t)))}$$

La cosa interessante è che se la trasfomrazione è monotona crescente (vale anche per il decrescente), tipo il logaritmo

$$P(log(X_t) < a) = 0.975 = P(X_t < exp(a)).$$

Il codice è utilizzato per tracciare i dati osservati insieme alle previsioni trasformate all'originale scala delle osservazioni.

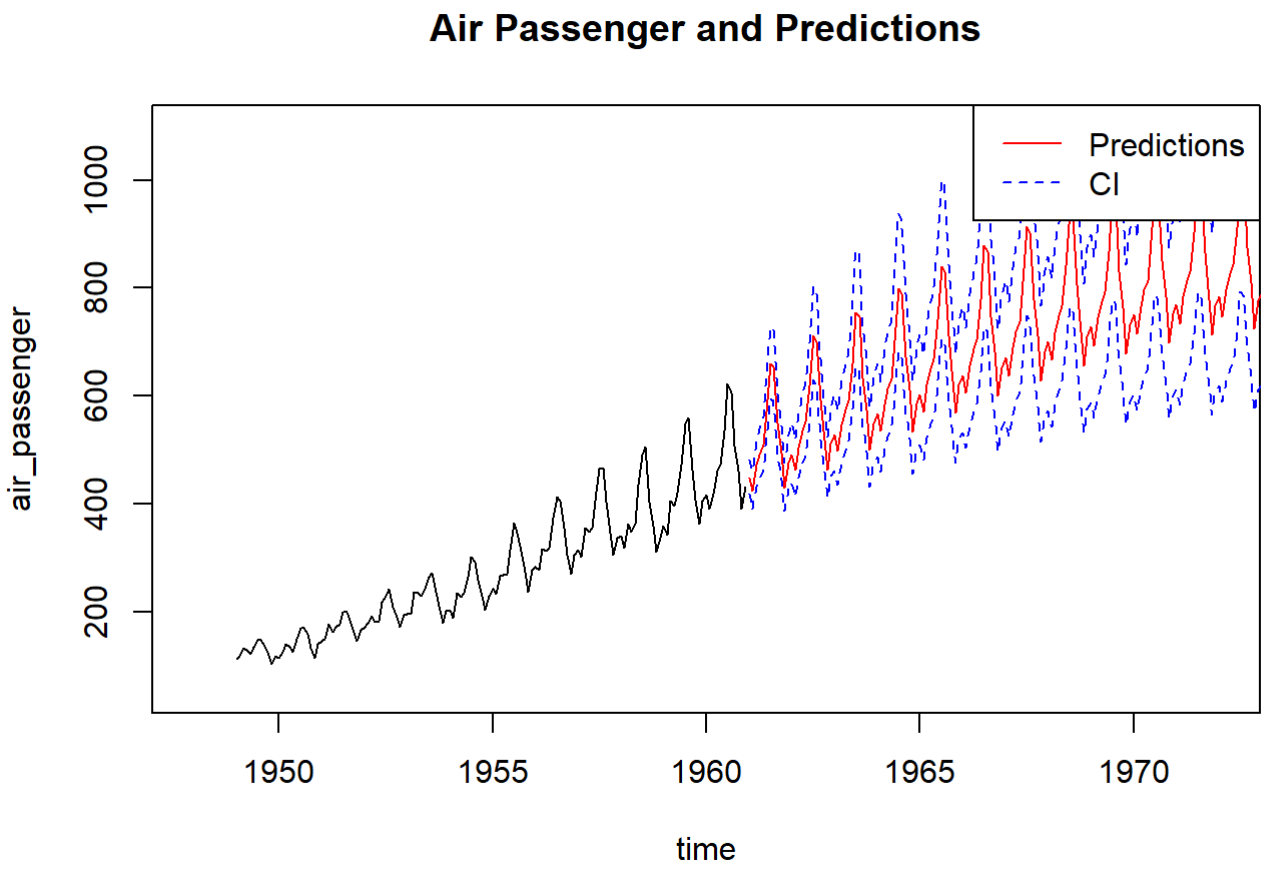
```
# Calcola le previsioni sulla scala originale
pred_scale_orig = exp(pred_log_reg + predizione)
q1_pred_scale_orig = exp(pred_log_reg + q1_pred)
q2_pred_scale_orig = exp(pred_log_reg + q2_pred)

# Crea un grafico dei dati osservati sulla scala originale
plot(time, air_passenger, type="l", xlim=c(1948, 1972), ylim=c(exp(4), exp(7)), main="Air Pas
senger and Predictions")

# Aggiunge le previsioni del modello sulla scala originale (linea continua in rosso)
lines(time_pred, pred_scale_orig, col="red", label="Predictions")
```

```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): parametro grafico "label"
## non valido
```

```
# Aggiunge gli intervalli di confidenza sulla scala originale
lines(time_pred, q1_pred_scale_orig, col="blue", lty=2, label="CI")
```

```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): parametro grafico "label"
## non valido
```

```
lines(time_pred, q2_pred_scale_orig, col="blue", lty=2)

# Aggiunge una legenda
legend("topright", legend=c("Predictions", "CI"), col=c("red", "blue"), lty=c(1, 2))
```



Questo grafico fornisce una rappresentazione visiva delle previsioni future e degli intervalli di confidenza sulle scale originali dei dati osservati.

DIFFERENZIAZIONI e ELIMINAZIONE DEL TREND

Possimao “trattare” il trend con delle differenziazioni (quindi aprte del modello SARIMA).

Nel codice sottostante si adattata un modello SARIMA con le seguenti specifiche:

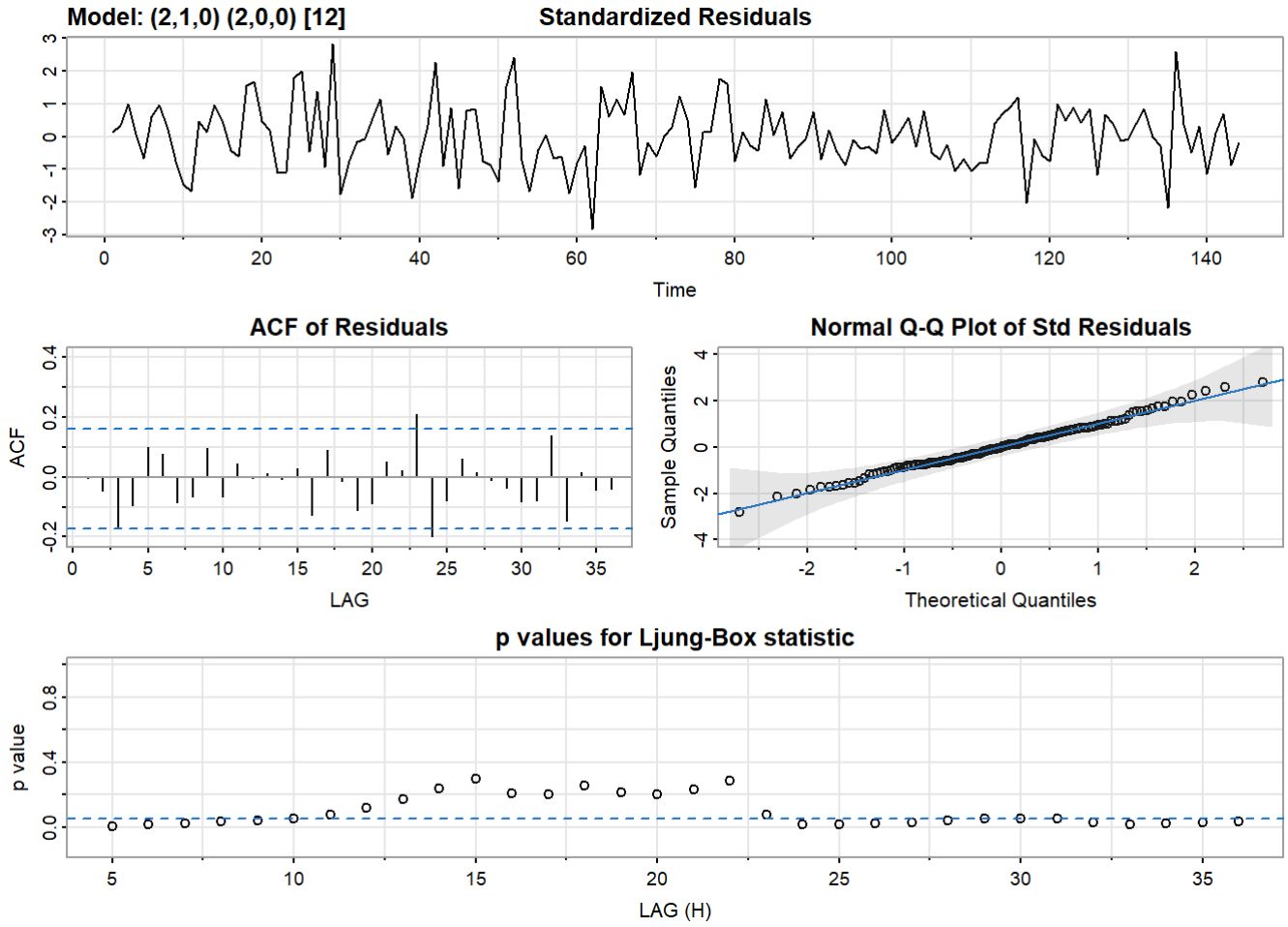
- Ordine di autoregressione non stagionale (p): 2
- Ordine di differenziazione non stagionale (d): 1
- Ordine di media mobile non stagionale (q): 0
- Ordine di autoregressione stagionale (P): 2
- Ordine di differenziazione stagionale (D): 0
- Ordine di media mobile stagionale (Q): 0 Periodicità stagionale (s): 12

Hai quindi effettuato previsioni future per 2 anni (24*2 periodi).

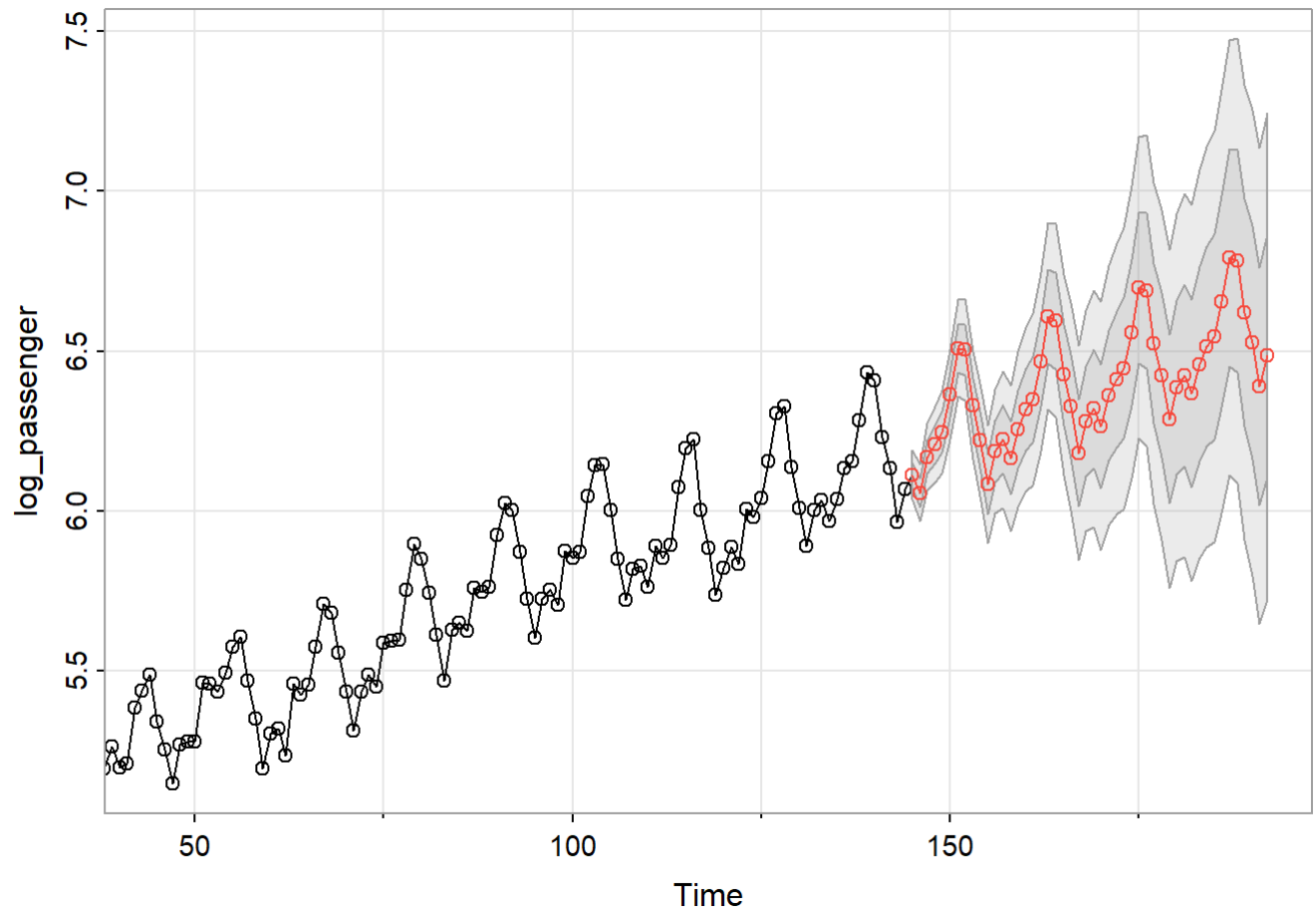
Ricorda che l’ordine di differenziazione non stagionale d=1 indica che hai applicato la differenziazione di prima differenza alla serie temporale logaritmica log_passenger.

```
p=2
d=1
q=0
P=2
D=0
Q=0
s = 12
model_ar2_ars2_d1 = sarima(log_passenger ,p,d,q,P,D,Q,s)
```

```
## initial  value -2.246490
## iter   2 value -2.281397
## iter   3 value -3.125618
## iter   4 value -3.167835
## iter   5 value -3.235652
## iter   6 value -3.254819
## iter   7 value -3.279531
## iter   8 value -3.281057
## iter   9 value -3.281199
## iter  10 value -3.281228
## iter  11 value -3.281257
## iter  12 value -3.281298
## iter  13 value -3.281301
## iter  14 value -3.281316
## iter  15 value -3.281347
## iter  16 value -3.281364
## iter  17 value -3.281437
## iter  18 value -3.281527
## iter  19 value -3.281623
## iter  20 value -3.281667
## iter  21 value -3.281673
## iter  22 value -3.281673
## iter  22 value -3.281673
## iter  22 value -3.281673
## final  value -3.281673
## converged
## initial  value -3.161024
## iter   2 value -3.164366
## iter   3 value -3.165086
## iter   4 value -3.166599
## iter   5 value -3.167398
## iter   6 value -3.167569
## iter   7 value -3.167589
## iter   8 value -3.167592
## iter   9 value -3.167594
## iter  10 value -3.167595
## iter  11 value -3.167595
## iter  11 value -3.167595
## iter  11 value -3.167595
## final  value -3.167595
## converged
```




```
pred_ar2_ars2_d1 = sarima.for(log_passenger,24*2,p,d,q,P,D,Q,s)
```



Vediamo cosa succede se stimiamo un modello senza la differenziazione.

Il codice adatta un modello SARIMA con le seguenti specifiche:

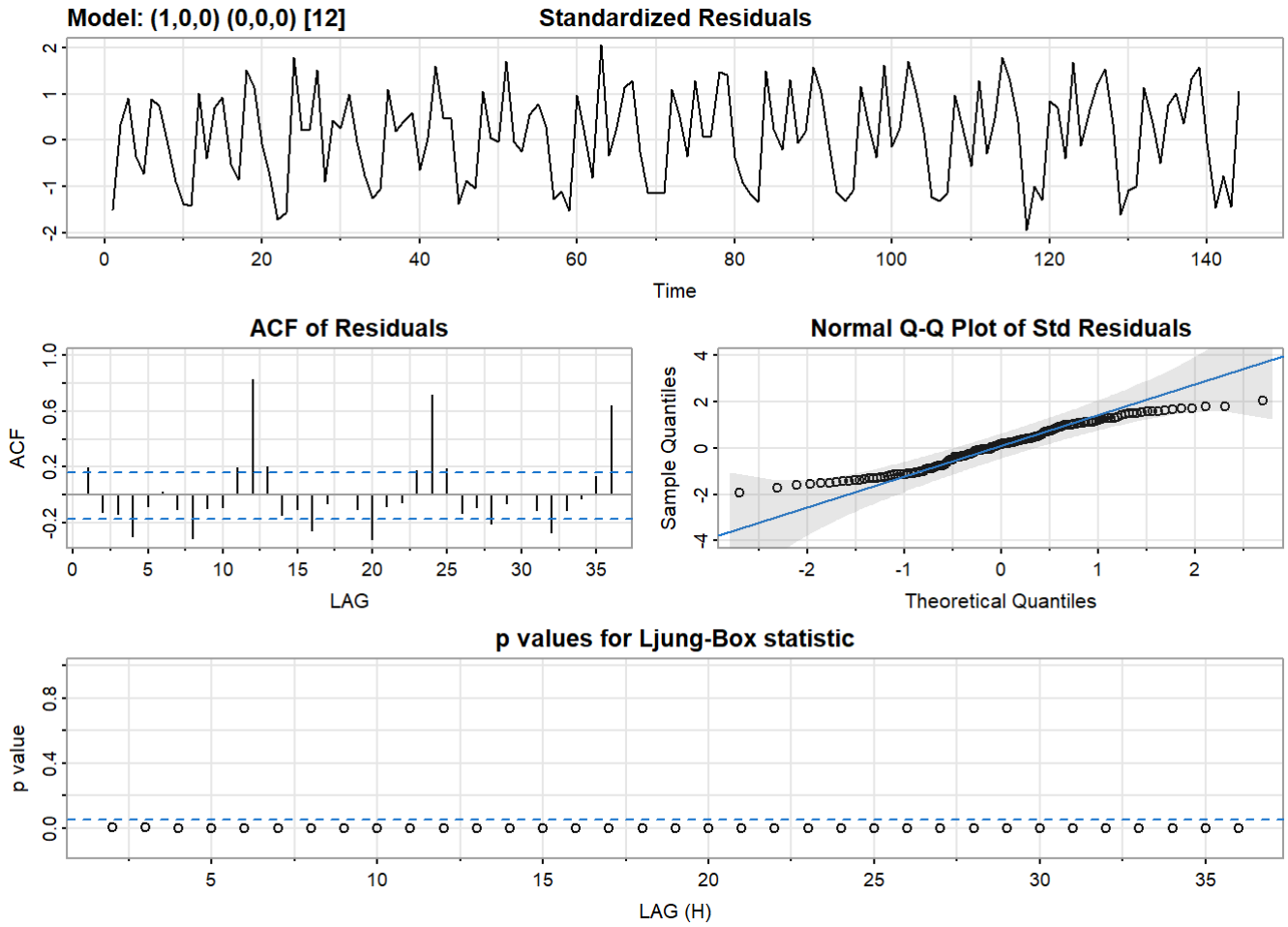
- Ordine di autoregressione non stagionale (p): 1
- Ordine di differenziazione non stagionale (d): 0
- Ordine di media mobile non stagionale (q): 0
- Ordine di autoregressione stagionale (P): 0
- Ordine di differenziazione stagionale (D): 0
- Ordine di media mobile stagionale (Q): 0
- Periodicità stagionale (s): 12

Questa specifica indica un modello ARIMA con un termine autoregressivo di ordine 1 (AR(1)) sulla parte non stagionale. Inoltre, la serie è stata considerata non stagionale, quindi non sono stati inclusi componenti stagionali.

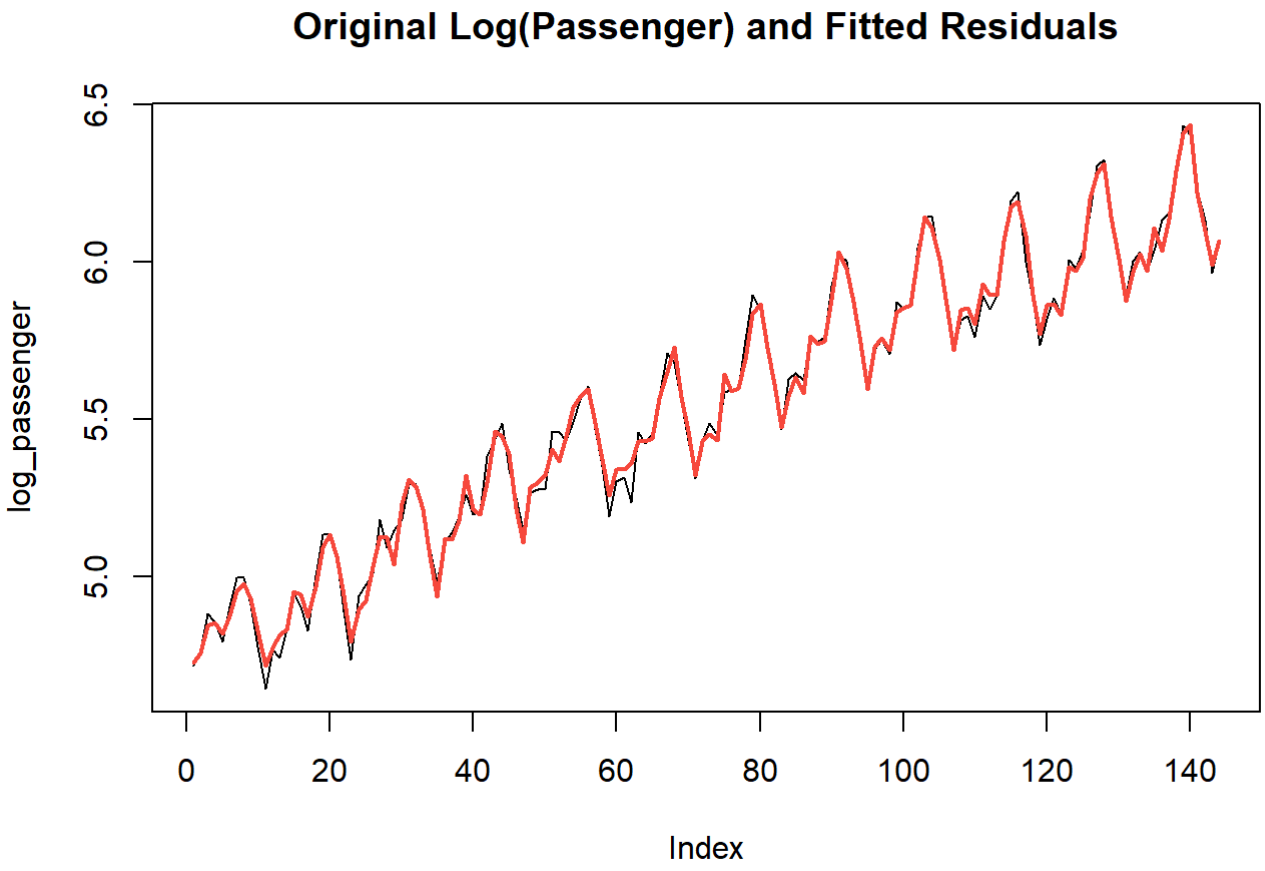
```
p=1
d=0
q=0
P=0
D=0
Q=0
s = 12
model_ar1 = sarima(log_passenger ,p,d,q,P,D,Q,s)
```

```
## initial  value -0.829999
## iter   2 value -2.248560
## iter   3 value -2.250003
## iter   4 value -2.250493
## iter   5 value -2.250645
## iter   6 value -2.251785
## iter   7 value -2.252303
## iter   8 value -2.252413
## iter   9 value -2.252541
## iter  10 value -2.252943
## iter  11 value -2.253590
## iter  12 value -2.253823
## iter  13 value -2.253841
## iter  14 value -2.253859
## iter  15 value -2.253919
## iter  16 value -2.254000
## iter  17 value -2.254051
## iter  18 value -2.254055
## iter  19 value -2.254063
## iter  20 value -2.254082
## iter  21 value -2.254113
## iter  22 value -2.254129
## iter  23 value -2.254130
## iter  24 value -2.254132
## iter  25 value -2.254136
## iter  26 value -2.254143
## iter  27 value -2.254147
## iter  28 value -2.254147
## iter  29 value -2.254148
## iter  30 value -2.254149
## iter  31 value -2.254151
## iter  32 value -2.254152
## iter  32 value -2.254152
## iter  32 value -2.254152
## final  value -2.254152
## converged
## initial  value -2.222794
## iter   2 value -2.224612
## iter   3 value -2.228908
## iter   4 value -2.229816
## iter   5 value -2.230136
## iter   6 value -2.230206
## iter   7 value -2.230224
## iter   8 value -2.231597
## iter   9 value -2.231670
## iter  10 value -2.231714
## iter  11 value -2.231729
## iter  12 value -2.231741
## iter  13 value -2.231851
## iter  14 value -2.231854
## iter  15 value -2.231855
## iter  16 value -2.231859
## iter  17 value -2.231859
## iter  18 value -2.231879
## iter  19 value -2.231880
```

```
## iter  20 value -2.231881
## iter  21 value -2.231885
## iter  22 value -2.231885
## iter  23 value -2.231893
## iter  23 value -2.231893
## iter  23 value -2.231893
## final  value -2.231893
## converged
```




```
# Calcola la componente residua sottraendo i residui del modello SARIMA adattato dalla serie logaritmica
fit_model_ar2_ars2 = log_passenger - residuals(model_ar2_ars2$fit)

# Crea un grafico della serie temporale logaritmica originale
plot(log_passenger, type="l", main="Original Log(Passenger) and Fitted Residuals")

# Aggiunge la componente residua del modello SARIMA
lines(fit_model_ar2_ars2, col=2, lwd=2)
```



Il grafico risultante mostra la serie temporale logaritmica originale insieme alla componente residua del modello SARIMA adattato. Questo è utile per visualizzare quanto bene il modello SARIMA spieghi la variabilità nella serie temporale. Se la componente residua sembra avere una struttura, potrebbe indicare che il modello non ha catturato completamente tutti i modelli nella serie temporale.

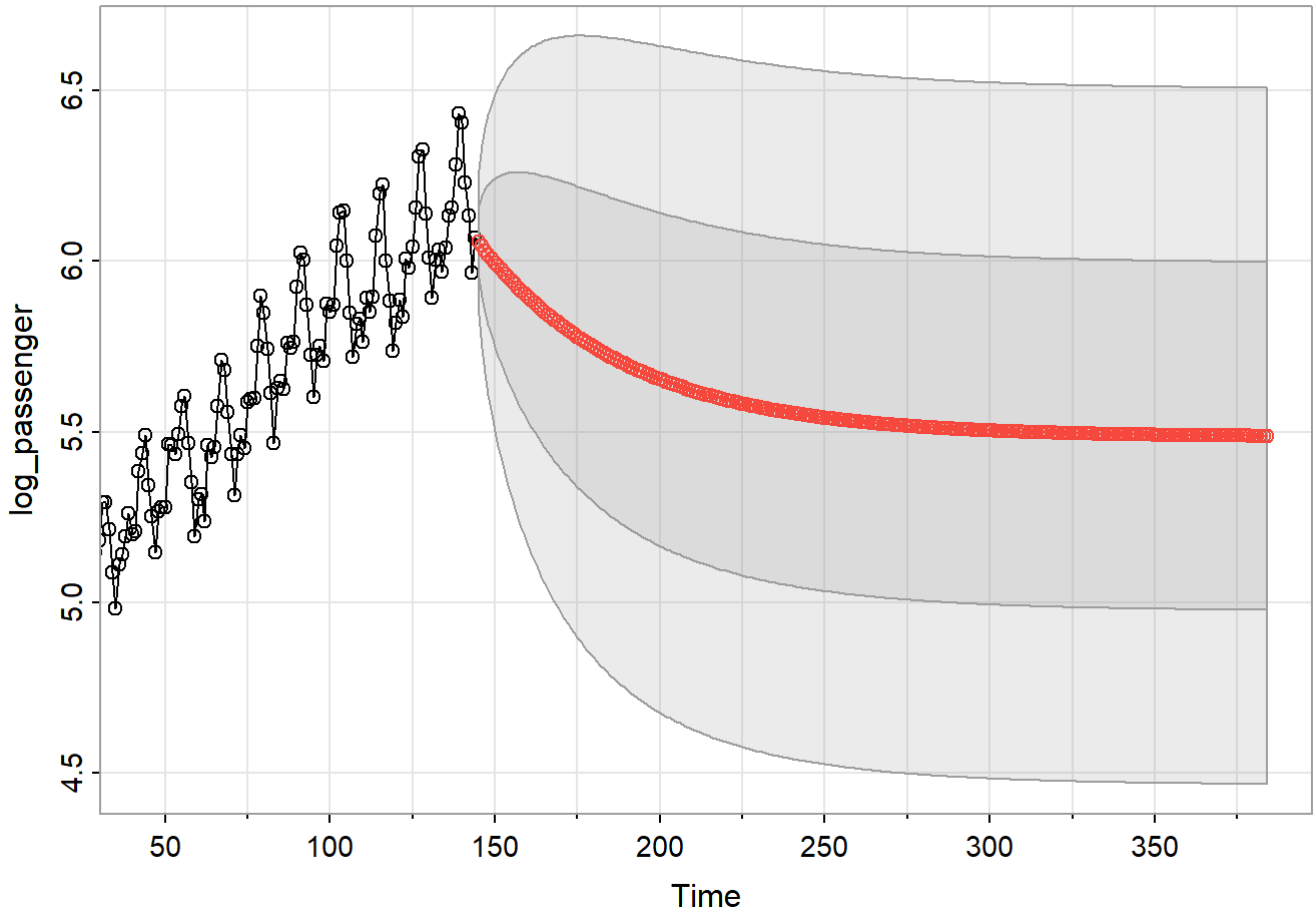
Se provo a fare predizione con questo modello, ne vedo i problemi.

Si utilizza la funzione sarima.for per ottenere previsioni future basate sul modello SARIMA adattato con le seguenti specifiche:

- Ordine di autoregressione non stagionale (p): 1
- Ordine di differenziazione non stagionale (d): 0
- Ordine di media mobile non stagionale (q): 0
- Ordine di autoregressione stagionale (P): 0
- Ordine di differenziazione stagionale (D): 0
- Ordine di media mobile stagionale (Q): 0
- Periodicità stagionale (s): 12

Si effettua previsioni per i successivi 10 anni (24*10 periodi).

```
p=1
d=0
q=0
P=0
D=0
Q=0
s = 12
pred_ar2_ars2 = sarima.for(log_passenger,24*10,p,d,q,P,D,Q,s)
```


log_passenger
6.5
6.0
5.5
5.0
4.5
50
100
150
200
250
300
350
Time