Metodologia Box & Jenkins utilizando o *software* R

Tainan Boff

1º de novembro de 2016

Abstract

Este documento foi elaborado para os alunos da disciplina ECO02007 - Econometria Aplicada, ministrada pelo prof. Sabino da Silva Pôrto Júnior na Universidade Federal do Rio Grande do Sul, e tem como objetivo apresentar algumas funções úteis para a análise de séries temporais através da metodologia Box & Jenkins utilizando o *software* R.

Contents

1 Introdução

Este documento tem como objetivo apresentar exemplos de aplicação da metodologia Box & Jenkins para séries temporais univariadas. Com este intuito, adotaremos os seguintes passos:

1. Carregar os dados no R;
2. Visualizar o gráfico da série temporal a fim de identificar padrões. Nesta etapa já é possível termos uma boa "pista" sobre a estacionariedade da série;
3. Realizar testes de raiz unitária para verificar a necessidade de diferenciação;
4. Caso a série seja não estacionária, tomar a primeira diferença (pode ser necessário tomar diferenças sucessivas até que a série se torne estacionária);
5. Se necessário, utilizar uma transformação Box-Cox para estabilizar a variância;
6. Examinar FAC e FACP para certificar-se de que a série é estacionária e identificar as ordens de dependência (AR(p), MA(q), ARMA(p,q));
7. Estimar os modelos escolhidos e utilizar critérios de informação (AIC, BIC) para escolher o melhor deles;
8. Checar os resíduos do modelo final: eles devem ser ruído branco. Pode-se visualizar o gráfico dos resíduos em busca de algum padrão, examinar a FAC e realizar um teste Portemanteau (ex.: Ljung-Box);
9. Se os resíduos forem ruído branco, fazer previsão.

2 Dados: simulação e *download*

Neste documento, iremos analisar sete séries de tempo diferentes: quatro delas serão simuladas - ou seja, utilizaremos o R para criar observações que correspondam a modelos previamente definidos - e três delas serão séries de tempo reais.

Para simularmos valores de séries temporais, utilizamos a função `arima.sim()`, inserindo como argumentos da função o modelo desejado e o número de observações. O modelo consiste um uma lista contendo os coeficientes dos componentes autorregressivos e os coeficientes dos componentes de médias móveis. No caso de um modelo ARIMA, é preciso inserir a ordem do modelo, como pode ser visto no código abaixo:

```
set.seed(2007)
ar2 <- arima.sim(model = list(ar=c(0.9,-0.2)), n=1000)
ma2 <- arima.sim(model = list(ma=c(0.5,-0.4)), n=1000)
arma11 <- arima.sim(model = list(ar=0.6, ma=0.7), n=1000)
arima111 <- arima.sim(model = list(order=c(1,1,1),ar=0.6, ma=0.7), n=1000)
```

Na primeira linha do código acima, escrevemos `set.seed(2007)`. O número 2007 é o que chamamos de "semente aleatória": é um número utilizado para iniciar o algoritmo gerador de números pseudo-aleatórios e foi incluído aqui para que este exemplo possa ser reproduzido (basta utilizar o mesmo número).

Além das quatro séries simuladas, utilizaremos três séries de tempo reais:

- A taxa de câmbio BRL/USD diária entre 30/09/11 e 07/10/2016, totalizando 1260 observações;
- O índice BOVESPA diário entre 30/09/11 e 05/10/16, totalizando 1239 observações;
- O PIB trimestral do Brasil entre 1995-1º e 2015-1º, totalizando 81 observações.

Para facilitar a reprodução dos exemplos, buscamos os dados no site https://www.quandl.com e utilizamos a função `Quandl()` para fazer o *download*. Para isso, é preciso instalar e carregar o pacote `Quandl` no R:

```
install.packages("Quandl")
```

```
library(Quandl)
```

Talvez a maneira mais simples de utilizar a função `Quandl()` seja buscar os dados diretamente no site (escolhendo o período e a frequência desejados) e simplesmente copiar o código exibido em "Export Data" - "R".

Ao fazermos o *download* das séries de tempo mencionadas, os dados são organizados como um *data frame* com duas colunas: uma contendo as datas e outra contendo os valores (a estes dados atribuímos os nomes de TXCAMBIO, BVSP e PIBTRIM). Porém, como pode ser visto no código abaixo, fazemos uma transformação a fim de obtermos os valores indexados pelas datas, em formato de série temporal, o que irá facilitar a análise a seguir (a estes dados atribuímos os nomes txcambio, bvsp e pibtrim).

```
TXCAMBIO <- Quandl("FED/RXI_N_B_BZ", api_key="_Wrzxx_yzGhfkVoJmU7s",
  start_date="2011-09-30")
TXCAMBIO$Date <- as.Date(as.character(TXCAMBIO$Date),format="%Y-%m-%d")
txcambio <- xts(TXCAMBIO$Value, TXCAMBIO$Date)

BVSP <- Quandl("BCB/7", api_key="_Wrzxx_yzGhfkVoJmU7s", start_date="2011-09-30",
  end_date="2016-10-05")
BVSP$Date <- as.Date(as.character(BVSP$Date),format="%Y-%m-%d")
```

```
bvsp <- xts(BVSP$Value, BVSP$Date)

PIBTRIM <- Quandl("IBGE/ST17_BR_BRASIL_ABS", api_key="_Wrzxx_yzGhfkVoJmU7s",
  start_date="1995-03-30", end_date="2015-06-29")
PIBTRIM$Date <- as.Date(as.character(PIBTRIM$Date),format="%Y-%m-%d")
pibtrim <- xts(PIBTRIM$Value, PIBTRIM$Date)
```

3 Análise das séries temporais

3.1 Estacionariedade

A estacionariedade é condição prévia para a modelagem de uma série temporal de acordo com a metodologia de Box e Jenkins (ARIMA). Uma série temporal fracamente estacionária é aquela em que a média e a variância não mudam ao longo do tempo e a autocovariância não depende do tempo, mas apenas da distância temporal entre as observacões.

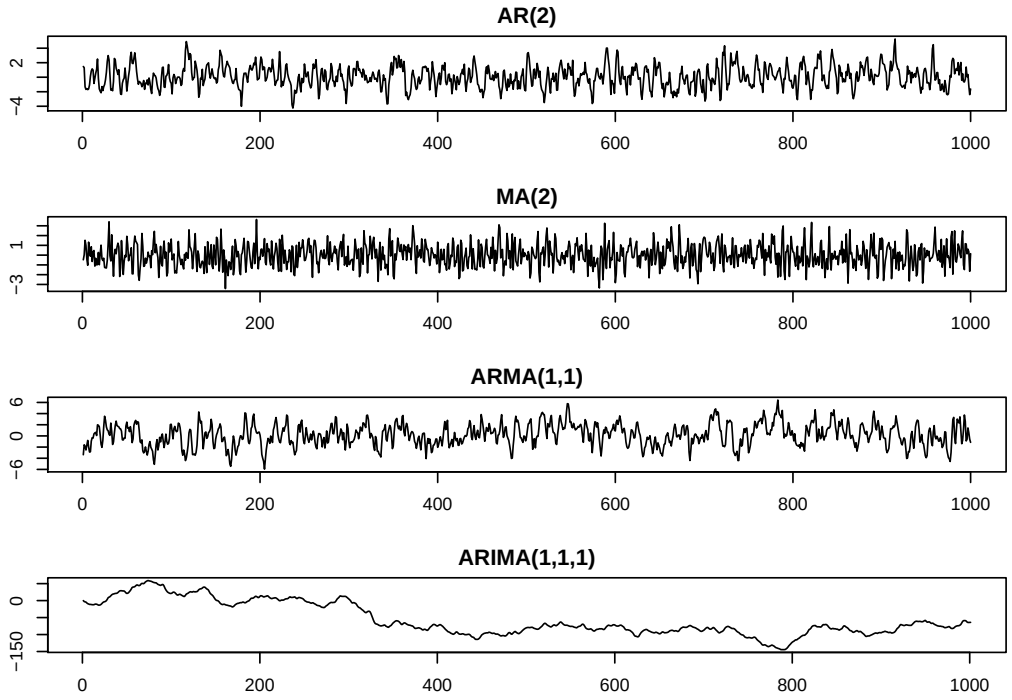
Visualmente, uma série estacionária flutua em torno de uma média fixa e sua a variância é constante ao longo do tempo. Deste modo, iniciamos nossa análise visualizando o gráfico das séries temporais.

Para visualizar o gráfico de cada uma das séries, podemos usar a função `plot()`. Neste exemplo, como desejamos aplicar a mesma função a sete séries diferentes, em vez de escrevermos a função sete vezes, podemos criar uma lista contendo todas as séries temporais e aplicar a função `plot()` a todos os elementos desta lista de uma vez.

No código abaixo, primeiramente criamos uma lista contendo as quatro séries temporais simuladas. Posteriormente, definimos alguns parâmetros gráficos: `mfrow = c(4,1)` significa que queremos dividir a área do gráfico em 4 linhas e uma coluna e prêenche-la por linhas. Usamos `mar = c(inferior, esquerda, superior, direita)` para definir o tamanho das margens de cada gráfico.

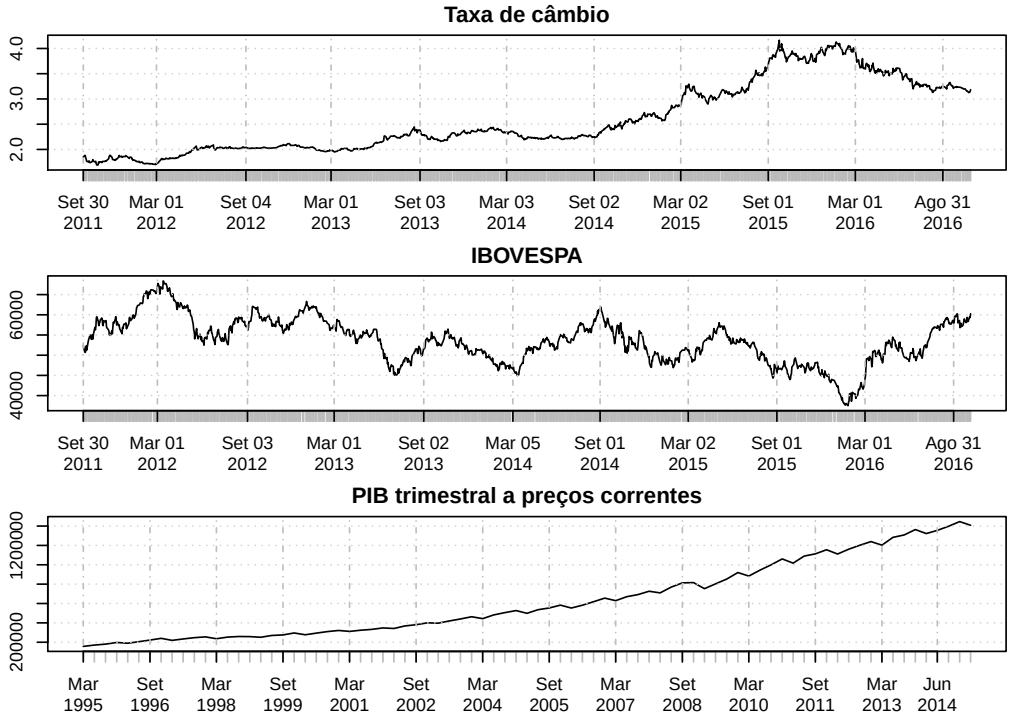
Por fim, usamos `lapply(x, função)` para aplicar a mesma função a cada um dos elementos da lista x. Poderíamos ter escrito simplesmente `lapply(series.simuladas, plot)`, mas o código abaixo gera um documento estéticamente mais agradável: insere um título em cada gráfico com o nome da série correspondente, elimina os rótulos dos eixos x e y e suprime uma "saída" do R que não nos interessa.

```
series.simuladas <- list("AR(2)"=ar2, "MA(2)"=ma2, "ARMA(1,1)"=arma11,
    "ARIMA(1,1,1)"=arima111)
par(mfrow= c(4,1), mar = c(3, 3, 2, 1))
invisible(lapply(names(series.simuladas), function(x) plot(series.simuladas[[x]], main=x,
  xlab = "", ylab = "")))
```



Fazemos o mesmo para as séries reais:

```
series.reais <- list("Taxa de câmbio"=txcambio, "IBOVESPA"=bvsp,
  "PIB trimestral a preços correntes"=pibtrim)
par(mfrow= c(3,1), mar = c(3, 3, 2, 1))
invisible(lapply(names(series.reais), function(x) plot(series.reais[[x]], main=x,
  xlab = "", ylab = "", type = "l")))
```



Apesar da análise gráfica nos fornecer uma boa "pista" sobre a estacionariedade das séries, é aconselhável realizar testes estatísticos. A condição necessária para estacionariedade fraca é que as raízes da equação característica devem estar fora do círculo unitário. Utilizamos, então, dois testes de raiz unitária.

3.1.1 Testes de raiz unitária

Os testes de raiz unitária de Dickey Fuller Aumentado (ADF) e Phillips-Perrón estabelecem as seguintes hipóteses:

Hipótese nula	H_0:	Existe raiz unitária;
Hipóteses alternativa	H_1:	Não existe raiz unitária.

Portanto, quando estes testes forem aplicados a séries temporais estacionárias, deverão apresentar um p-valor próximo de zero.

Diversos pacotes do R incluem funções para a realização de testes de raiz unitária, entre eles: `tseries`, `CADFtest`, `urca` e `fUnitRoots`. Neste exemplo, vamos utilizar o pacote `tseries`, o qual contém as funções `adf.test` e `pp.test` para conduzir os testes ADF e Phillips-Perrón, respectivamente. Para maiores detalhes sobre estes testes, sugerimos utilizar a ajuda do R, que pode ser acessada através da função `help()`.

```
install.packages("tseries")
```

```
library(tseries)
```

No código abaixo, combinamos as duas listas de séries que havíamos criado anteriormente em uma lista única, chamada "series" e, aplicamos o teste ADF a cada elemento dessa nova lista, utilizando a função `lapply()`. Posteriormente, aplicamos o teste de Phillips-Perrón.

```
series <- c(series.simuladas, series.reais)
lapply(series, adf.test)
```

```
## $`AR(2)`
##
##  Augmented Dickey-Fuller Test
##
## data:  X[[i]]
## Dickey-Fuller = -9.1806, Lag order = 9, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $`MA(2)`
##
##  Augmented Dickey-Fuller Test
##
## data:  X[[i]]
## Dickey-Fuller = -9.9117, Lag order = 9, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $`ARMA(1,1)`
##
##  Augmented Dickey-Fuller Test
```

```
##
## data:  X[[i]]
## Dickey-Fuller = -8.104, Lag order = 9, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $`ARIMA(1,1,1)`
##
##  Augmented Dickey-Fuller Test
##
## data:  X[[i]]
## Dickey-Fuller = -1.9665, Lag order = 9, p-value = 0.5925
## alternative hypothesis: stationary
##
##
## $`Taxa de câmbio`
##
##  Augmented Dickey-Fuller Test
##
## data:  X[[i]]
## Dickey-Fuller = -1.2915, Lag order = 10, p-value = 0.8782
## alternative hypothesis: stationary
##
##
## $IBOVESPA
##
##  Augmented Dickey-Fuller Test
##
## data:  X[[i]]
## Dickey-Fuller = -2.4315, Lag order = 10, p-value = 0.3957
## alternative hypothesis: stationary
##
##
## $`PIB trimestral a preços correntes`
##
##  Augmented Dickey-Fuller Test
##
## data:  X[[i]]
## Dickey-Fuller = -1.2344, Lag order = 4, p-value = 0.8892
## alternative hypothesis: stationary
```

```
lapply(series, pp.test)
```

```
## $`AR(2)`
##
##  Phillips-Perron Unit Root Test
##
## data:  X[[i]]
## Dickey-Fuller Z(alpha) = -256.47, Truncation lag parameter = 7,
## p-value = 0.01
## alternative hypothesis: stationary
##
##
## $`MA(2)`
```

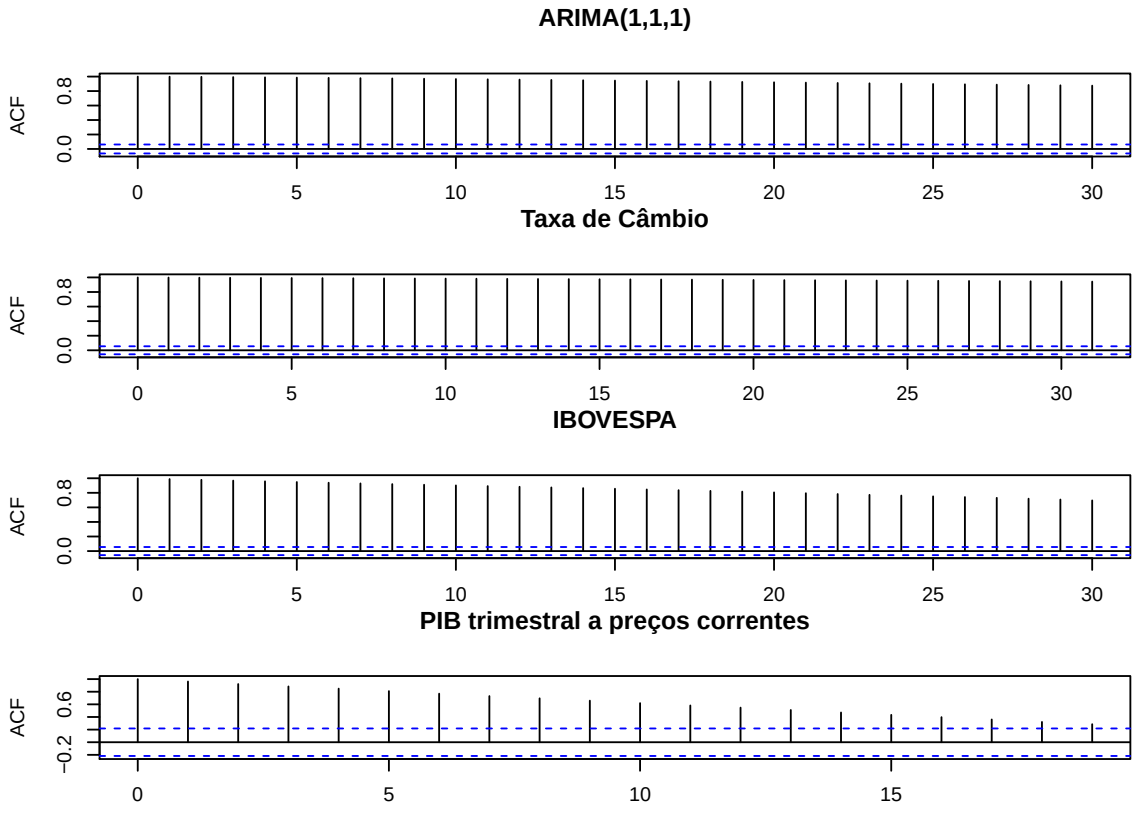
```
##
##  Phillips-Perron Unit Root Test
##
## data:  X[[i]]
## Dickey-Fuller Z(alpha) = -620.13, Truncation lag parameter = 7,
## p-value = 0.01
## alternative hypothesis: stationary
##
##
## $`ARMA(1,1)`
##
##  Phillips-Perron Unit Root Test
##
## data:  X[[i]]
## Dickey-Fuller Z(alpha) = -185.25, Truncation lag parameter = 7,
## p-value = 0.01
## alternative hypothesis: stationary
##
##
## $`ARIMA(1,1,1)`
##
##  Phillips-Perron Unit Root Test
##
## data:  X[[i]]
## Dickey-Fuller Z(alpha) = -5.0779, Truncation lag parameter = 7,
## p-value = 0.8265
## alternative hypothesis: stationary
##
##
## $`Taxa de câmbio`
##
##  Phillips-Perron Unit Root Test
##
## data:  X[[i]]
## Dickey-Fuller Z(alpha) = -4.6777, Truncation lag parameter = 7,
## p-value = 0.8488
## alternative hypothesis: stationary
##
##
## $IBOVESPA
##
##  Phillips-Perron Unit Root Test
##
## data:  X[[i]]
## Dickey-Fuller Z(alpha) = -15.537, Truncation lag parameter = 7,
## p-value = 0.243
## alternative hypothesis: stationary
##
##
## $`PIB trimestral a preços correntes`
##
##  Phillips-Perron Unit Root Test
##
## data:  X[[i]]
```

```
## Dickey-Fuller Z(alpha) = -1.3541, Truncation lag parameter = 3,
## p-value = 0.9803
## alternative hypothesis: stationary
```

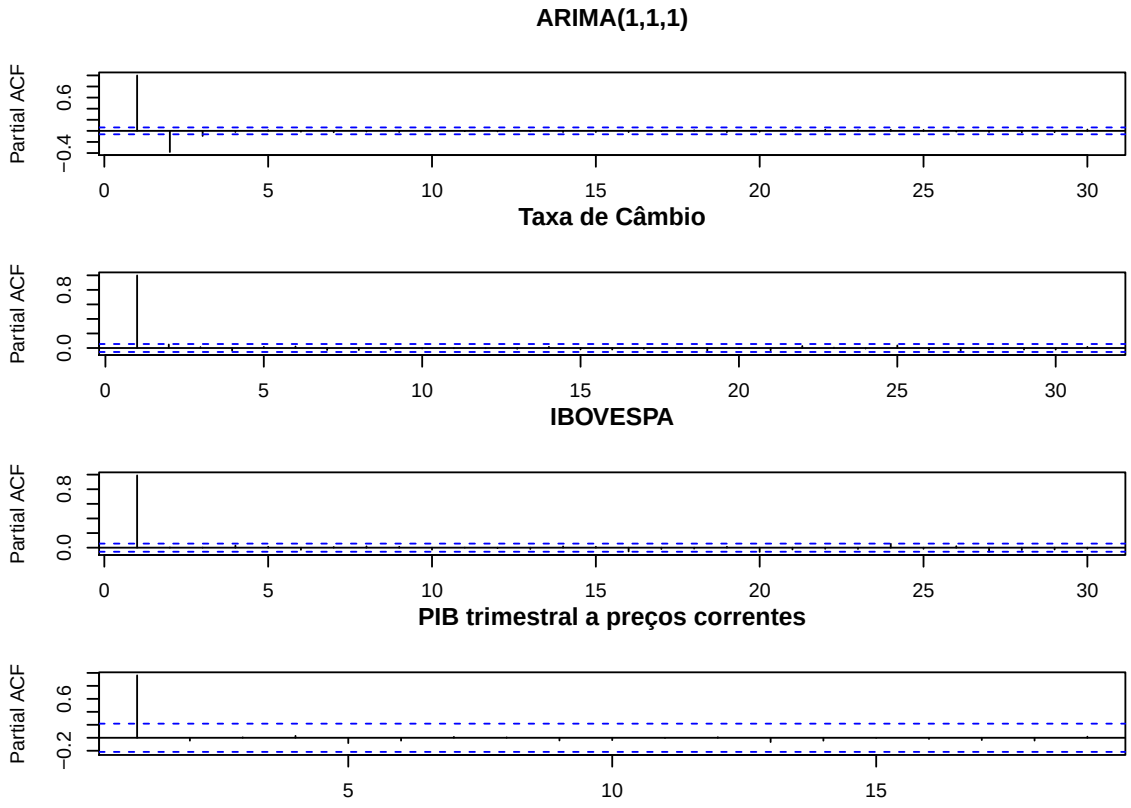
Após a análise gráfica e os testes de raiz unitária, podemos concluir que as séries arima111, txcambio, bvsp e pibtrim parecem ser não estacionárias. O gráfico da função de autocorrelação (FAC) de uma série não estacionária tem um decaimento muito lento e o gráfico da função de autocorrelação parcial (FACP) apresenta um valor próximo de 1, como pode ser visto nos exemplos a seguir.

No código abaixo, criamos duas listas: a primeira delas contém as séries estacionárias e a segunda contém as séries não estacionárias. Aplicamos as funções `acf()` e `pacf()` a cada item da segunda lista a fim de visualizarmos os gráficos da FAC e da FACP das séries não estacionárias.

```
series.estacionarias <- list("AR(2)"=ar2, "MA(2)"=ma2, "ARMA(1,1)"=arma11)
series.nao.estacionarias <- list("ARIMA(1,1,1)"=arima111, "Taxa de Câmbio"=txcambio,
  "IBOVESPA"=bvsp, "PIB trimestral a preços correntes"=pibtrim)
par(mfrow = c(4,1), mar = c(2, 4.5, 3, 1))
invisible(lapply(names(series.nao.estacionarias), function(x)
  acf(series.nao.estacionarias[[x]], main=x)))
```




```
invisible(lapply(names(series.nao.estacionarias), function(x)
  pacf(series.nao.estacionarias[[x]], main=x)))
```

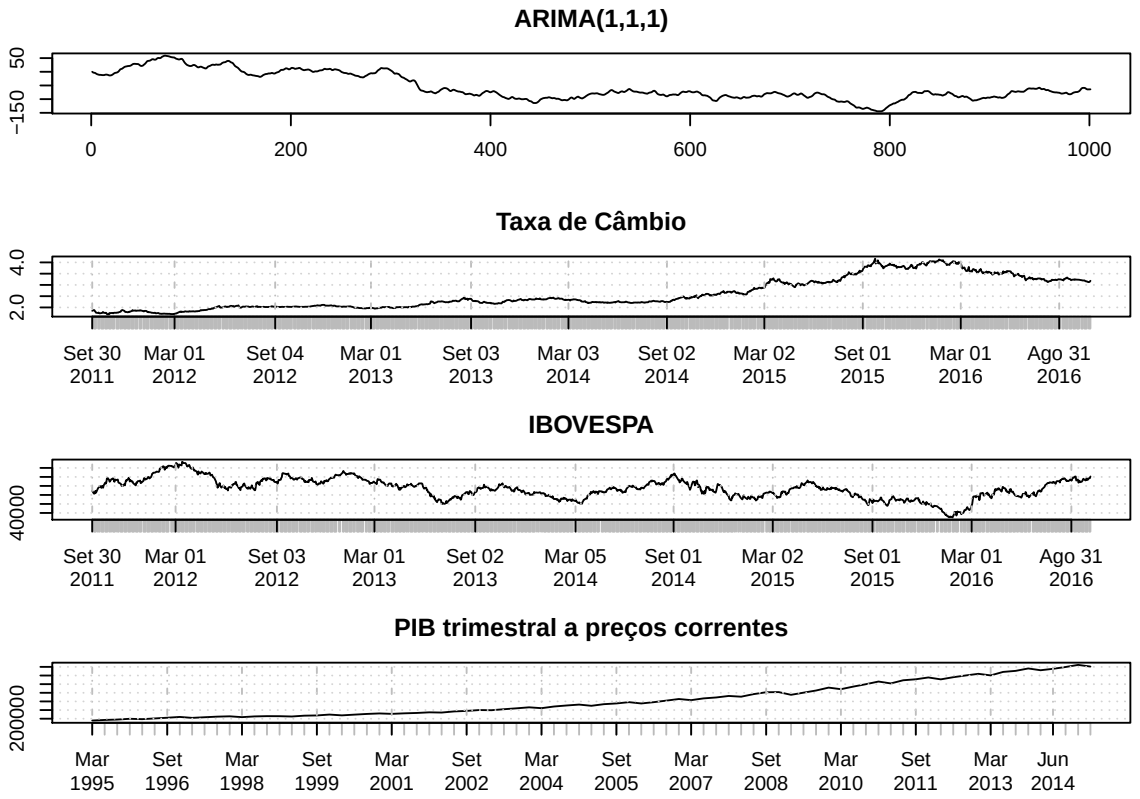


3.2 Tornando as séries estacionárias

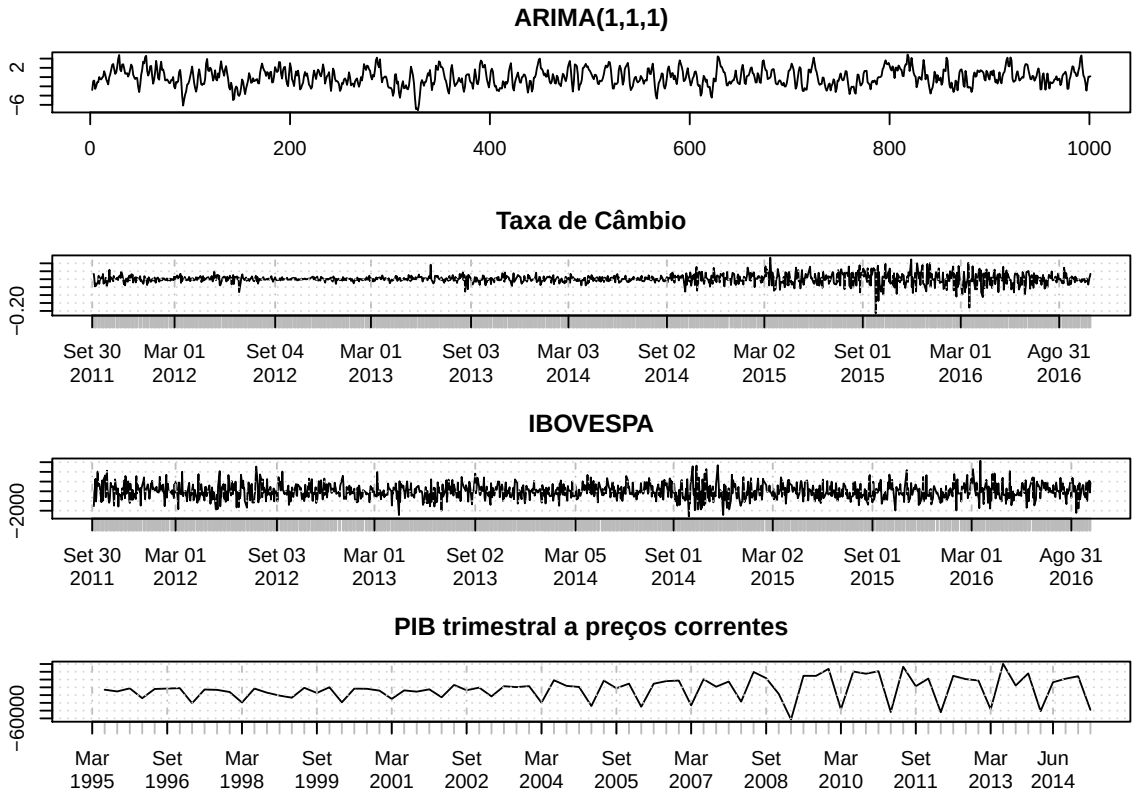
As séries não estacionárias devem passar por uma transformação a fim de estabilizar sua média, variância e autocovariâncias. Podemos tornar uma série estacionária através de sucessivas diferenciações. O número de diferenças necessárias para que o processo se torne estacionário é aquele a partir do qual as funções de autocorrelação (FAC) e autocorrelação parcial (FACP) amostrais decrescem rapidamente. Podemos repetir os testes de raiz unitária para confirmar que a série tornou-se estacionária após determinado número de diferenciações.

No código abaixo, primeiramente criamos uma lista com as séries diferenciadas aplicando a função `diff()` a cada item da lista que contém as séries não estacionárias. Então, utilizamos a função `plot()` para visualizar o gráfico das séries antes e depois da diferenciação.

```
series.diferenciadas <- lapply(series.nao.estacionarias, diff)
par(mfrow = c(4,1), mar = c(3, 3, 3, 1))
invisible(lapply(names(series.nao.estacionarias), function(x)
  plot(series.nao.estacionarias[[x]], main=x, ylab = "")))
```



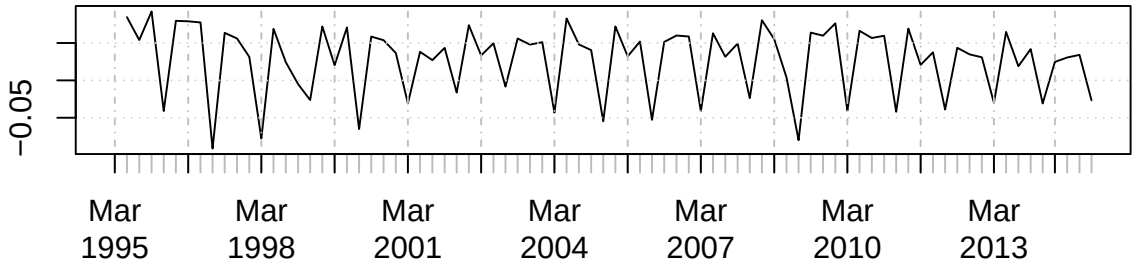

```
invisible(lapply(names(series.diferenciadas), function(x) plot(series.diferenciadas[[x]],
  main=x)))
```



Os gráficos sugerem que a primeira diferenciação foi suficiente para que as séries apresentem média constante. Apesar disso, a variância parece mudar ao longo do tempo. Neste caso, uma alternativa é utilizar um transformação do tipo Box-Cox, por exemplo, tirando o log da série ou utilizando a função `BoxCox()` do pacote "forecast".

No código abaixo, tiramos o log da série do PIB trimestral antes de diferenciá-la. Após esta transformação, o gráfico da série passa a apresentar variância mais homogênea ao longo do tempo.

```
dif.log.pibtrim <- diff(log(pibtrim))
par(mar = c(3, 3, 3, 1))
plot(dif.log.pibtrim, main="")
```



Então, substituímos a série diferenciada do PIB na lista "series.diferenciadas" pela série transformada e diferenciada.

```
series.diferenciadas[[4]] <- dif.log.pibtrim
```

3.3 Identificação do modelo

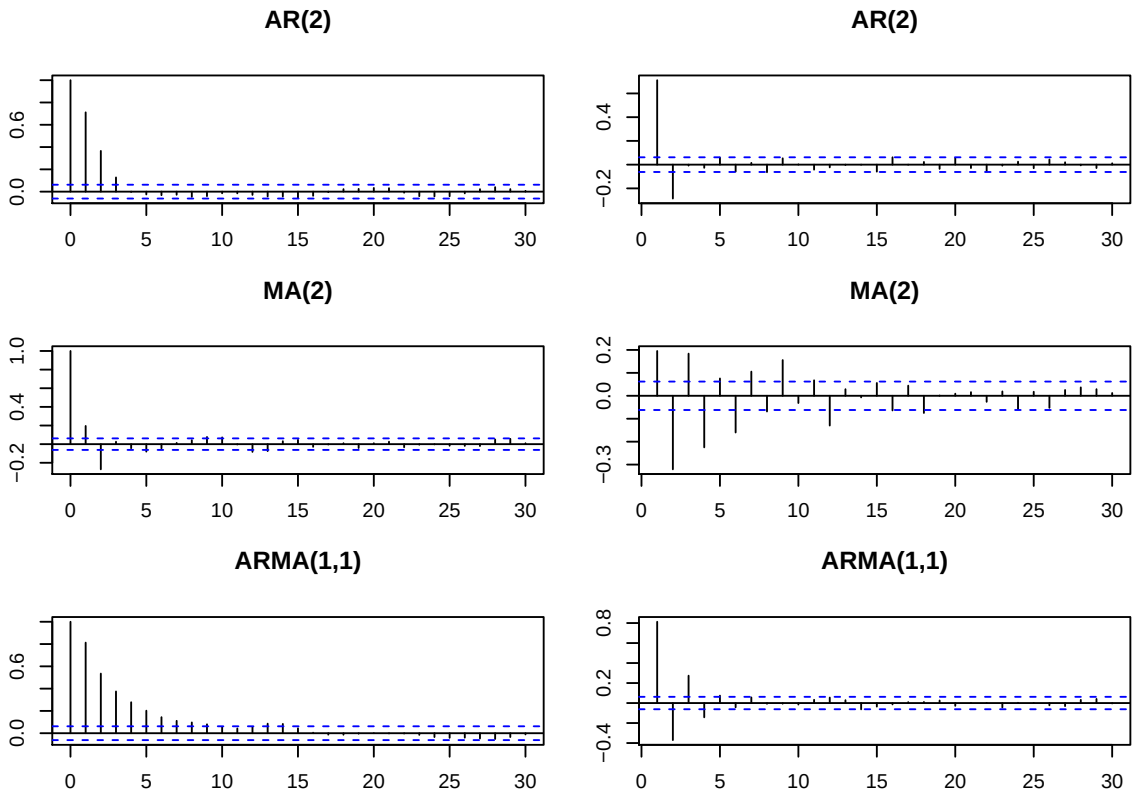
A identificação dos modelos é feita comparando-se as funções de autocorrelação (FAC) e funções de autocorrelação parcial (FACP) empíricas (ou seja, aquelas estimadas a partir dos dados que estamos analisando) com os seus valores teóricos. A tabela abaixo apresenta um resumo do comportamento da FAC e da FACP para os modelos AR, MA e ARMA:

	AR	MA	ARMA
FAC	Decai exponencialmente	Corte brusco após a defasagem q	Decai exponencialmente após a defasagem q
FACP	Corte brusco após a defasagem p	Decai exponencialmente	Decai exponencialmente após a defasagem p

Muitas vezes, a identificação através do gráficos não é clara e direta. Nestes casos, escolhemos o modelo com a maior ordem (considerando os lags significativos das funções FAC e FACP) e estimamos combinações de ordens menores até encontrarmos aquela que minimiza os critérios de informação. Em geral, utilizamos o critério de informação AIC para amostras pequenas, BIC para amostras grandes (digamos, n $>$ 500) e HQ como um critério intermediário.

Abaixo, utilizamos as funções `acf()` e `pacf()` para visualizarmos os gráficos da FAC e FACP para cada uma das séries e realizarmos a etapa de identificação do modelo.

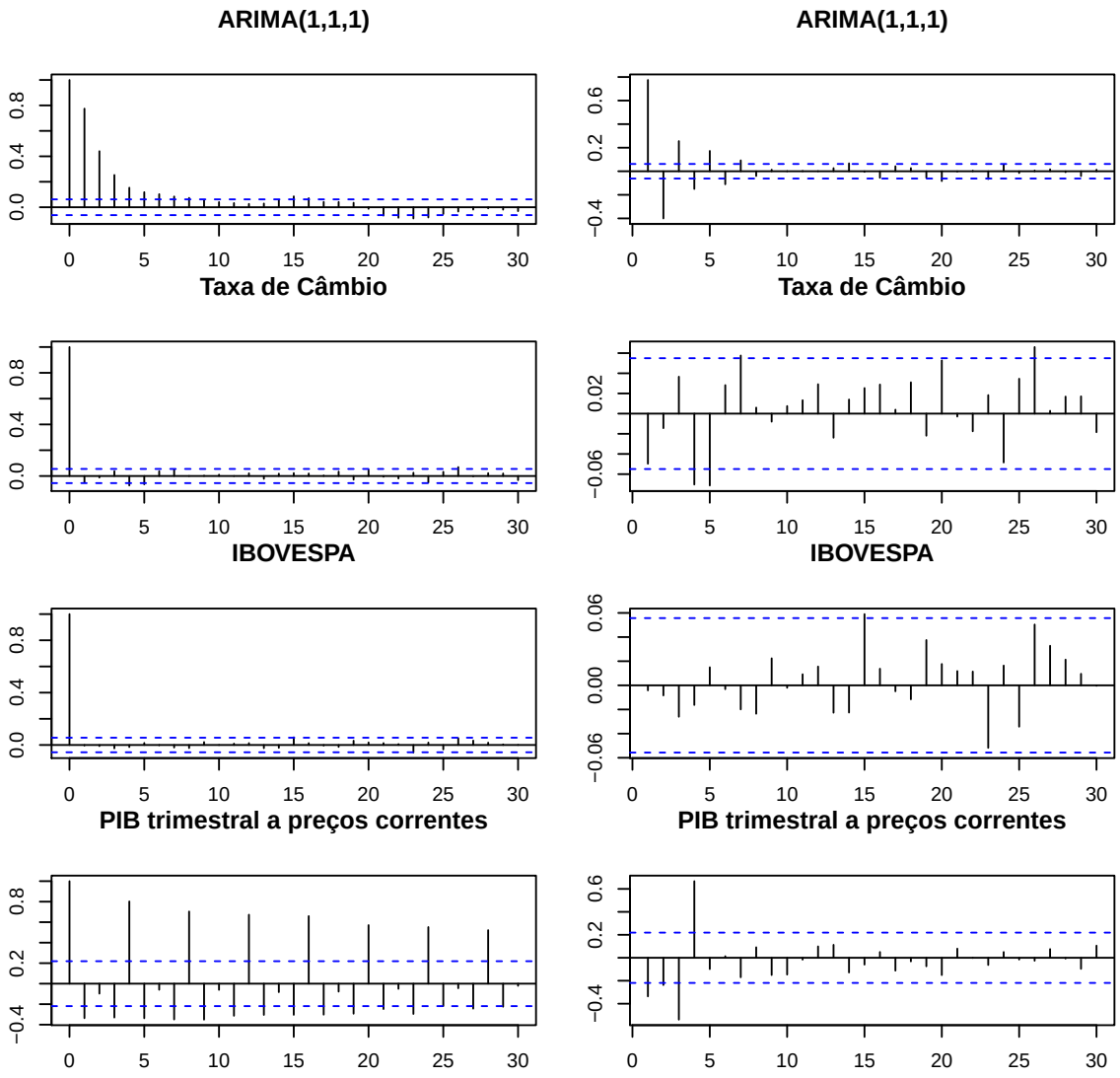
```
par(mfcol = c(3,2), mar = c(3, 3, 3, 1))
invisible(lapply(names(series.estacionarias), function(x) acf(series.estacionarias[[x]],
  main = x)))
invisible(lapply(names(series.estacionarias), function(x) pacf(series.estacionarias[[x]],
  main = x)))
```



Conforme esperado:

- Para a série AR(2), a FAC decai exponencialmente e a FACP apresenta um corte brusco após a segunda defasagem;
- Para a série MA(2), a FACP decai exponencialmente e a FAC apresenta um corte brusco após a segunda defasagem;
- Para a série ARMA(1,1), ambas FAC e FACP apresentam decaimento exponencial após a primeira defasagem.

```
par(mfcol = c(4,2), mar = c(2, 3, 3, 1))
invisible(lapply(names(series.diferenciadas), function(x)
  acf(na.omit(series.diferenciadas[[x]]), main = x, lag.max = 30)))
invisible(lapply(names(series.diferenciadas), function(x)
  pacf(na.omit(series.diferenciadas[[x]]), main = x, lag.max = 30)))
```



Para as séries diferenciadas obtivemos os seguintes resultados:

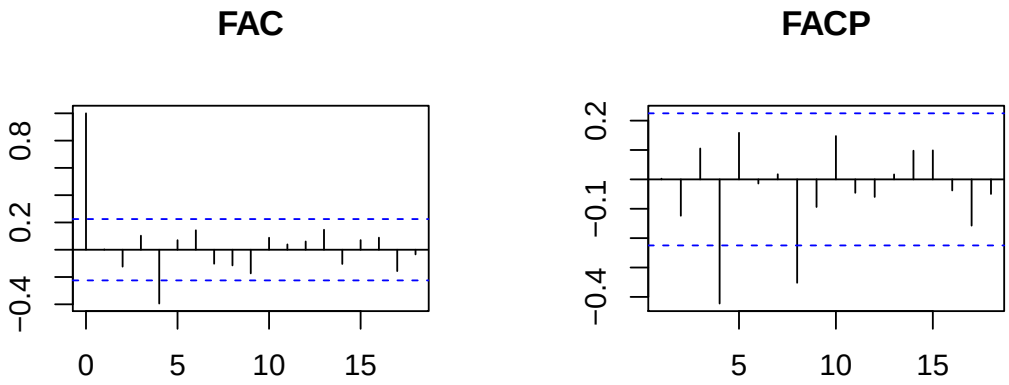
- Para a série ARIMA(1,1,1), ambas FAC e FACP apresentam decaimento exponencial após a primeira defasagem, conforme esperado;
- Para a série da taxa de câmbio, não está claro o melhor modelo a utilizar. Neste caso, podemos lançar mão da função 'auto.arima()', contida no pacote "forecast", a qual nos sugere o melhor modelo de acordo com critérios de informação, tais como AIC e BIC;
- Para a série do Ibovespa não parece haver nenhuma defasagem significativa tanto na FAC quanto na FACP, indicando que esta série segue um passeio aleatório. Note que quando a série diferenciada é um ruído branco, o modelo para a série original pode ser escrito como:

$$y_t - y_{t-1} = \varepsilon_t \qquad y_t = y_{t-1} + \varepsilon_t$$

- Para a série do PIB trimestral a preços correntes, a FAC apresenta um comportamento típico de série com sazonalidade: decaimento lento para as defasagens 4, 8, 12, etc. Isto sugere que tiremos uma diferença sazonal da série diferenciada.

Para tirar uma diferença sazonal, utilizamos novamente a função `diff()`, mas agora incluímos um argumento indicando a defasagem correspondente à sazonalidade. Note que quando diferenciamos uma série, perdemos observações. Neste caso, a série (duplamente) diferenciada do PIB trimestral perdeu as observações referentes aos 5 primeiros trimestres. Estes valores foram substituídos por NAs. Deste modo, para conseguirmos imprimir os gráficos da FAC e da FACP utilizando apenas as datas para as quais temos valores, tivemos que usar o comando `na.omit()`.

```
difs.pibtrim <- diff(dif.log.pibtrim, 4)
par(mfrow = c(1,2))
acf(na.omit(difs.pibtrim), main="FAC", xlab="", ylab="")
pacf(na.omit(difs.pibtrim), main="FACP", xlab="", ylab="")
```



Após a diferenciação sazonal, os gráficos da FAC e da FACP sugerem a inclusão de uma parte sazonal no modelo, aparentemente de ordem (2,1,1) (para identificar a ordem sazonal, olhamos apenas para os lags sazonais: 4, 8, 12, etc.). Na próxima etapa, a de estimação dos coeficientes, escolheremos o modelo mais adequado para o PIB trimestral minimizando o critério de informação AIC.

3.4 Estimação dos parâmetros

Para estimar os coeficientes do modelo podemos usar, por exemplo, as funções `arima()` do pacote "stats" ou as funções `Arima()` ou `auto.arima()` do pacote "forecast". Por exemplo, para estimar um AR(2) para a primeira série simulada, escrevemos `Arima(ar2[1:990], order = c(2,0,0))`. "ar2" é o nome que atribuímos à série de tempo quando ela foi criada, [1:990] refere-se às observações que serão utilizadas na estimação, uma vez que desejamos reservar uma parte da amostra para comparar com o resultado da previsão, e "order = c(2,0,0)" informa que o modelo a ser utilizado é um ARIMA(2,0,0) (ou AR(2)). Entre os valores resultantes da aplicação desta função, obtemos os critério de informação AIC e BIC, os quais podem ser utilizados para a escolha do modelo quando estamos em dúvida.

Após a estimação, podemos visualizar um gráfico da série original contra o modelo ajustado.

```
install.packages("forecast")
```

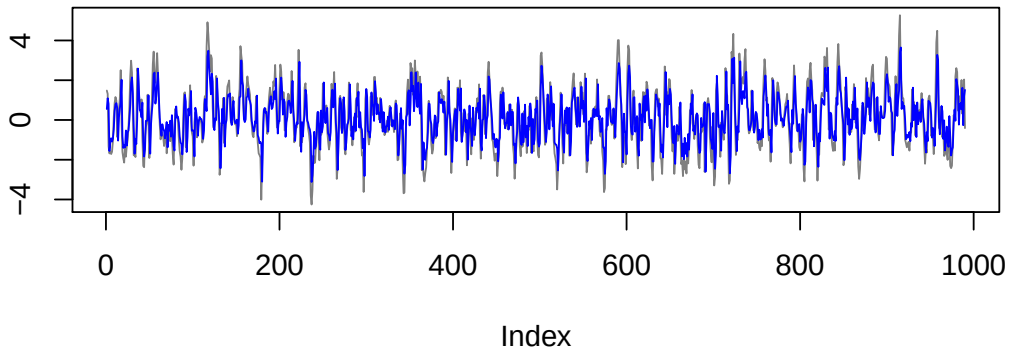
```
library(forecast)
```

AR(2)

```
est.ar2 <- Arima(ar2[1:990], order = c(2,0,0))
est.ar2
```

```
## Series: ar2[1:990]
## ARIMA(2,0,0) with non-zero mean
##
## Coefficients:
##          ar1      ar2  intercept
##       0.9198  -0.2898     0.0952
## s.e.  0.0304   0.0305     0.0873
##
## sigma^2 estimated as 1.038:  log likelihood=-1422.28
## AIC=2852.56   AICc=2852.6   BIC=2872.15
```

```
plot(est.ar2$x,col="gray50", type = "l", ylab="")
lines(fitted(est.ar2),col="blue")
```

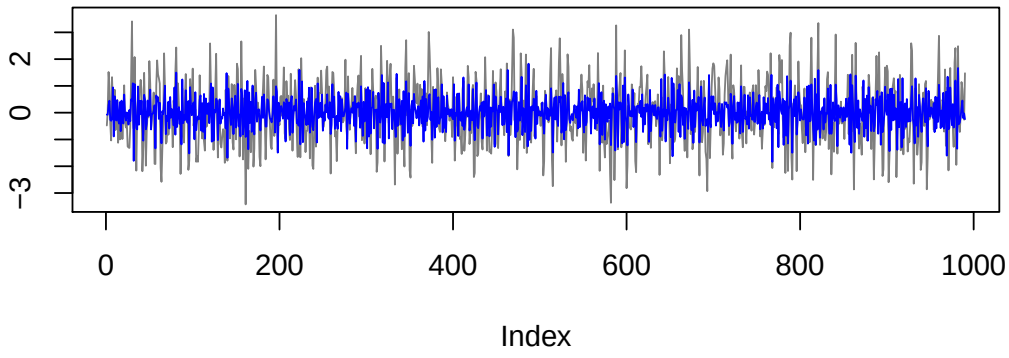


MA(2)

```
est.ma2 <- Arima(ma2[1:990], order = c(0,0,2))
est.ma2
```

```
## Series: ma2[1:990]
## ARIMA(0,0,2) with non-zero mean
##
## Coefficients:
##          ma1      ma2  intercept
##       0.4798  -0.4217     0.0013
## s.e.  0.0292   0.0289     0.0335
##
## sigma^2 estimated as 0.9954:  log likelihood=-1401.75
## AIC=2811.5   AICc=2811.54   BIC=2831.09
```

```
plot(est.ma2$x,col="gray50", type="l", ylab="")
lines(fitted(est.ma2),col="blue")
```

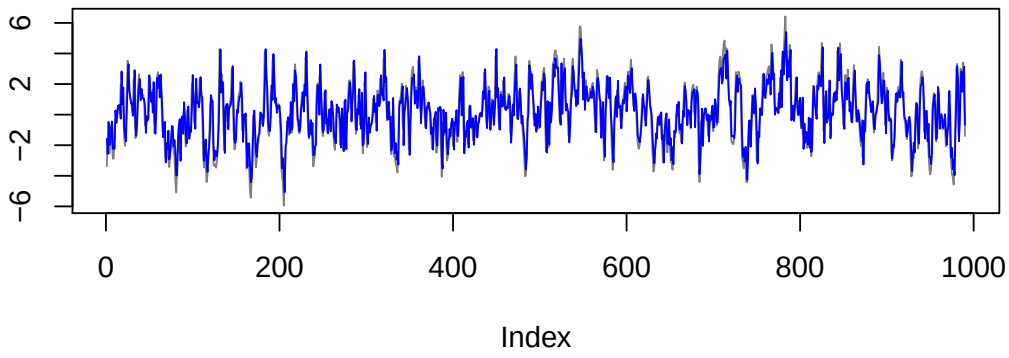


ARMA(1,1)

```
est.arma11 <- Arima(arma11[1:990], order = c(1,0,1))
est.arma11
```

```
## Series: arma11[1:990]
## ARIMA(1,0,1) with non-zero mean
##
## Coefficients:
##          ar1     ma1  intercept
##       0.6502  0.6533     0.1779
## s.e.  0.0265  0.0265     0.1490
##
## sigma^2 estimated as 0.9917:  log likelihood=-1400.05
## AIC=2808.09   AICc=2808.13   BIC=2827.68
```

```
plot(est.arma11$x,col="gray50", type = "l", ylab="")
lines(fitted(est.arma11),col="blue")
```



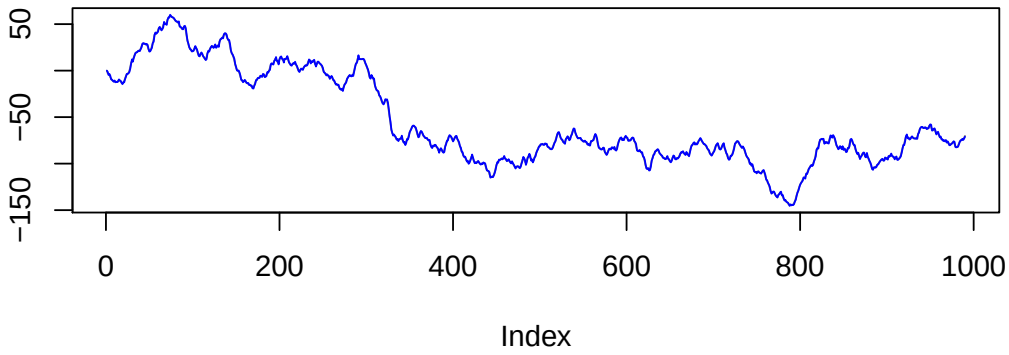
ARIMA(1,1,1)

```
est.arima111 <- Arima(arima111[1:990], order = c(1,1,1))
est.arima111
```

```
## Series: arima111[1:990]
## ARIMA(1,1,1)
##
## Coefficients:
```

```
##          ar1     ma1
##       0.5614  0.7382
## s.e.  0.0285  0.0228
##
## sigma^2 estimated as 0.9994:  log likelihood=-1402.97
## AIC=2811.95   AICc=2811.97   BIC=2826.64
```

```
plot(est.arima111$x,col="gray50", type = "l", ylab="")
lines(fitted(est.arima111),col="blue")
```



Taxa de câmbio

```
auto.arima(txcambio[1:1250], ic="bic")
```

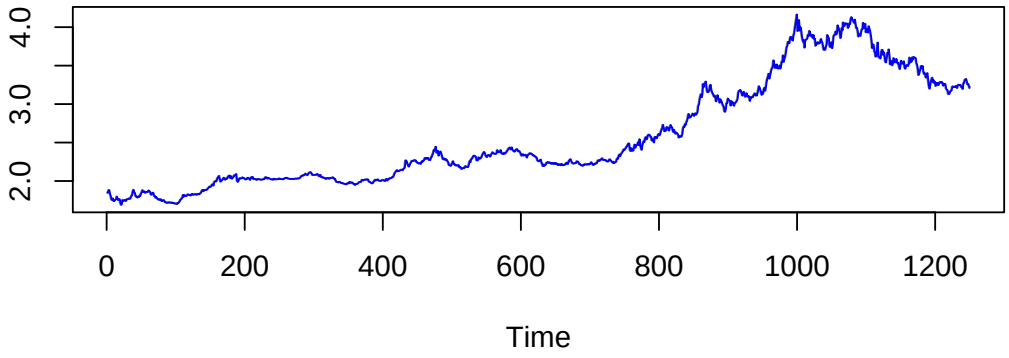
```
## Series: txcambio[1:1250]
## ARIMA(0,1,0) with drift
##
## Coefficients:
##         drift
##       0.0011
## s.e.  0.0008
##
## sigma^2 estimated as 0.0008767:  log likelihood=2624.33
## AIC=-5244.67   AICc=-5244.66   BIC=-5234.41
```

```
auto.arima(txcambio[1:1250], ic="aic")
```

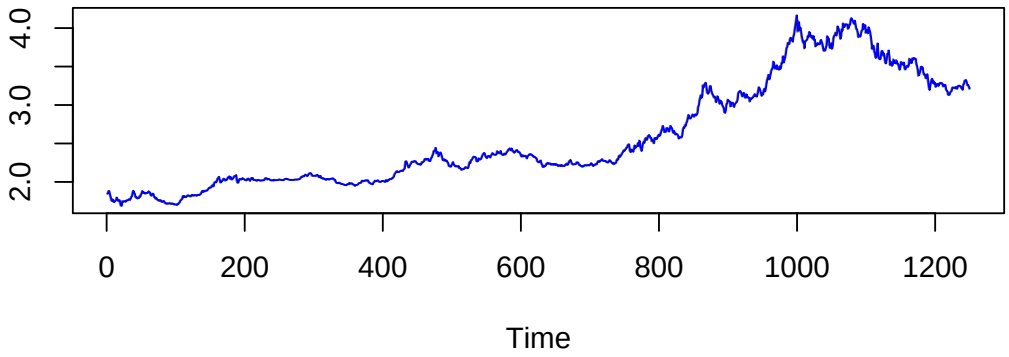
```
## Series: txcambio[1:1250]
## ARIMA(2,1,2)
##
## Coefficients:
##           ar1      ar2      ma1     ma2
##       -0.0494  -0.7587  -0.0127  0.7897
## s.e.   0.2298   0.1240   0.2253  0.0962
##
## sigma^2 estimated as 0.0008699:  log likelihood=2630.67
## AIC=-5251.34   AICc=-5251.29   BIC=-5225.69
```

```
est.txcambio <- Arima(txcambio[1:1250], order = c(0,1,0))
est.txcambio2 <- Arima(txcambio[1:1250], order = c(2,1,2))

plot(as.ts(est.txcambio$x),col="gray50", type="l", ylab="")
lines(fitted(est.txcambio), col="blue")
```




```
plot(as.ts(est.txcambio2$x),col="gray50", type="l", ylab="")
lines(fitted(est.txcambio2), col="blue")
```



Ibovespa

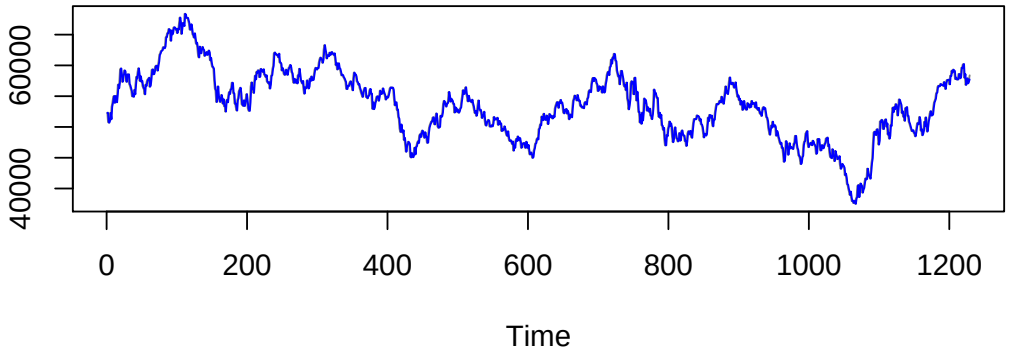
```
auto.arima(bvsp[1:1229], ic="bic")
```

```
## Series: bvsp[1:1229]
## ARIMA(0,1,0) with drift
##
## Coefficients:
##         drift
##        4.9422
## s.e.  22.3960
##
## sigma^2 estimated as 616449:  log likelihood=-9927.64
## AIC=19859.27   AICc=19859.28   BIC=19869.5
```

```
est.bvsp <- Arima(bvsp[1:1229], order = c(0,1,0))
est.bvsp
```

```
## Series: bvsp[1:1229]
## ARIMA(0,1,0)
##
## sigma^2 estimated as 615971:  log likelihood=-9927.66
## AIC=19857.32   AICc=19857.33   BIC=19862.43
```

```
plot(as.ts(est.bvsp$x),col="gray50", type="l", ylab="")
lines(fitted(est.bvsp), col="blue")
```



PIB trimestral

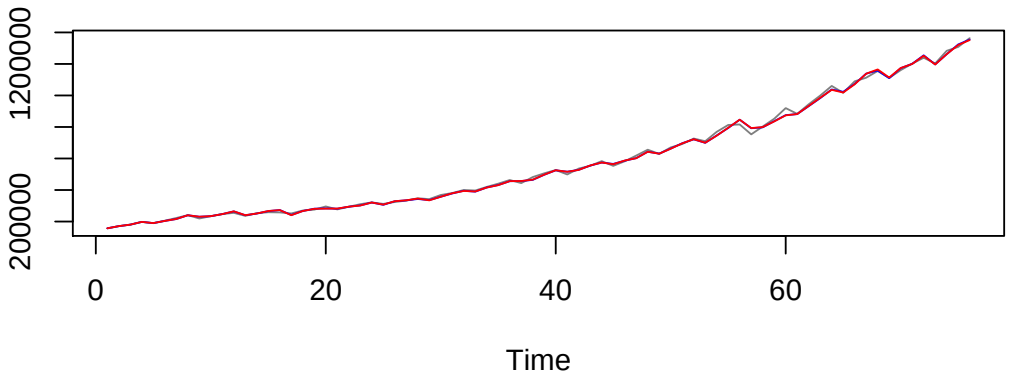
```
est.pibtrim <- Arima(pibtrim[1:76], order = c(0,1,0), seasonal= list(order=c(2,1,1),
  period=4))
est.pibtrim
```

```
## Series: pibtrim[1:76]
## ARIMA(0,1,0)(2,1,1)[4]
##
## Coefficients:
##          sar1     sar2     sma1
##       -0.4448  -0.2347  -0.1648
## s.e.   0.2572   0.1640   0.2510
##
## sigma^2 estimated as 174827937:  log likelihood=-773.84
## AIC=1555.69   AICc=1556.29   BIC=1564.74
```

```
est.pibtrim2 <- Arima(pibtrim[1:76], order = c(0,1,0), seasonal = list(order=c(1,1,1),
  period=4))
est.pibtrim2
```

```
## Series: pibtrim[1:76]
## ARIMA(0,1,0)(1,1,1)[4]
##
## Coefficients:
##          sar1     sma1
##       -0.1830  -0.4138
## s.e.   0.1713   0.1390
##
## sigma^2 estimated as 176867256:  log likelihood=-774.64
## AIC=1555.29   AICc=1555.64   BIC=1562.07
```

```
plot(as.ts(est.pibtrim$x), col="gray50", type="l", ylab="")
lines(fitted(est.pibtrim), col="blue")
lines(fitted(est.pibtrim2), col="red")
```



3.5 Diagnóstico

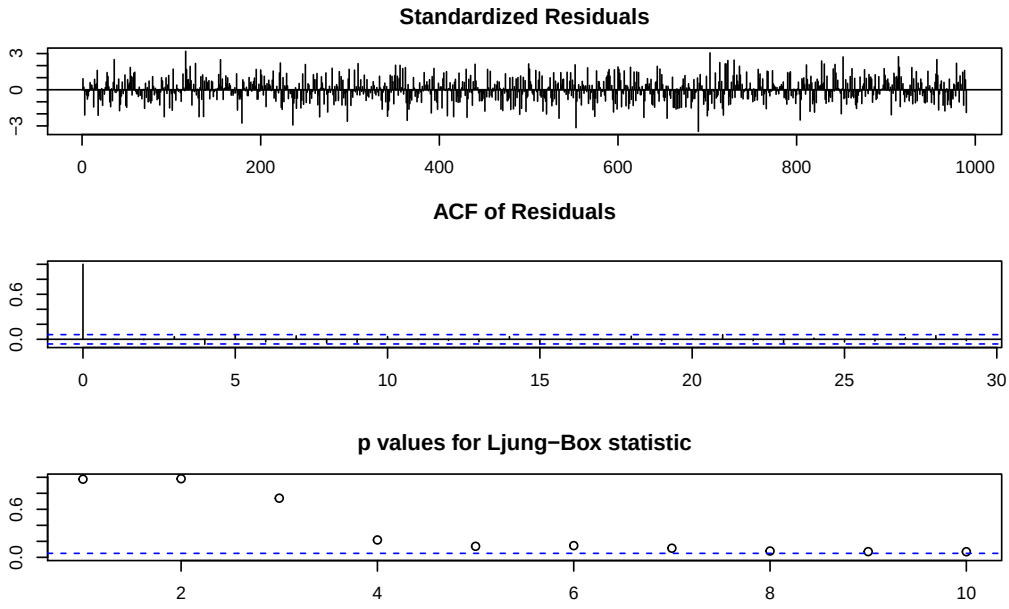
A fim de realizar um diagnóstico da qualidade do ajuste, analisamos os resíduos, procurando identificar qualquer sinal de não-aleatoriedade. Algumas opções de análise são:

- Visualizar um gráfico dos resíduos, o qual deve lembrar um ruído branco;
- Visualizar o gráfico da função da autocorrelação dos resíduos, o qual não deve apresentar qualquer sinal de autocorrelação;
- Realizar o teste de Ljung-Box: sua hipótese nula é de que os resíduos são independentemente distribuídos. Portanto, para cada defasagem, esperamos que o p-valor seja alto.

Podemos realizar estas três análises de uma só vez utilizando a função `tsdiag()`. Como argumento da função, informamos o nome do modelo ajustado.

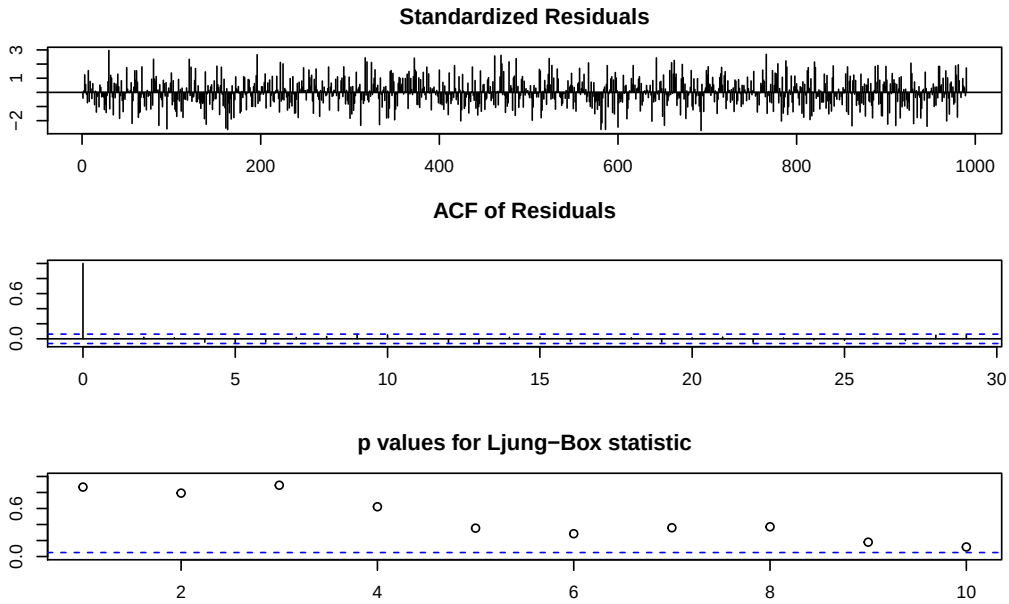
AR(2)

```
par(mar = c(3,3,3,1))
tsdiag(est.ar2)
```



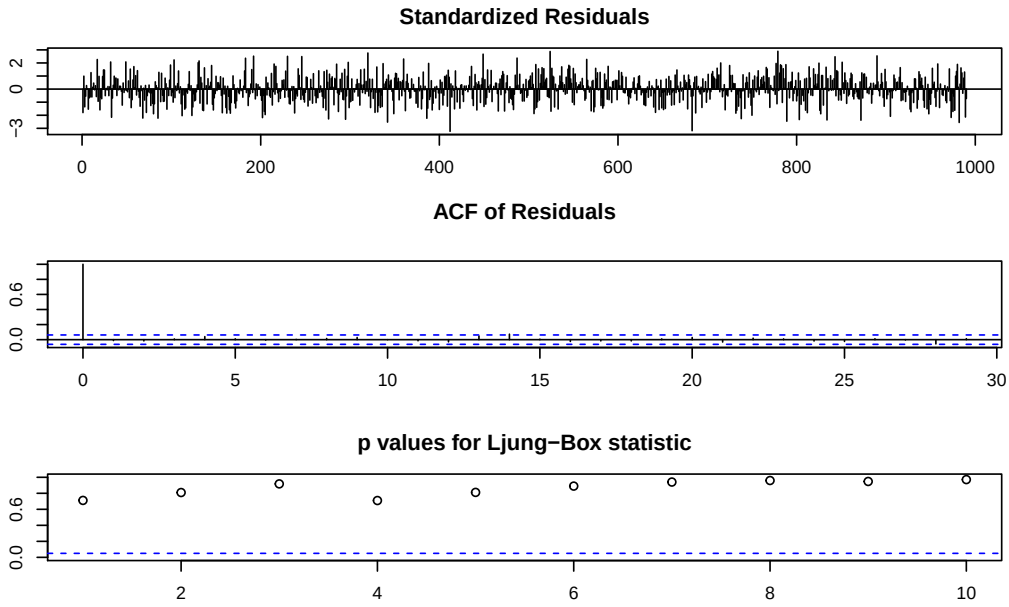
MA(2)

```
par(mar = c(3,3,3,1))
tsdiag(est.ma2)
```



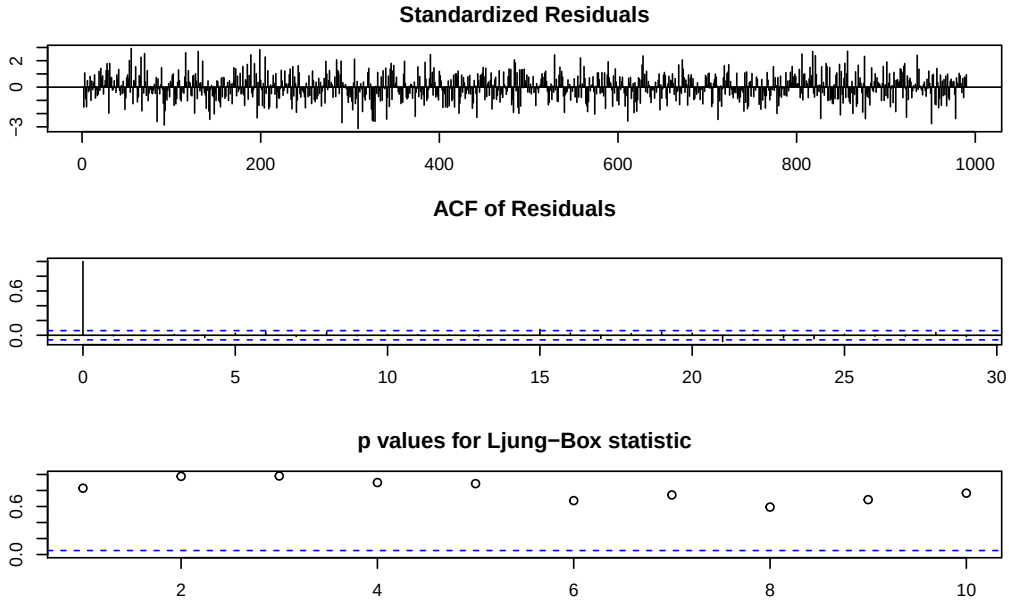
ARMA(1,1)

```
par(mar = c(3,3,3,1))
tsdiag(est.arma11)
```



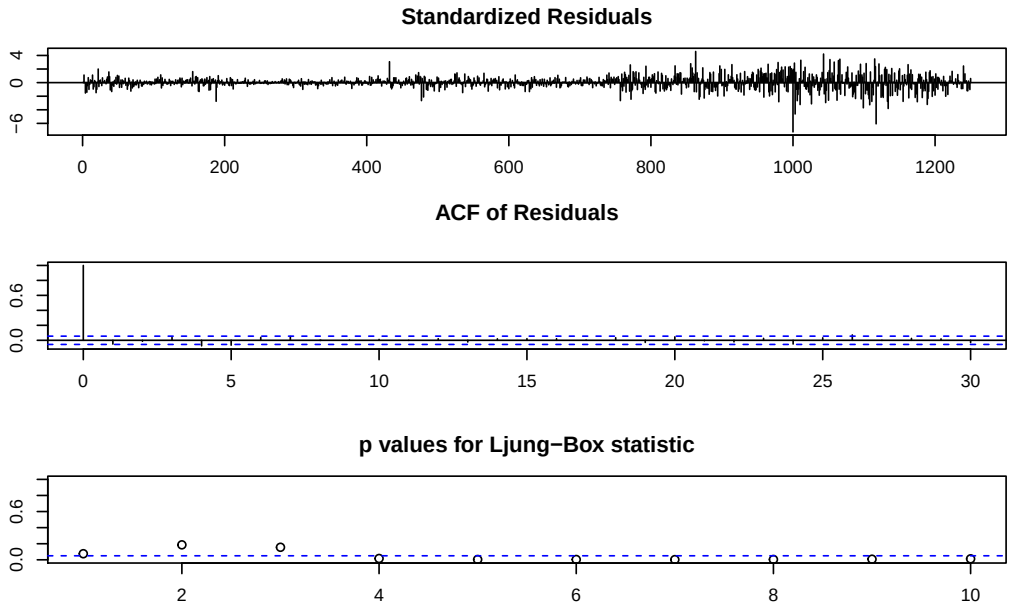
ARIMA(1,1,1)

```
par(mar = c(3,3,3,1))
tsdiag(est.arima111)
```

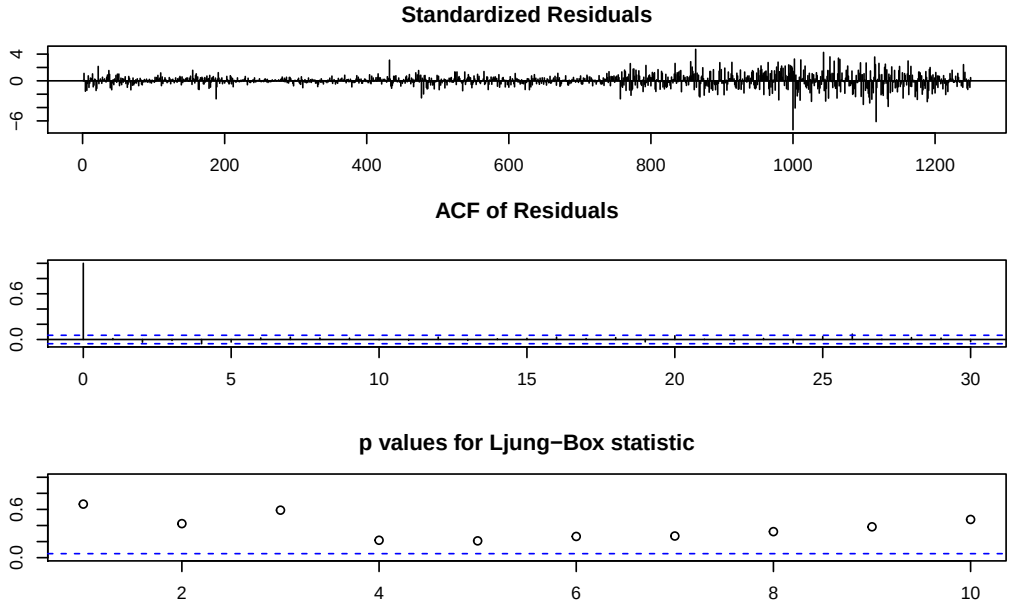


Taxa de câmbio

```
par(mar = c(3,3,3,1))
tsdiag(est.txcambio)
```

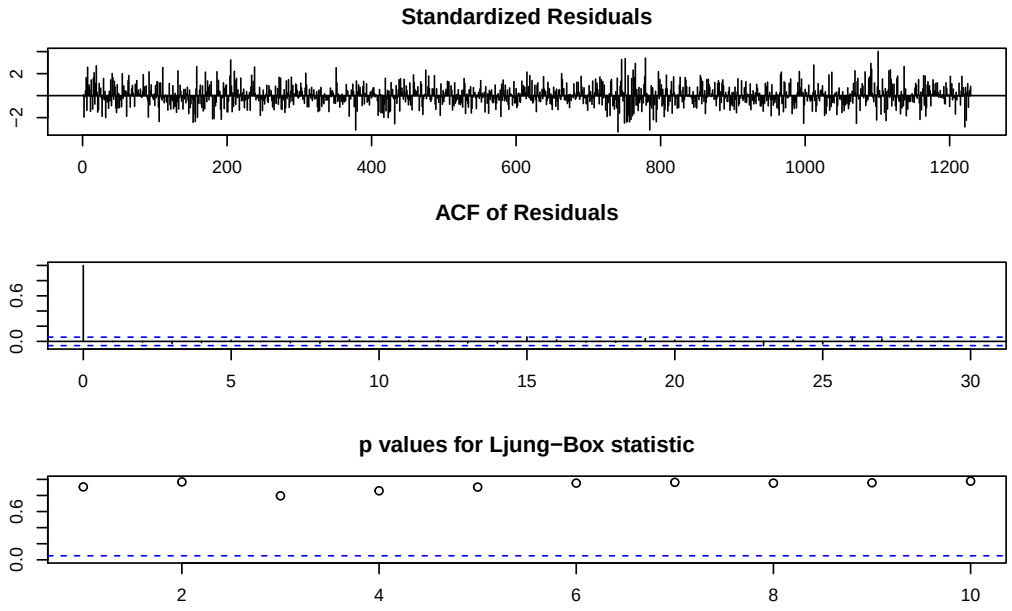



```
tsdiag(est.txcambio2)
```



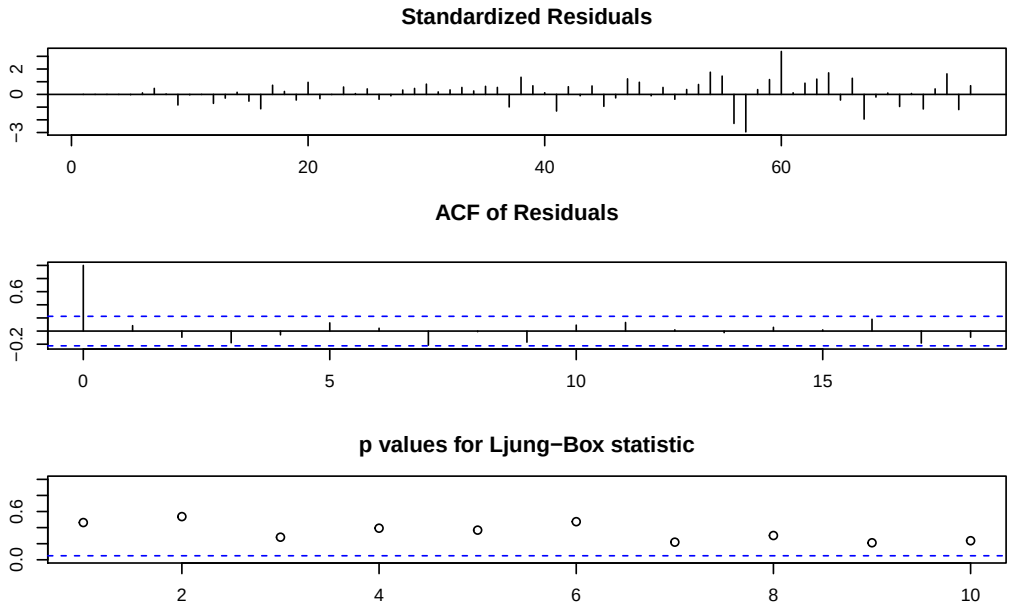
Ibovespa

```
par(mar = c(3,3,3,1))
tsdiag(est.bvsp)
```

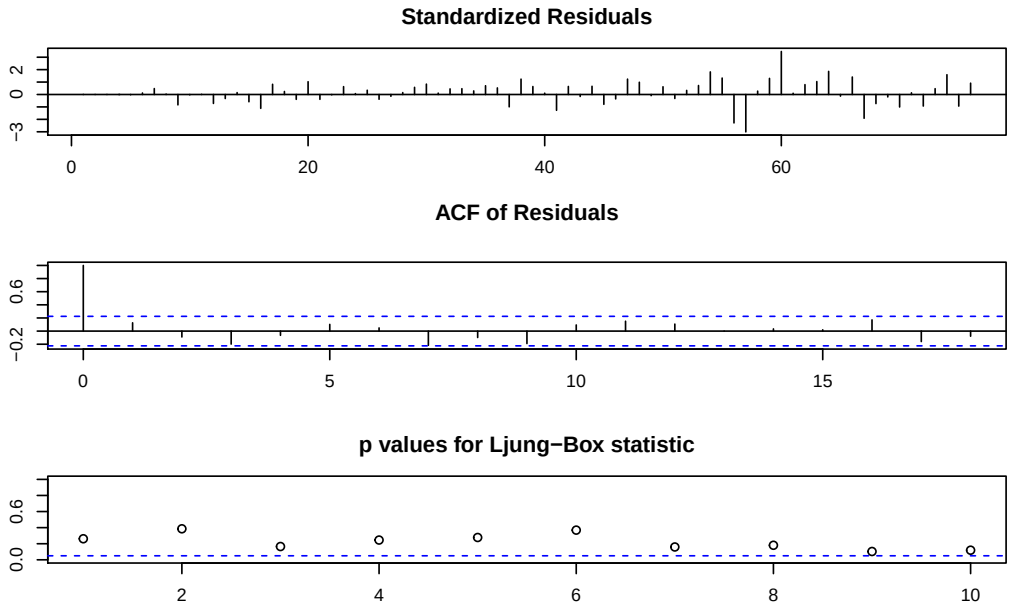


PIB trimestral

```
par(mar = c(3,3,3,1))
tsdiag(est.pibtrim)
```




```
tsdiag(est.pibtrim2)
```



3.6 Previsão

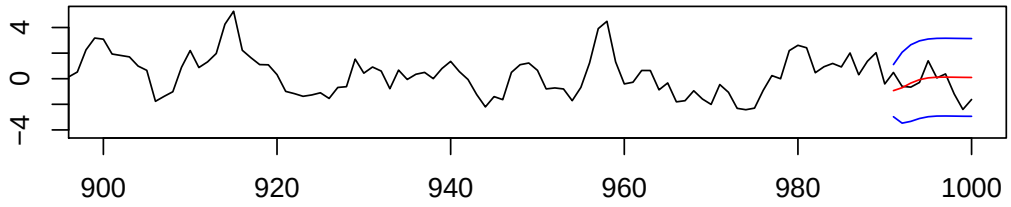
Como última etapa deste exemplo, fazemos a previsão dentro da amostra utilizando a função `predict()` do pacote "forecast". Como argumentos desta função, informamos o nome da série e o número de passos à frente que desejamos prever. Para cada uma das séries, fazemos um gráfico contendo a série original, a previsão e o intervalo de confiança.

Na primeira linha do código abaixo, usamos a função `predict()` para fazer a previsão 10 passos à frente do modelo estimado anteriormente "est.ar2". A esta previsão damos o nome de "previsao.ar2. Se quisermos obter os valores previstos, podemos usar o comando `previsao.ar2$pred` e, para obter os erros padrão, podemos utilizar `previsao.ar2$se`.

Na terceira linha, usamos a função `ts.plot()` para imprimir um gráfico contendo a série de tempo original ("ar2"), e seus valores previstos (`previsao.ar2$pred`). "col=1:2" significa que queremos imprimir a série original e a previsão com cores diferentes, já que elas se sobrepõem (lembre que reservamos uma parte da amostra durante a estimação, para compará-la com a previsão). Usamos "xlim=c(900,1000)" para imprimir apenas as últimas 100 observações da série. Por fim, `lines()` adiciona os intervalos de confiança ao gráfico (dois erros padrão acima e abaixo do valor previsto).

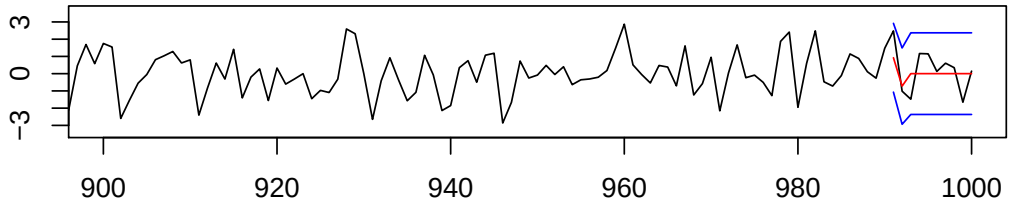
AR(2)

```
previsao.ar2 <- predict(est.ar2, n.ahead=10)
par(mar = c(3, 3, 3, 1))
ts.plot(ar2, previsao.ar2$pred, col=1:2, xlim=c(900,1000), xlab="")
lines(previsao.ar2$pred+2*previsao.ar2$se, col=4)
lines(previsao.ar2$pred-2*previsao.ar2$se, col=4)
```



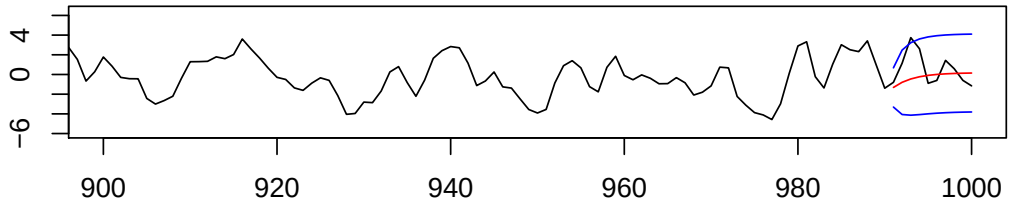
MA(2)

```
previsao.ma2 <- predict(est.ma2, n.ahead=10)
par(mar = c(3, 3, 3, 1))
ts.plot(ma2, previsao.ma2$pred, col=1:2, xlim=c(900,1000), xlab="")
lines(previsao.ma2$pred+2*previsao.ma2$se, col=4)
lines(previsao.ma2$pred-2*previsao.ma2$se, col=4)
```



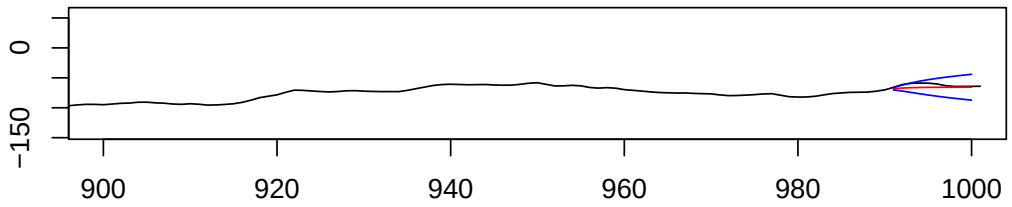
ARMA(1,1)

```
previsao.arma11 <- predict(est.arma11, n.ahead=10)
par(mar = c(3, 3, 3, 1))
ts.plot(arma11, previsao.arma11$pred, col=1:2, xlim=c(900,1000), xlab="")
lines(previsao.arma11$pred+2*previsao.arma11$se, col=4)
lines(previsao.arma11$pred-2*previsao.arma11$se, col=4)
```



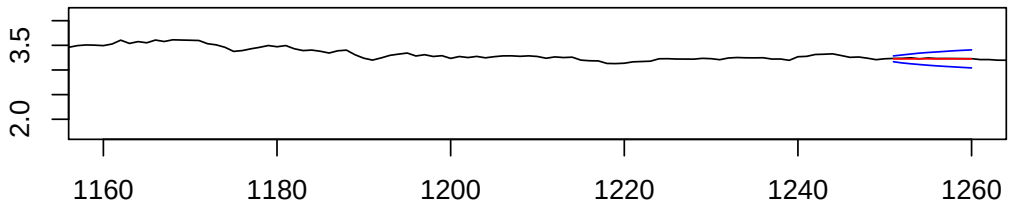
ARIMA(1,1,1)

```
previsao.arima111 <- predict(est.arima111, n.ahead=10)
par(mar = c(3, 3, 3, 1))
ts.plot(arima111, previsao.arima111$pred, col=1:2, xlim=c(900,1000), xlab="")
lines(previsao.arima111$pred+2*previsao.arima111$se, col=4)
lines(previsao.arima111$pred-2*previsao.arima111$se, col=4)
```



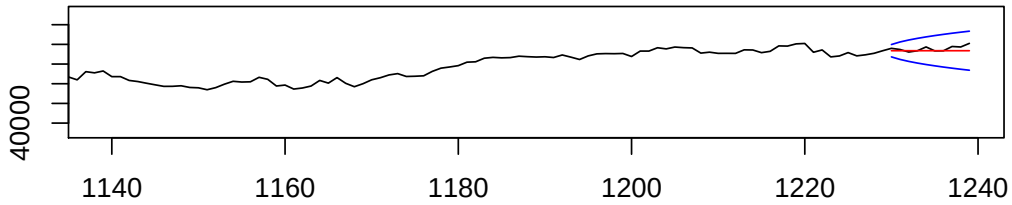
Taxa de câmbio

```
previsao.txcambio <- predict(est.txcambio2, n.ahead=10)
par(mar = c(3, 3, 3, 1))
ts.plot(ts(txcambio), previsao.txcambio$pred, col=1:2, xlim=c(1160,1260), xlab="")
lines(previsao.txcambio$pred+2*previsao.txcambio$se, col=4)
lines(previsao.txcambio$pred-2*previsao.txcambio$se, col=4)
```



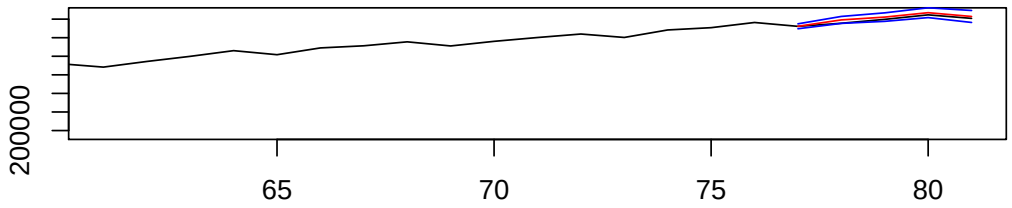
Ibovespa

```
previsao.bvsp <- predict(est.bvsp, n.ahead=10)
par(mar = c(3, 3, 3, 1))
ts.plot(ts(bvsp), previsao.bvsp$pred, col=1:2, xlim=c(1139,1239), xlab="")
lines(previsao.bvsp$pred+2*previsao.bvsp$se, col=4)
lines(previsao.bvsp$pred-2*previsao.bvsp$se, col=4)
```



PIB trimestral

```
previsao.pibtrim <- predict(est.pibtrim, n.ahead=5)
par(mar = c(3, 3, 3, 1))
ts.plot(ts(pibtrim), previsao.pibtrim$pred, col=1:2, xlim=c(61,81), xlab="")
lines(previsao.pibtrim$pred+2*previsao.pibtrim$se, col=4)
lines(previsao.pibtrim$pred-2*previsao.pibtrim$se, col=4)
```



Para maiores detalhes sobre as funções e os pacotes utilizados, sugerimos utilizar os manuais dos pacotes e as funções de ajuda do R, os quais podem ser facilmente acessados através do R Studio.