Práctica 4: Modelos ARMA

1 Introducción

En esta práctica se mostrará cómo realizar un ajuste mediante modelos ARMA a series temporales en R. La explicación se desarrollará en cuatro apartados:

1. Simulación de modelos ARMA.
2. Funciones de autocorrelación, autocorrelación parcial y autocorrelación parcial extendida muestrales.
3. Estimación de parámetros en un modelo ARMA y diagnóstico del ajuste.
4. Predicción en modelos ARMA.

Nota: Todas las explicaciones de este tema para modelos ARMA se podrán extrapolar a los modelos ARIMA que estudiaremos en la siguiente práctica. En esta práctica se adaptarán las funciones a modelos ARMA con tan sólo indicar que cierto parámetro toma el valor 0 en las funciones. Cuando tratemos con modelos ARIMA, simplemente se permitirá que ese parámetro tome valores enteros mayores que 0.

OJO!! para R los coeficientes del modelo MA o de la parte MA en un modelo ARMA (también ARIMA) tienen el signo cambiado, con respecto a como se han visto en teoría:

- Modelo MA(q): $Y_t = \alpha_t + \theta_1\alpha_{t-1} + \theta_2\alpha_{t-2} + \ldots + \theta_q\alpha_{t-q}$.
- Modelo ARMA(p,q): $Y_t = \phi_1 Y_{t-1} + \ldots + \phi_p Y_{t-p} + \alpha_t + \theta_1\alpha_{t-1} + \ldots + \theta_q\alpha_{t-q}$

Si quiero estudiar el modelo
Yt = alfat-0.7alfa t-1
tengo que meter en R
theta1= -0.7
a mano theta1 = 0.7

2 Simulación de modelos

Como primer paso, se estudiará cómo simular modelos ARMA de los órdenes que se desee. Usaremos estas simulaciones en los apartados posteriores para ilustrar resultados.

2.1 La función arima.sim

Se usa para simular un modelo ARIMA. Su sintaxis es la siguiente:

La normal es por defecto

```
arima.sim(model, n, rand.gen = rnorm, innov = rand.gen(n, ...),
          n.start = NA, start.innov = rand.gen(n.start, ...),   ...)
```

donde

- `model`: Una lista con las componentes de los coeficientes de las partes AR y/o MA. Opcionalmente se puede usar `order` y especificar un vector con tres órdenes, para modelos integrados. Si la lista es vacía se usará un modelo ARIMA(0, 0, 0).
- `n`: Número de valores a generar de la serie, sin diferenciar. Debe ser entero positivo.
- `innov`: La serie de tiempo de las innovaciones. Por defecto rand.gen
- `rand.gen`: Es opcional: la function para generar las innovaciones Por defecto, genera datos i.i.d. de una normal standard
- `start.innov`: Serie de tiempo opcional para generar las innovaciones de un periodo de prueba
- `n.start`: Longitud del periodo de prueba. Si es NA, por defecto, se calcula un valor razonable.
- `sd`: desviación estándard de las innovaciones generadas por `rnorm`.

La salida en un objeto de la clase `ts`.

Por ejemplo:

- Para generar $n = 63$ datos del modelo ARMA(1,2):

$$X_t = 0.8X_{t-1} + \alpha_t - 0.2\alpha_{t-1} + 0.4\alpha_{t-2}, \quad \{\alpha_t\}\ iid\ N(0, \sigma^2), \quad \sigma^2 = 0.1796,$$

escribimos

si no se pone nada se entiende que d es 0
order SOLO es necesario si d>= 1

vector de long q vector de long q

```
arima.sim(n = 63, list(ar = 0.8, ma = c(-0.2, 0.4)), sd = sqrt(0.1796))
```

siempre en forma de lista

- Para generar $n = 200$ datos de $\{\alpha_t\}\ iid\ N(0, 1)$, escribimos

(p,d,q)

```
arima.sim(list(order = c(0,0,0)), n=200)
```

- Para generar $n = 200$ datos del modelo AR(1)

$$X_t = 0.7X_{t-1} + \alpha_t, \quad \{\alpha_t\}\ iid\ t_5,$$

escribimos

```
arima.sim(list(order = c(1,0,0), ar=0.7), n=200, rand.gen=rt, df=5)
```

ó simplemente

```
arima.sim(list(ar=0.7), n=200, rand.gen=rt, df=5)
```

2.2 Las funciones diff y diffinv

para diferenciar y deshacer el cambio

La función `diff` se usa para calcular diferencias de series. Se deshace con `diffinv`. La sintaxis es la siguiente:

x es un objeto de la serie ts
orden de la diferencia

```
diff(x, lag = 1, differences = 1, ...)
diffinv(x, lag = 1, differences = 1, xi = 1, ...)
```

Si lag=1
paso de la serie x1,x2,x3,x4,...
a la serie: x2-x1,x3-x2.x4-x3,...

si diff es 1, me quedo con eso
si diff = 2
me quedo:
x3-2x2+x1,x4-2x3+x2,.x5-2x4+x3

Si lag = 8
paso de la normal a la
x9-x1,x10-x2,...

donde

- `x` es un vector o una matriz numérica con los valores que se desean diferenciar.
- `lag` es un entero que indica el retraso a usar, 1 por defecto.
- `differences` es un entero que indica el orden de las diferencias, 1 por defecto.
- `xi` es un vector o matriz con los valores iniciales de las series originales en cada diferenciación.

Por ejemplo:

```
s = diff(1:10, lag=2, dif=1)
diff(1:10, 2)
diffinv(s,lag=2,dif=1, xi = c(1,2))
x=(1:10)^2
diff(x, lag=1, dif=1)
diff(x, lag=1, dif=2)
```

Ejercicio 2.1 *1. Simular 100 valores de un proceso de ruido blanco.*

2. *Simular 100 valores de un proceso autorregresivo de orden 1 de parámetro 0.5.*
3. *Simular 100 valores de un proceso de media móvil de orden 1 de parámetro 0.6.*
4. *Simular 85 valores de un proceso ARMA(2,2) y de parámetros (0.75,-0.34) en la parte AR y (-0.3, 0.35) en la parte MA.*
5. *Simular 150 valores de un proceso ARIMA(1,1,1) de parámetros 0.75 en la parte AR y -0.65 en la parte MA.(para hacer después de ver el Tema 4)*
6. *En todos los casos, guardar los datos en un objeto de serie temporal, y realizar la gráfica correspondiente.*

Ejercicio 2.2 *Generar un proceso ARIMA no estacionario. Representarlo gráficamente. Convertirlo en un proceso estacionario con una transformación adecuada. (para hacer después de ver el Tema 4)*

Ejercicio 2.3 *Generar 100 observaciones de un modelo ARIMA(2,1,0), con parámetros* $\phi_1 = 0.8$ *y* $\phi_2 = 0.15$*. Para los valores de ruido blanco, usar una varianza de 0.16. Representarla. Diferenciar la serie anterior y volver a representarla.(para hacer después de ver el Tema 4)*

```
y = arima.sim(100, model=list(order=c(2,1,0),ar=c(0.8,0.15)),sd=0.4)
plot(y)
y = diff(y)
plot(y)
```

3 Funciones de autocorrelación, autocorrelación parcial y autocorrelación parcial extendida muestrales

Una vez que se tienen datos correspondientes a una serie temporal, tras su definición como serie temporal, y su representación gráfica como primer acercamiento a la misma, el análisis continúa con la estimación de las funciones de autocorrelación y autocorrelación parcial.

3.1 La función acf

Se usa para calcular (y por defecto representar) la función de autocovarianza o autocorrelación estimada. La sintaxis es la siguiente:

```
acf(x, lag.max = NULL, type = c("correlation", "covariance", "partial"),
    plot = TRUE, na.action=na.fail, ...)
```

donde

- `x` es una serie temporal univariante o multivariante, o un vector o una matriz con los valores de los datos.
- `lag.max` es el número máximo de retardos a estimar. Por defecto es $10\log(10n)$, siendo n el número de observaciones.
- `type` es el tipo de función a estimar (por defecto es `correlation`).
- `na.action`. Si se pone como valor `na.fail` no calcula la acf si la serie tiene valores perdidos. Este es el valor que tiene por defecto. Si se pone `na.pass` entonces calcula la acf ignorando los valores perdidos (puede que matriz de autocorrelaciones no sea definida positiva).

Ejemplo:

```
set.seed(1234)
sim.ar = arima.sim(list(ar=c(0.4,0.4)),n=1000)
sim.ma = arima.sim(list(ma=c(0.6,-0.4)),n=1000)
par(mfrow=c(2,1))
acf(sim.ar,main="ACF de un proceso AR(2)")
acf(sim.ma,main="ACF de un proceso MA(2)")
```

3.2 La función pacf

Se usa para calcular (y por defecto representar) la función de autocorrelación parcial estimada. Equivale a la anterior con `type="partial"`. La sintaxis es la siguiente:

```
pacf(x, lag.max, plot, na.action, ...)
```

Ejemplo (continuación):

```
pacf(sim.ar,main="PACF de un proceso AR(2)")
pacf(sim.ma,main="PACF de un proceso MA(2)")
?presidents
presidents # tiene  valores perdidos
acf(presidents)  #da mensaje de error
acf(presidents, na.action = na.pass)
pacf(presidents, na.action = na.pass)
```

3.3 La FACE muestral

En un modelo $ARMA(p,q)$, com $p \geq 1$ y $q \geq 1$ las FAS y FAP decrecen exponencialmente, pero no aportan información sobre p y q. En las clases de teoría se ha estudiado la FACE (función de autocorrelación extendida), útil para especificar los órdenes p y q. Para el cálculo de la versión muestral con `R`:

```
library(TSA)
?arma11.s       #datos ARMA(1,1)
data(arma11.s)  #cargamos este conjuto de
acf(arma11.s)
pacf(arma11.s)
eacf(arma11.s)  #FACE
#----------------------------------
# AR/MA
#   0 1 2 3 4 5 6 7 8 9 10 11 12 13
# 0 x x x x o o o o o o o  o  o  o
# 1 x o o o o o o o o o o  o  o  o
# 2 x o o o o o o o o o o  o  o  o
# 3 x x o o o o o o o o o  o  o  o
# 4 x o x o o o o o o o o  o  o  o
# 5 x o o o o o o o o o o  o  o  o
# 6 x o o o x o o o o o o  o  o  o
# 7 x o o o x o o o o o o  o  o  o
#----------------------------------
```

que sugiere examinar los modelos $ARMA(1,1)$ y $ARMA(2,1)$. De mirar la FAS y la FAP, parece que un $AR(1)$ podría se adecuado. Más tarde ajustamos estos tres modelos, y compararemos los ajustes.

4 Estimación de parámetros

Una vez que se tienen datos correspondientes a una serie temporal y se ha detectado el orden (p,d,q), se usa la función `arima` para estimar los parámetros del modelo.

4.1 La función arima

La forma simple de utilizar esta función es la siguiente:

```
arima(x, order = c(0L, 0L, 0L), seasonal = list(order = c(0L, 0L, 0L), period = NA))
```

donde

- `x` es una serie temporal univariante.
- `order` es la especificación del modelo no estacional (p,d,q). ($d = 0$ en un modelo ARMA(p,q))

- `seasonal` es la especificación del modelo estacional (P, D, Q) más el periodo (se estudiará en temas posteriores).

Salvo que se especifique lo contrario, para la estimación se incluye término independiente en el modelo, esto es, no se supone a priori que la media sea nula.

Nota: La componente `$residuals` proporciona los residuos del modelo

Ejemplo: Como vimos antes, para los datos `arma11.s` del paquete `TSA`, podríamos ajustar los modelos AR(1), ARMA(1,1) y ARMA(2,1)

```
(fit10=arima(arma11.s,order=c(1,0,0)))
(fit11=arima(arma11.s,order=c(1,0,1)))
(fit21=arima(arma11.s,order=c(2,0,1)))
```

Una vez que se tienen los parámetros estimados de un modelo ARIMA, se deben analizar los residuos para detectar cualquier indicio de falta de ajuste. La función `tsdiag` proporciona gráficos para la diagnosis del modelo ajustado.

4.2 La función tsdiag

La forma simple de utilizar esta función es la siguiente:

```
tsdiag(object, gof.lag, ...)
```

donde

- `object` es el modelo de serie temporal ajustado.
- `gof.lag` es el máximo número de retardos para el test de Ljung-Box

Ejemplo:

```
tsdiag(fit10)
tsdiag(fit11)
tsdiag(fit21)
```

El problema que presenta esta función es que para calcular los p-valores del test de Ljung-Box, ignora que se estén estimando parámetros: los calcula suponiendo m grados de libertad (m=num. de correlaciones en el hipótesis nula), y no $m - h$ (h=num de parámetros estimados=p (parte AR) + q (parte MA) + 1(para la media), que es lo correcto). La diferencia puede ser importante. Para que lo haga correctamente podemos usar la función `sarima` del paquete `astsa`, que ajusta el modelo y hace diagnosis, con sintaxis

```
sarima(x, p, d, q)
```

donde

- `x` es una serie temporal univariante.
- `order` es la especificación del modelo no estacional (p, d, q). ($d = 0$ en un modelo ARMA(p,q))
- También admite especificación para modelo estacional (se verá más adelante)

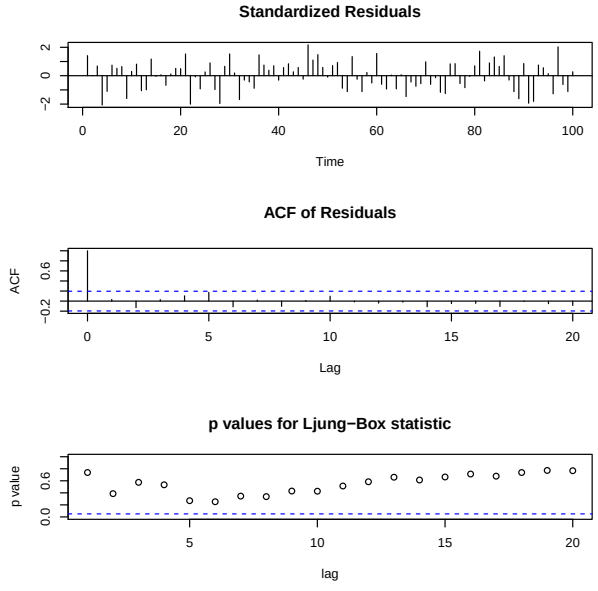
Salvo que se especifique lo contrario, para la estimación se incluye término independiente en el modelo, esto es, no se supone a priori que la media sea nula.

Veamos que existe diferencia:

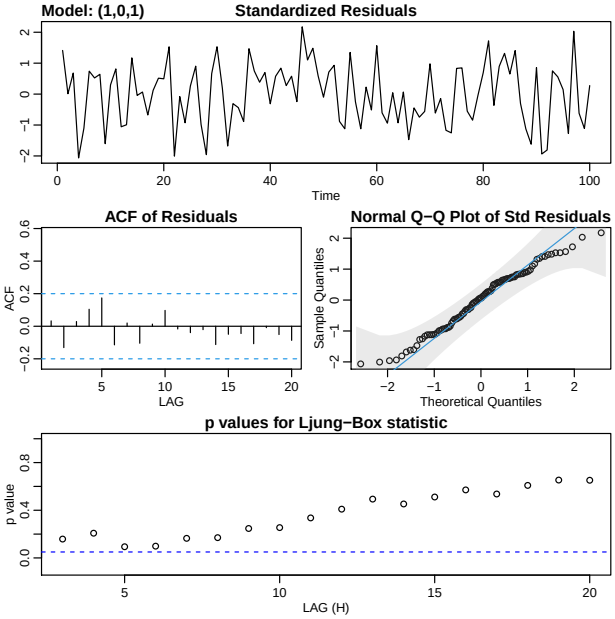
```
set.seed(12345)
x=arima.sim(list(ar=c(0.6, -0.7), ma=c(-0.8, 0.7)), n=100)
(aj1=arima(x,order=c(1,0,1)))
tsdiag(aj1, gof.lag=20)
(aj2=astsa::sarima(x,1,0,1))
```

En algunas librerías, se omite el término constante en el AIC, lo cual no afecta a la selección del modelo, ya que la constante es la misma para todos los modelos. Pero hay que tener cuidado al comparar el valor de AIC entre distintas librerías o entre clases de modelos, ya que pueden tratar el término constante de manera diferente y, por lo tanto, los valores de AIC no son comparables (por ejemplo en `tsdiag`, que toma número de parámetros $p + q + 1$ y `astsa::sarima`, que toma número de parámetros $p + q + 2$).

Gráficos usando `tsdiag`



Gráficos usando `sarima`



4.3 La función Box.test

Otra función útil para analizar la incorrelación de una serie es `Box.test`. Calcula el contraste de Box-Pierce y el de Ljung-Box. La forma simple de utilizar esta función es la siguiente:

```
Box.test(x, lag = 1, type = c("Box-Pierce", "Ljung-Box"), fitdf = 0))
```

donde

- `x` es un vector o una serie temporal univariante.
- `lag` número de autocorrelaciones a considerar.
- `fitdf` número de grados de libertad que se restarán si x es una serie de residuos (el número de paraámetros estimados, sin contrar σ^2).

Ejemplo:

```
x = rnorm (100)
Box.test (x, lag = 1)
Box.test (x, lag = 1, type = "Ljung")
```

Como ya se vio en clase de teoría, si la entrada son residuos, hay que restar el número de parámetros estimados para el cálculo de los residuos. Esto es importante, hay una gran diferencia si no se especifica `fitdf`: Ejemplo:

```
Box.test (aj1$residuals, lag = 7, type = "Ljung")
#  X-squared = 7.8539, df = 7, p-value = 0.3457
Box.test (aj1$residuals, lag = 7, type = "Ljung", fitdf=3)
# X-squared = 7.8539, df = 4, p-value = 0.09708  ##----- hay una gran diferencia!!

Box.test (aj1$residuals, lag = 5, type = "Ljung")
#  X-squared = 6.3938, df = 5, p-value = 0.2698
Box.test (aj1$residuals, lag = 5, type = "Ljung", fitdf=3)
#X-squared = 6.3938, df = 2, p-value = 0.04089   ##----- hay una gran diferencia!!
```

5 Predicción en modelos ARIMA

Una vez identificado un modelo y estimados los parámetros correspondientes, se pueden efectuar predicciones.

5.1 La función predict

Se usa para predecir. La función proporciona la predicción y estimación de su error estándar. A partir de estos valores pueden calcularce intervalos de confianza (asintóticos): predición $\mp$ 1.96 $\times$ error estándar.

Ejemplo:

```
?LakeHuron
LakeHuron
plot(LakeHuron,xlim=c(1875,1980),ylim=c(575,584))
(fit = arima(LakeHuron,order=c(1,0,1)))
(LH.pred = predict(fit,n.ahead=8))
lines(LH.pred$pred,col="red")
```

Ejercicio 5.1 *1. Generar un modelo ARMA(2,0) con 1200 observaciones y parámetros:* 0.4, −0.5 *(fijar* `set.seed(12345)`*).*

2. Realizar un ajuste por distintos modelos mirando las funciones de autocorrelación muestrales.

3. Estimar sus parámetros.

4. Predecir los próximos 24 instantes, y calcular intervalos de confianza para las predicciones.

5. Realizar las gráficas correspondientes.

6 Uso del paquete forecast

Existen varios paquetes que se pueden usar en el análisis de series temporales. Consultar

`https://cran.r-project.org/web/views/TimeSeries.html`

para un breve resumen actualizado de los mismos. Uno de los paquetes más destacables para series univariantes mediante modelos ARIMA es **forecast**. Es una de las principales herramientas usadas para la predicción de series temporales, gracias a las funciones que incorpora.

- La función `Arima()` permite ajustar un modelo dado.
- La función `auto.arima()` permite ajustar mediante un algoritmo automático basado el criterio de AIC para la selección de los órdenes (p, d, q), así como la estimación del modelo.
- La función `checkresiduals` hace diganosis del modelo
- La función `forecast(modelo,h=número de periodos a predecir)` es una función genérica que permite realizar predicciones de series temporales o de modelos de series temporales.

Veremos un ejemplo de aplicación de estas funciones, aplicándolas a WWWusage, una serie de tiempo con los datos del número de usuarios conectado a un servidor cada minuto (de longitud 100), para ajustar un modelo arima adecuado, predecir los siguientes 20 valores, incluidos intervalos de confianza, y representarlos.

```
set.seed(12345)
x=arima.sim(list(ar=c(0.6, -0.7), ma=c(-0.8)), n=200)

install.packages('forecast', dependencies = TRUE)
library(forecast)
?auto.arima
?forecast
(fit = auto.arima(x, stepwise=FALSE))
#fit = auto.arima(x, stepwise=FALSE, allowmean=TRUE) para que incluya media
autoplot(fit)
checkresiduals(fit)
(prediccion=forecast(fit,h=20))
plot(prediccion)
```