Séries Temporais

Guia Prático

Luís Torgo
FEP, Universidade do Porto
ltorgo@liacc.up.pt

4 de Dezembro de 2004

Este guia tem como objectivo ajudar os(as) alunos(as) na utilização de métodos para lidar com séries temporais usando o R. O código fornecido deve ser visto como exemplificativo de algumas das possibilidades do R, devendo os(as) alunos(as) explorar em detalhe as funções usadas (por exemplo através do seu *help*) de modo a explorar mais possibilidades.

1. Introdução

O R tem várias packages dedicadas às séries temporais, bem como vários tipos de objectos que podem ser usados para guardar dados referentes a séries temporais, i.e. dados etiquetados por um índice temporal.

Quanto às packages uam das principais é a package *tseries*, mas existem outras como a *zoo*, a *dse* ou a *its*.

Quanto a objectos temos por exemplo os objectos da classe *ts* que podem ser usados para representar séries temporais regulares, isto é em que as observações têm um espaçamento temporal regular. A package *its* fornece uma classe de objectos *its* que pode ser usada para séries temporais irregulares. Existem outras alternativas, mas estas são suficientes para a maioria das aplicações.

Nos nossos primeiros exemplos vamos usar os dados `lh` que vêm com o R e que contêm informação sobre uma série temporal regular com observações de um parâmetro médico tiradas de 10 em 10 minutos. O objecto `lh` é um objecto da classe *ts*.

```
> data(lh)
> lh

Time Series:
Start = 1
End = 48
Frequency = 1
 [1] 2.4 2.4 2.4 2.2 2.1 1.5 2.3 2.3 2.5 2.0 1.9 1.7 2.2 1.8 3.2 3.2 2.7 2.2 2.2
[20] 1.9 1.9 1.8 2.7 3.0 2.3 2.0 2.0 2.9 2.9 2.7 2.7 2.3 2.6 2.4 1.8 1.7 1.5 1.4
[39] 2.1 3.3 3.5 3.5 3.1 2.6 2.1 3.4 3.0 2.9

> class(lh)

[1] "ts"
```

A vantagem de guardarmos estes 48 valores da série temporal num objecto da classe *ts* em vez de um vector "normal", permite-nos tirar partido de uma série de funções específicas para esta classe de objectos, como veremos mais à frente. Podemos criar um objecto desta classe usando a função `ts`,

```
> serieTrimestral <- ts(rnorm(100), frequency = 4, start = c(1990,
+     2))
> serieTrimestral
```

	Qtr1	Qtr2	Qtr3	Qtr4
1990		0.198994084	-0.287353738	-0.308518525
1991	-1.822750886	-1.832586025	1.723030537	-0.547756207
1992	-0.114100861	0.204537997	0.340972799	0.569278662
1993	0.939522401	-0.166237043	-1.305210829	-0.541354647
1994	1.204113898	-0.186450301	-0.812261805	-0.268842485
1995	-0.059433378	-0.514851317	0.061202781	-1.819350416
1996	-0.358846766	-1.479256408	-0.112766691	-0.597066957
1997	0.339561674	0.641215445	-0.927529269	0.455507265
1998	-0.296610091	-1.444170734	-1.266819991	-0.313269043
1999	1.225562559	1.285878003	-1.738930255	-1.464531013
2000	0.945055607	-0.302626762	0.911366713	-0.654782368
2001	0.973264469	0.214448209	-0.631249714	1.099778865
2002	-0.670338593	-0.083220889	-0.579308396	0.874672521
2003	0.701950559	0.819745753	-0.454595288	0.118456528
2004	-0.412679157	1.040792805	0.867438811	0.978256811
2005	1.557810178	0.236784652	0.471325514	-2.104503598
2006	0.973790533	0.161047174	1.405673157	-1.676146306
2007	-0.829300487	-1.491558817	0.562186583	-0.925580143
2008	0.219994409	-0.998867533	-0.406945926	-1.997324977
2009	0.917731173	-2.250084428	-0.033389838	0.144387929
2010	0.228724029	0.227546138	-1.155702863	0.007498215
2011	-0.698210150	-0.924512723	-1.302742351	-1.623060477
2012	1.067888984	2.043213102	-0.183303222	0.185938852
2013	-0.145496635	2.514047522	0.664265985	-0.459049958
2014	0.241312234	0.541548621	-1.186211324	-1.737168713
2015	-0.624905980			

```
> serieDiaria <- ts(rnorm(20), frequency = 7, end = c(10, 14))
> serieDiaria

Time Series:
Start = c(9, 2)
End = c(11, 7)
Frequency = 7
 [1] -0.34080964 -1.51410810 -0.00350895  1.14357209  0.15492171 -0.98446356
 [7] -0.01977488  0.33116825 -0.30602755  0.92906097  1.90848637 -0.99508968
[13] -0.38791141  0.46819920  0.22490905 -0.85324916  1.13903581 -0.48520146
[19] -0.66445648 -2.29096731

> serieMensal <- ts(rnorm(30), frequency = 12, start = c(2000,
+     5))
> serieMensal

            Jan        Feb        Mar        Apr        May        Jun
2000                                                1.9230907  1.2708876
2001  0.7450162  0.3665809  0.9577726 -0.8182036 -1.1480757  0.3841645
2002 -0.3831805  0.7902960 -0.6055760  0.3910736  0.3804827 -1.6555623
            Jul        Aug        Sep        Oct        Nov        Dec
2000  0.2941687  1.0037458 -2.6876183  0.9961709  0.4276739 -0.5190493
2001  0.6053062  2.1036941 -1.4198743  0.1171551 -1.2766155 -0.0304629
2002 -0.6856561 -0.7621952 -0.6048381 -0.2304250
```

Repare como o R tem formas diferentes de mostrar as séries temporais dependendo da sua periodicidade.

Como foi mencionado o R tem diversas funções que se comportam de forma diferente em face de objectos da classe *ts*. Começemos pela representação gráfica destes objectos (Figura 1),

```
> plot(serieTrimestral)
```

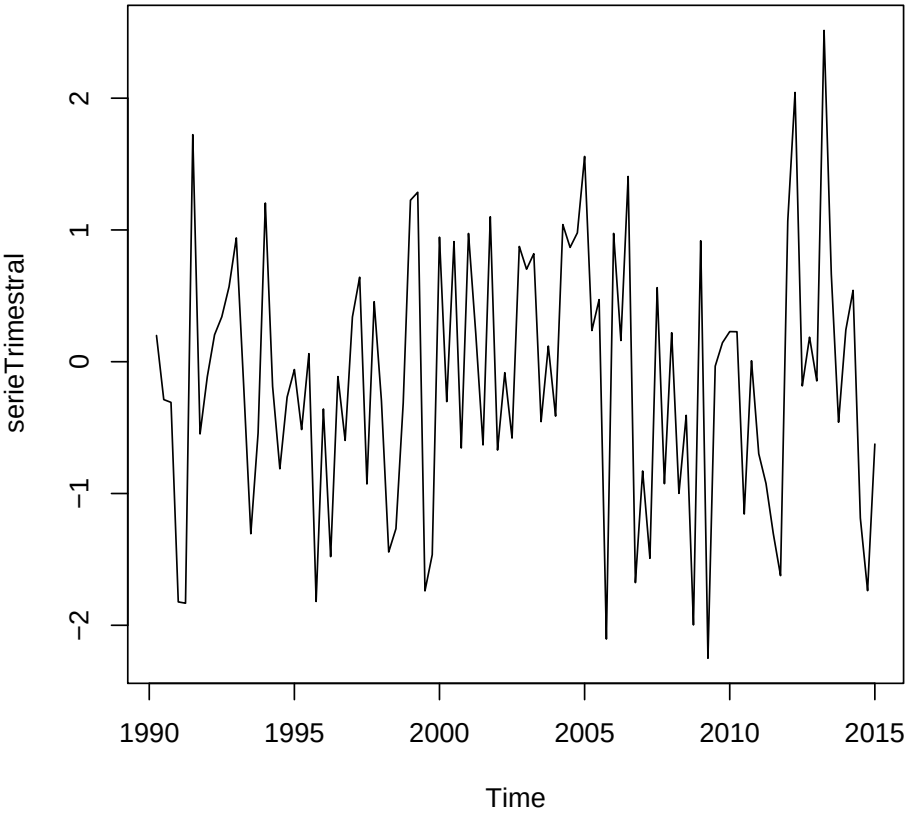


Figura 1: Uma série temporal trimestral.

Vejamos agora algumas funções úteis para manipulação de séries temporais,

```
> window(serieTrimestral, start = c(1995, 2), end = c(1999, 4))

           Qtr1        Qtr2        Qtr3        Qtr4
1995             -0.51485132  0.06120278 -1.81935042
1996 -0.35884677 -1.47925641 -0.11276669 -0.59706696
1997  0.33956167  0.64121545 -0.92752927  0.45550727
1998 -0.29661009 -1.44417073 -1.26681999 -0.31326904
1999  1.22556256  1.28587800 -1.73893025 -1.46453101

> diff(window(serieTrimestral, start = c(1995, 2), end = c(1999,
+     4)))

           Qtr1        Qtr2        Qtr3        Qtr4
1995                           0.57605410 -1.88055320
1996  1.46050365 -1.12040964  1.36648972 -0.48430027
1997  0.93662863  0.30165377 -1.56874471  1.38303653
1998 -0.75211736 -1.14756064  0.17735074  0.95355095
1999  1.53883160  0.06031544 -3.02480826  0.27439924
```

Procure entender o resultado destes dois exemplos.
A função `acf` permite-nos visualizar (Figura 2) a auto-correlação de uma série temporal,

```
> acf(lh)
```

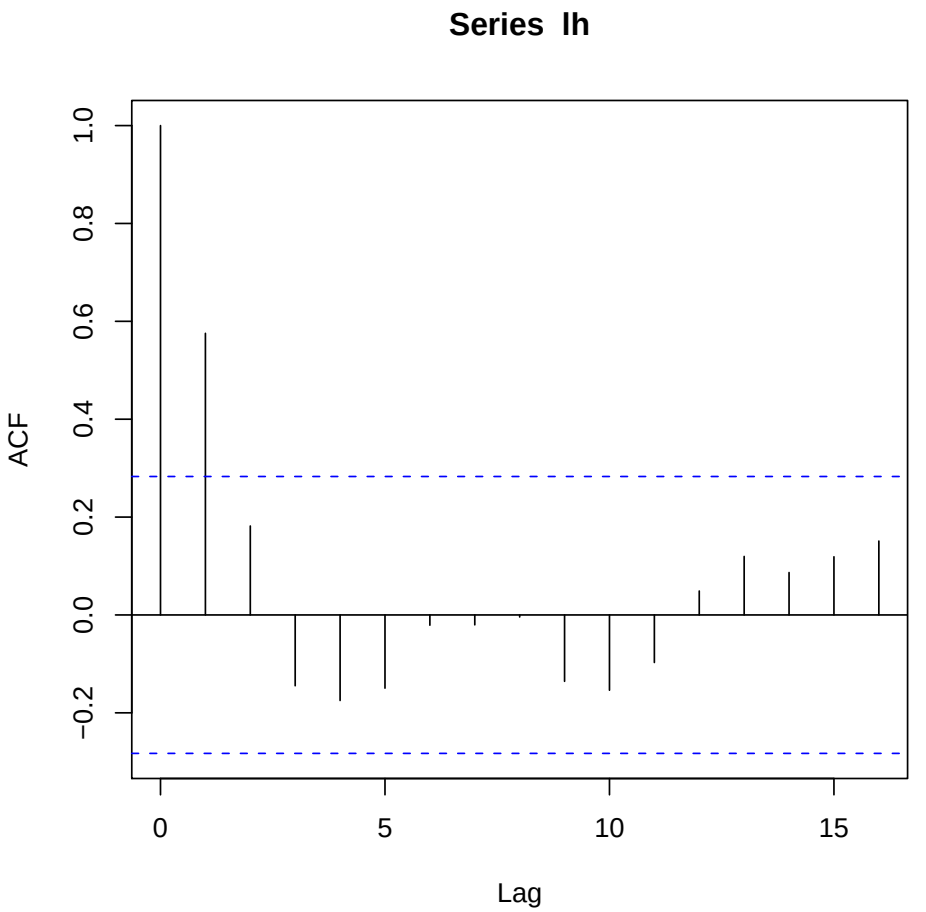


Figura 2: Os valores da auto-correlação para diferentes *lags*.

2. Modelos Clássicos

Um dos modelos mais simples que podemos usar para tentar aproximar uma série temporal é uma média móvel dos seus últimos k valores. A função `filter` pode ser usada para obter este tipo de modelos,

```
> lh

Time Series:
Start = 1
End = 48
Frequency = 1
 [1] 2.4 2.4 2.4 2.2 2.1 1.5 2.3 2.3 2.5 2.0 1.9 1.7 2.2 1.8 3.2 3.2 2.7 2.2 2.2
[20] 1.9 1.9 1.8 2.7 3.0 2.3 2.0 2.0 2.9 2.9 2.7 2.7 2.3 2.6 2.4 1.8 1.7 1.5 1.4
[39] 2.1 3.3 3.5 3.5 3.1 2.6 2.1 3.4 3.0 2.9

> mm4 <- filter(lh, rep(1/4, 4), sides = 1)
> mm4

Time Series:
Start = 1
End = 48
Frequency = 1
 [1]    NA    NA    NA 2.350 2.275 2.050 2.025 2.050 2.150 2.275 2.175 2.025
[13] 1.950 1.900 2.225 2.600 2.725 2.825 2.575 2.250 2.050 1.950 2.075 2.350
[25] 2.450 2.500 2.325 2.300 2.450 2.625 2.800 2.650 2.575 2.500 2.275 2.125
[37] 1.850 1.600 1.675 2.075 2.575 3.100 3.350 3.175 2.825 2.800 2.775 2.850
```

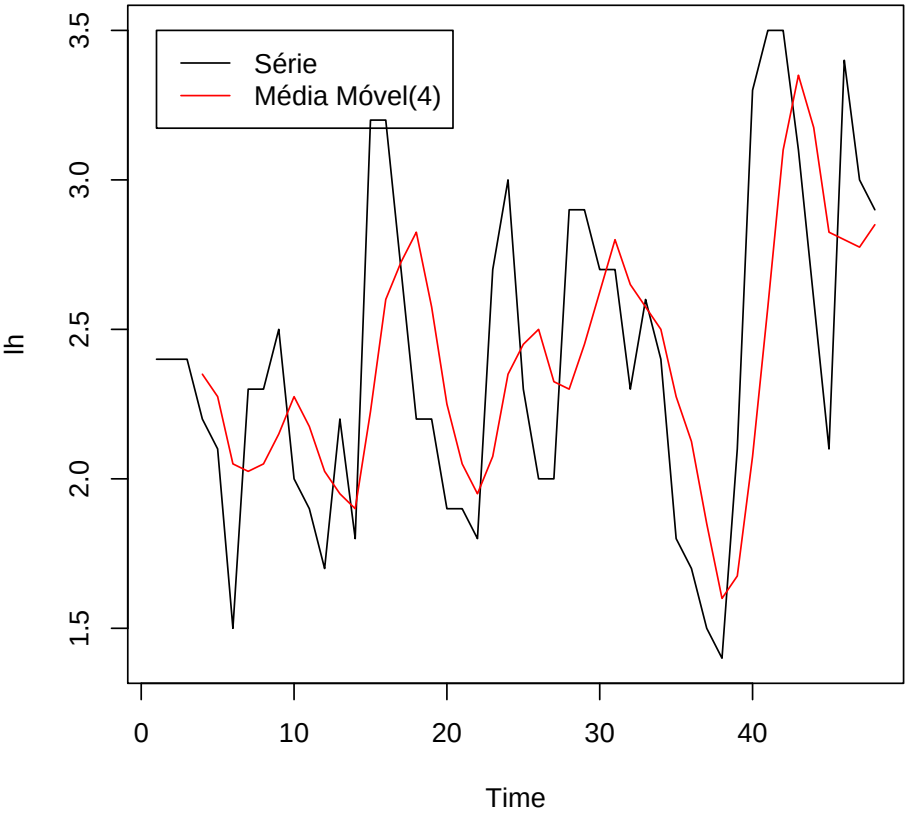
As instruções acima obtêm uma média móvel usando os 4 últimos valores da série. A função `filter` na realidade serve para muitas outras coisas, e daí a aparente complexidade da sua utilização. Poderiamos criar uma função para ocultar isto do utilizador "comum" se quisessemos,

```
> mm <- function(s, k) {
+     filter(s, rep(1/k, k), sides = 1)
+ }
> mm(lh, 5)

Time Series:
Start = 1
End = 48
Frequency = 1
 [1]   NA   NA   NA   NA 2.30 2.12 2.10 2.08 2.14 2.12 2.20 2.08 2.06 1.92 2.16
[16] 2.42 2.62 2.62 2.70 2.44 2.18 2.00 2.10 2.26 2.34 2.36 2.40 2.44 2.42 2.50
[31] 2.64 2.70 2.64 2.54 2.36 2.16 2.00 1.76 1.70 2.00 2.36 2.76 3.10 3.20 2.96
[46] 2.94 2.84 2.80
```

Podemos ver o comportamento da média móvel na Figura 3,

```
> plot(lh)
> lines(mm(lh, 4), col = "red")
> legend(1, 3.5, c("Série", "Média Móvel(4)"), col = c("black",
+     "red"), lty = c(1, 1))
```



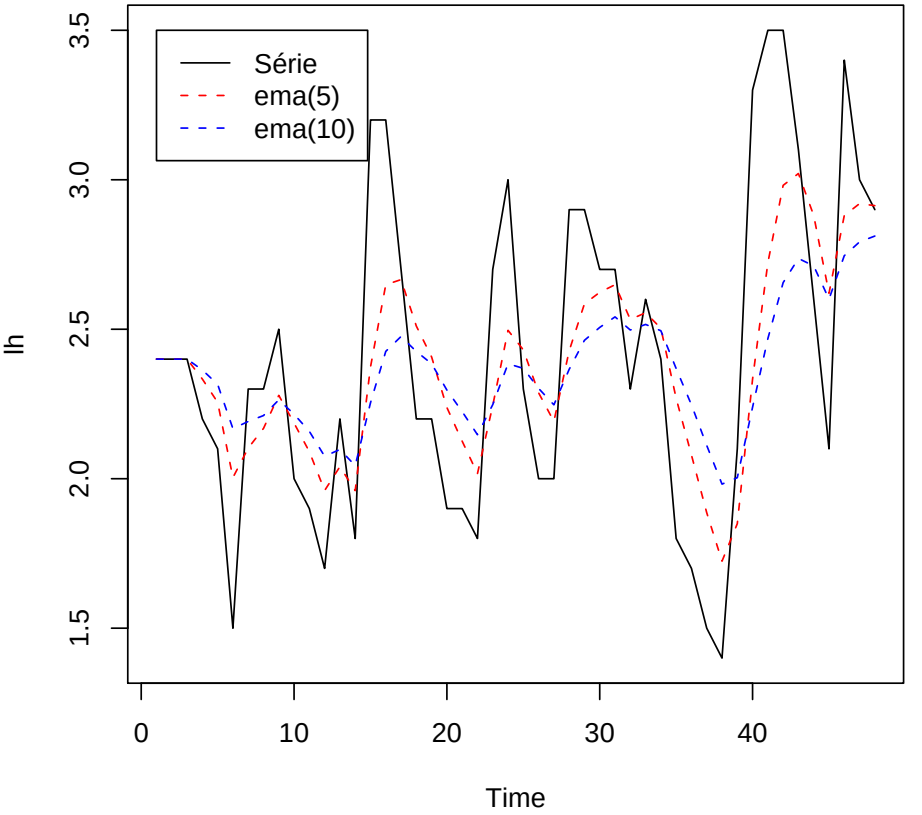
A função `filter` também pode ser usada para obter médias exponenciais,

```
> ema <- function(x, k, init = x[1]) {
+     alfa <- 2/(k + 1)
+     filter(alfa * x, filter = 1 - alfa, method = "recursive",
+         init = init)
+ }
> ema(lh, 4)

Time Series:
Start = 1
End = 48
Frequency = 1
 [1] 2.400000 2.400000 2.400000 2.320000 2.232000 1.939200 2.083520 2.170112
 [9] 2.302067 2.181240 2.068744 1.921247 2.032748 1.939649 2.443789 2.746274
[17] 2.727764 2.516658 2.389995 2.193997 2.076398 1.965839 2.259503 2.555702
[25] 2.453421 2.272053 2.163232 2.457939 2.634763 2.660858 2.676515 2.525909
[33] 2.555545 2.493327 2.215996 2.009598 1.805759 1.643455 1.826073 2.415644
[41] 2.849386 3.109632 3.105779 2.903467 2.582080 2.909248 2.945549 2.927329
```

Podemos visualizar algumas médias exponenciais na Figura 4,

```
> plot(lh)
> lines(ema(lh, 5), lty = 2, col = "red")
> lines(ema(lh, 10), lty = 2, col = "blue")
> legend(1, 3.5, c("Série", "ema(5)", "ema(10)"), col = c("black",
+     "red", "blue"), lty = c(1, 2, 2))
```



Modelos auto regressivos podem ser obtidos com a função `ar`. Esta função pode obter um modelo de uma ordem específica ou estimar a melhor ordem usando o critério AIC. Vejamos exemplos de ambas as situações,

```
> ar(lh, order.max = 5)

Call:
ar(x = lh, order.max = 5)

Coefficients:
      1        2        3
 0.6534  -0.0636  -0.2269

Order selected 3  sigma^2 estimated as  0.1959

> ar(lh, aic = F, order.max = 4)

Call:
ar(x = lh, aic = F, order.max = 4)

Coefficients:
      1        2        3        4
 0.6767  -0.0571  -0.2941   0.1028

Order selected 4  sigma^2 estimated as  0.1983
```

Na primeira utilização nós obtivemos um modelo AR com a ordem seleccionada automáticamente pelo critério AIC, enquanto que na segunda chamada obtivemos um modelo $AR(4)$, isto é não é feita qualquer selecção pelo critério AIC.

Também podemos usar a função `arima` para obter modelos AR. De facto, a função `arima` obtêm modelos $ARIMA(p, d, q)$, pelo que basta fazer $d = q = 0$ para obter um modelo $AR(p)$ usando esta função.

Vejamos então um exemplo de obtenção de um modelo $ARIMA(3, 1, 2)$,

```
> arima(lh, c(3, 1, 2))

Call:
arima(x = lh, order = c(3, 1, 2))

Coefficients:
          ar1     ar2      ar3      ma1      ma2
      -0.0755  0.4064  -0.3601  -0.1580  -0.7311
s.e.   0.1994  0.1563   0.1426   0.1893   0.1783

sigma^2 estimated as 0.18:  log likelihood = -27.37,  aic = 66.75
```

Qualquer uma das duas funções anteriores produzem modelos que podem ser usados para obter previsões futuras dos valores de uma série temporal. De facto, este tipo de objectos têm também uma função `predict` associada. Na Figura 5 vemos exemplos da sua utilização,

```
> ar4 <- ar(lh, aic = F, order.max = 4)
> arima312 <- arima(lh, c(3, 1, 2))
> preds.ar4 <- predict(ar4, n.ahead = 5)$pred
> preds.arima312 <- predict(arima312, n.ahead = 5)$pred
> plot(c(lh, preds.ar4), type = "n")
> lines(lh)
> lines(preds.ar4, col = "red")
> lines(preds.arima312, col = "blue")
> legend(1, 3.5, c("Série", "Previsões AR(4)", "Previsões ARIMA(3,1,2)"),
+     lty = c(1, 1, 1), col = c("black", "red", "blue"))
```

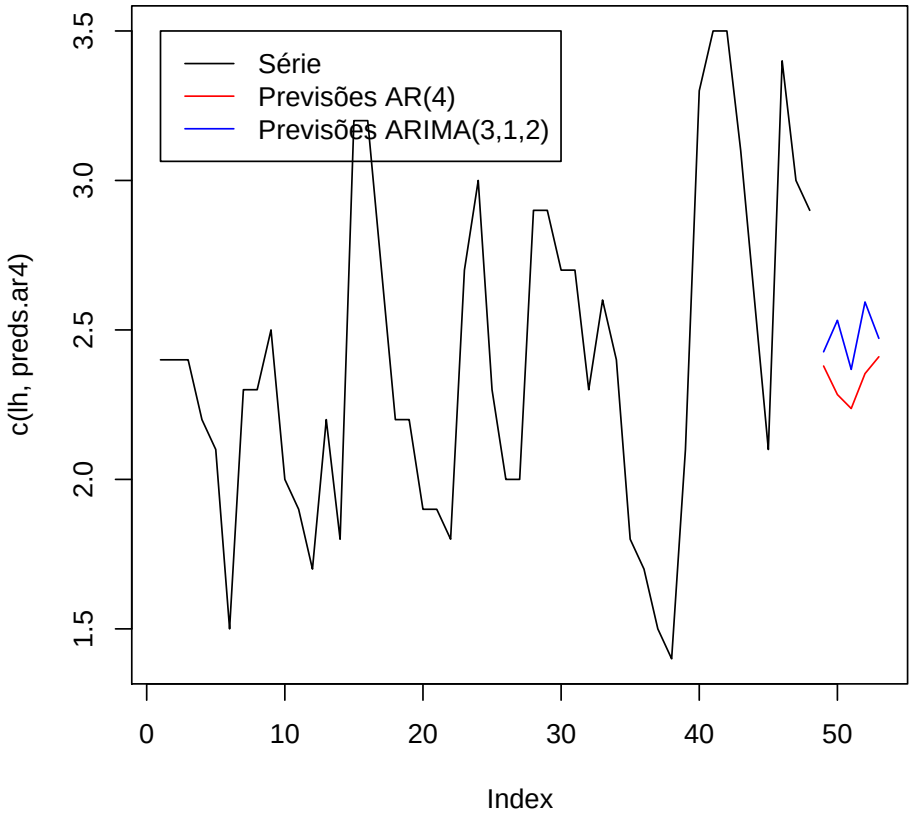


Figura 5: A série `lh` e as previsões para os próximos 5 valores de um modelo AR(4) e de um modelo ARIMA(3,1,2).

Terminamos a nossa exemplificação de modelos clássicos para previsão de séries temporais com o modelo Holt Winters. Vamos exemplificar o uso da função `HoltWinters` com uma outra série temporal que vêm com o R,

```
> (HW <- HoltWinters(AirPassengers))

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

Call:
 HoltWinters(x = AirPassengers)

Smoothing parameters:
 alpha:  0.2437152
 beta :  0.01922789
 gamma:  1

Coefficients:
          [,1]
a   479.163460
b     2.891389
s1  -29.289686
s2  -56.480957
s3  -22.084757
s4   11.063521
s5   17.054786
s6   73.525007
s7  151.243765
s8  133.033523
s9   32.276189
s10 -19.760979
s11 -89.131583
s12 -47.163460
```

A função quando executada deste modo estima os parâmetros do modelo Holt Winters. No entanto, é obviamente possível especificar valores específicos para estes parâmetros (veja o Help da função). Podemos ver o ajuste do modelo à série na Figura 6,

```
> plot(HW)
```

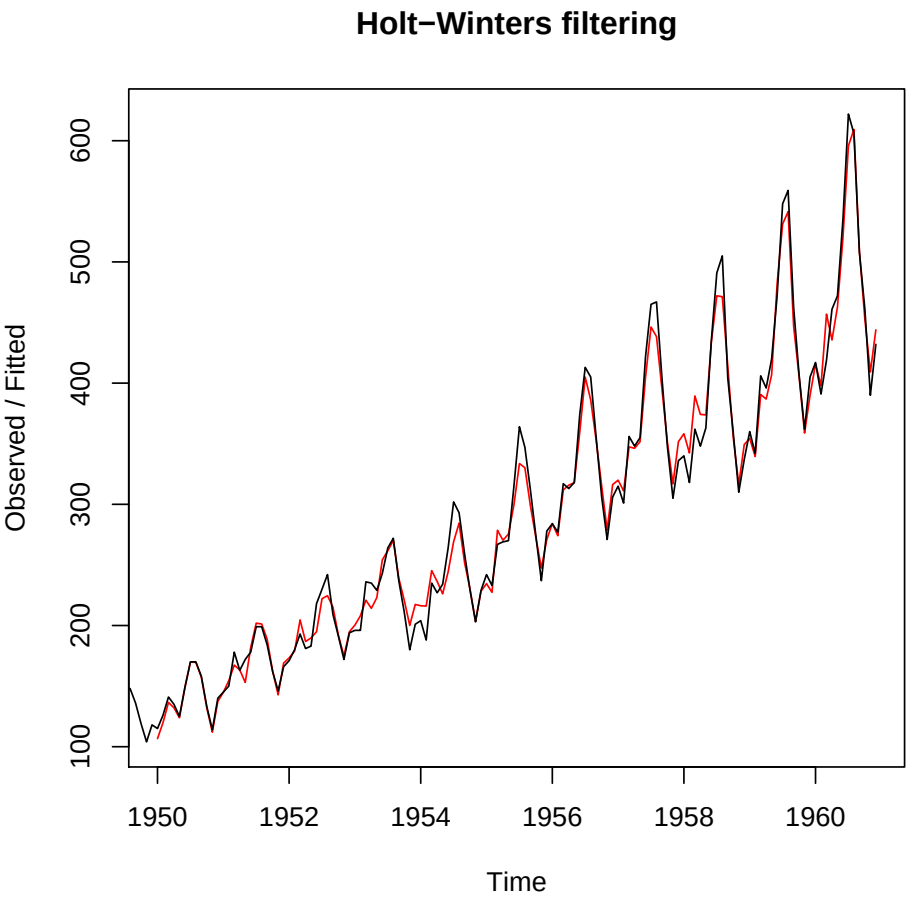


Figura 6: A série `AirPassengers` e o resultado de aplicar um modelo Holt Winters.

3. Time Delay Embedding

Nesta secção vamos explorar a possibilidade de usar técnicas de *time delay embedding* de modo a podermos transformar o problema de previsão de séries temporais num problema de regressão múltipla e assim podermos usar alguns dos modelos anteriormente explorados nesta cadeira.

Vamos usar nas nossas experiências a série `AirPassengers`. Começemos por dividir esta série em vários blocos para levar a cabo a estimação dos modelos e depois a posterior faze de teste,

```
> t.train <- window(AirPassengers, end = c(1955, 12))
> t.modsel <- window(AirPassengers, start = c(1956, 1), end = c(1957,
+     12))
> t.test <- window(AirPassengers, start = c(1958, 1))
> t.train.best <- window(AirPassengers, end = c(1957, 12))
```

A função `embed` permite obter *time delay embedding* de uma qualqeur dimensão de um série temporal. Veja o seguinte exemplo ilustrativo,

```
> lh

Time Series:
Start = 1
End = 48
Frequency = 1
 [1] 2.4 2.4 2.4 2.2 2.1 1.5 2.3 2.3 2.5 2.0 1.9 1.7 2.2 1.8 3.2 3.2 2.7 2.2 2.2
[20] 1.9 1.9 1.8 2.7 3.0 2.3 2.0 2.0 2.9 2.9 2.7 2.7 2.3 2.6 2.4 1.8 1.7 1.5 1.4
[39] 2.1 3.3 3.5 3.5 3.1 2.6 2.1 3.4 3.0 2.9

> head(embed(lh, dim = 4))

     [,1] [,2] [,3] [,4]
[1,]  2.2  2.4  2.4  2.4
[2,]  2.1  2.2  2.4  2.4
[3,]  1.5  2.1  2.2  2.4
[4,]  2.3  1.5  2.1  2.2
[5,]  2.3  2.3  1.5  2.1
[6,]  2.5  2.3  2.3  1.5
```

Para além disso, como já vimos, a função `diff` permite-nos obter diferenças dos valores de uma série. Podemos assim construír uma amostra para treino usando um *embed* de diferenças (de modo a eliminar efeitos nocivos de tendências),

```
> train <- data.frame(embed(diff(t.train), dim = 6))
> names(train) <- c("dt", "dt.1", "dt.2", "dt.3", "dt.4", "dt.5")
> head(train)

   dt dt.1 dt.2 dt.3 dt.4 dt.5
1  13   14   -8   -3   14    6
2   0   13   14   -8   -3   14
3 -12    0   13   14   -8   -3
4 -17  -12    0   13   14   -8
5 -15  -17  -12    0   13   14
6  14  -15  -17  -12    0   13

> sel <- data.frame(embed(diff(t.modsel), dim = 6))
> test <- data.frame(embed(diff(t.test), dim = 6))
> train.best <- data.frame(embed(diff(t.train.best), dim = 6))
> names(sel) <- names(test) <- names(train.best) <- c("dt", "dt.1",
+     "dt.2", "dt.3", "dt.4", "dt.5")
```

Ficamos assim com 3 data frames que podem ser usados para obter modelos, seleccionar alternativas e depois testar a melhor alternativa num conjunto separado de teste.

Vamos então tentar vários tipos de redes neuronais nestes dados, escolhendo a melhor usando a estatística de Theil,

```
> library(nnet)
> alt <- expand.grid(size = c(10, 20), decay = c(0.01, 0.001),
+     theil = 0)
> prevs.ant <- c(train[nrow(train), "dt"], sel[1:(nrow(sel) - 1),
+     "dt"])
> for (a in 1:nrow(alt)) {
+     nn <- nnet(dt ~ ., train, size = alt[a, "size"], decay = alt[a,
+         "decay"], maxit = 1000, linout = T)
+     prevs <- predict(nn, sel)
+     alt[a, "theil"] <- sqrt(sum(((sel[, "dt"] - prevs)/prevs.ant)^2)/sum(((sel[,
+         "dt"] - prevs.ant)/prevs.ant)^2))
+ }
> best <- which.min(alt[, "theil"])
```

Depois de termos estimado qual a melhor rede neuronal usando a amostra separada para selecção de modelos, podemos agora usar a amostra total para treino para obter a melhor rede, antes de a aplicarmos ao conjunto de teste,

```
> nn <- nnet(dt ~ ., train.best, size = alt[best, "size"], decay = alt[best,
+     "decay"], maxit = 1000, linout = T)
> prevs.ant <- c(train.best[nrow(train.best), "dt"], test[1:(nrow(test) -
+     1), "dt"])
> prevs <- predict(nn, test)
> theil <- sqrt(sum(((test[, "dt"] - prevs)/prevs.ant)^2)/sum(((test[,
+     "dt"] - prevs.ant)/prevs.ant)^2))
```