Séries de Tempo

Aula 4 - Processo não estacionários e raíz unitária

Regis A. Ely

Departamento de Economia
Universidade Federal de Pelotas

07 de agosto de 2020

Conteúdo

Processos não estacionários

- Muitas séries de tempo, especialmente em economia, apresentam características não estacionárias, como séries macroeconômicas, preços de ativos, taxas de câmbio e taxas de juros
- Veremos três casos mais comuns de processos não estacionários:
  1. Passeio aleatório
  2. Processos com tendência determinística
  3. Processos integrados ou com raíz unitária

Passeio aleatório

Um passeio aleatório pode ser descrito como[1]:

$$Y_t = Y_{t-1} + \varepsilon_t = \varepsilon_1 + \varepsilon_2 + \ldots + \varepsilon_t$$

- Sendo ε_t um processo independente e identicamente distribuído com média μ_ε e variância σ_ε^2
- Um passeio aleatório corresponde a um processo AR(1) com $\phi = 1$ e $c = 0$, embora o intercepto c possa assumir outro valor
- Ao resolvermos esta equação em diferença de primeira ordem recursivamente encontramos que um passeio aleatório é uma soma de ruídos brancos com coeficientes iguais a um

[1] A expressão à direita da equação é obtida resolvendo a equação em diferença de primeira ordem e considerando o valor inicial $Y_0 = 0$.

Momentos de um passeio aleatório

Um passeio aleatório possui os seguintes momentos:

- **Valor esperado**: $E(Y_t) = t\mu_\varepsilon$
- **Variância**: $\gamma_0 = t\sigma_\varepsilon^2$
- **Autocovariância**: $\gamma(t1, t2) = \sigma_\varepsilon^2 \min\{t1, t2\}$

Note que todos os momentos deste processo estocástico dependem do tempo t, caracterizando um processo não estacionário

Passeio aleatório com intercepto

Podemos adicionar um intercepto ao passeio aleatório, de modo a transformar ε_t em um ruído branco com média zero[2]:

$$Y_t = \mu + Y_{t-1} + \varepsilon_t = t\mu + \varepsilon_1 + \varepsilon_2 + \ldots + \varepsilon_t$$

- Este modelo é conhecido como passeio aleatório com intercepto (*random walk with drift*)
- O intercepto neste modelo corresponde a inclinação temporal da série de tempo

[2] A expressão à direita da equação é obtida resolvendo a equação em diferença de primeira ordem e considerando o valor inicial $Y_0 = 0$.

Passeio aleatório com intercepto e tendência

Por fim, uma outra alternativa de passeio aleatório inclui intercepto e tendência determinística:

$$Y_t = \mu + Y_{t-1} + \beta t + \varepsilon_t$$

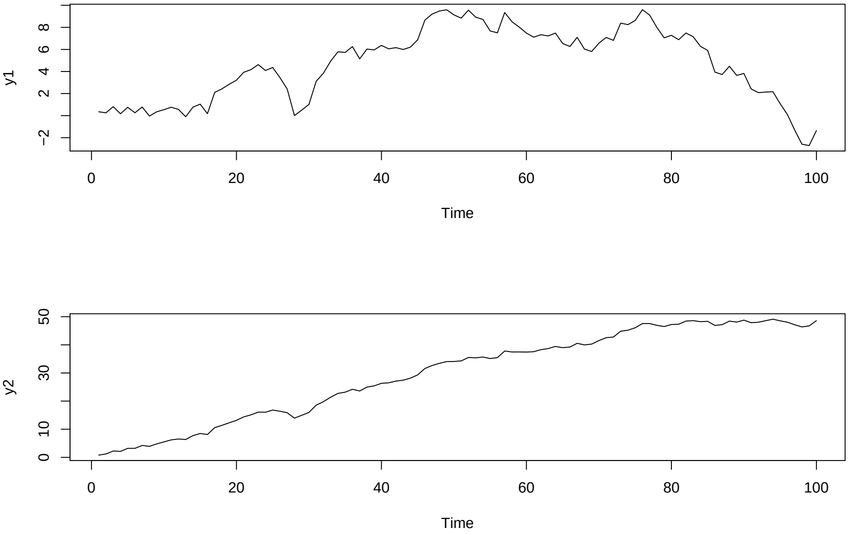
- Este modelo é conhecido como passeio aleatório com intercepto e tendência (*random walk with drift and trend*)

Simulação de passeios aleatórios

Vamos simular um passeios aleatórios com e sem *drift* no `R` através da soma cumulativa de ruídos brancos:

```
set.seed(4210)
e <- rnorm(100)
y1 <- cumsum(e)
y2 <- 0.5 * 1:100 + cumsum(e)
par(mfrow = c(2,1))
ts.plot(y1)
ts.plot(y2)
```

Simulação de passeios aleatórios



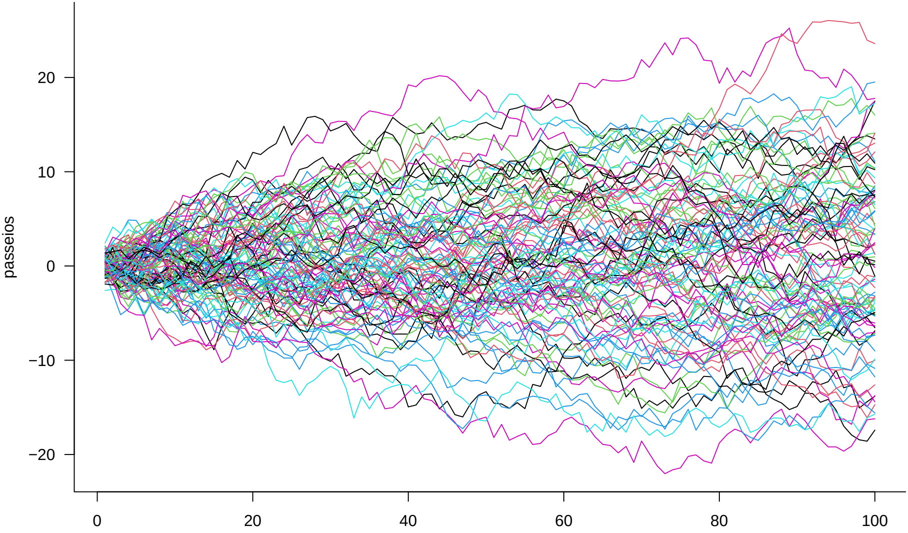
Simulação de passeios aleatórios

Podemos simular 100 passeios aleatórios diferentes no `R` através da função `replicate`:

```
passeios <- replicate(100, cumsum(rnorm(100)))
par(las = 1, bty= "l")
matplot(passeios, type = "l", lty = 1)
```

Passeios aleatórios tem formatos completamente diferentes em cada simulação

Simulação de passeios aleatórios



Diferença de um passeio aleatório

A diferença de um passeio aleatório corresponde a um ruído branco:

$$\Delta Y_t = Y_t - Y_{t-1} = \mu + \varepsilon_t$$

Embora o ruído branco seja estacionário, não há qualquer tipo de regularidade ou autocorrelação a ser modelada, assim dizemos que **um passeio aleatório é imprevisível**

- A dinâmica de preços de ativos costuma ser muito semelhante a um passeio aleatório, sendo difícil a previsão e com isso a criação de estratégias que possibilitem arbitragem nos mercados acionários

Processo com tendência determinística

Um processo com tendência determinística pode ser descrito como:

$$Y_t = c + \beta t + \varepsilon_t$$

Nesse caso temos uma tendência linear[3], de modo que o processo se torna estacionário após a remoção da tendência

- Chamamos este tipo de processo de *estacionário em tendência*

[3] Processos com tendências polinomiais de graus maiores possuem dinâmica semelhante.

Momentos de um processo com tendência determinística

Um processo com tendência determinística possui os seguintes momentos:

- **Valor esperado**: $E(Y_t) = c + \beta t$
- **Variância**: $\gamma_0 = \sigma_\varepsilon^2$
- **Autocovariância**: $\gamma_j = 0$ para $j > 0$

Note que apenas o valor esperado deste processo estocástico depende do tempo t

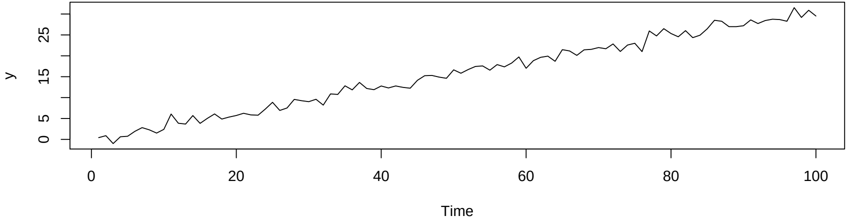
Simulação de um processo com tendência determinística

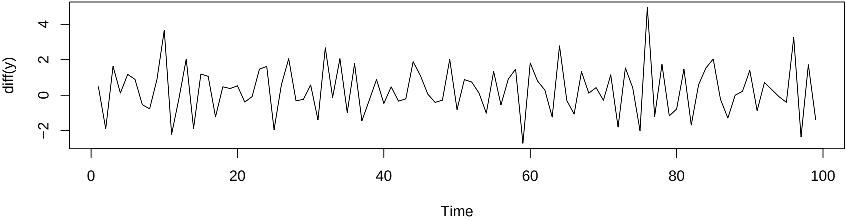
Vamos simular um processo com tendência linear no R e plotar juntamente com a primeira diferença:

```
e <- rnorm(100)
y <- 0.5 + 0.3 * 1:100 + e
par(mfrow = c(2,1))
ts.plot(y)
ts.plot(diff(y))
```

A primeira diferença deste processo remove a tendência linear, $\Delta Y_t = \beta + \varepsilon_t - \varepsilon_{t-1}$

Simulação de um processo com tendência determinística





Processos integrados ou com raíz unitária

- Quando os modelos ARMA têm raízes da equação característica iguais a um, chamamos eles de modelos ARIMA(p,d,q)
- O d corresponde ao número de diferenças necessárias para tornar o processo estacionário em covariância
- Um modelo ARIMA é não estacionário e com raíz unitária quando alguma das raízes da equação característica é igual a um
- Neste caso, os coeficientes da representação MA deste processo não irão decair no tempo, de modo que os choques passados tem efeitos permanentes

Testes de raíz unitária

- Para identificar se um processo possui raíz unitária, podemos utilizar testes estatísticos de hipótese, sendo os mais comuns:
    1. Teste ADF (Augmented Dickey-Fuller)
    2. Teste de Zivot e Andrews
    3. Teste de Phillips-Perron
    4. Teste KPSS (Kwiatkowski-Phillips-Schmidt-Shin)

Séries de tempo econômicas

Nos exemplos a seguir vamos utilizar a base de dados `economics`, que contém dados da economia dos Estados Unidos

- Também vamos carregar alguns pacotes que utilizaremos e definir a base de dados como um `tsibble`

```
library(tidyverse)
library(tsibble)
library(feasts)
econ <- tsibble(economics, index = date)
```

Utilizaremos a série de tempo `unemploy`, que corresponde ao número de desempregados em milhares

Séries de tempo econômicas

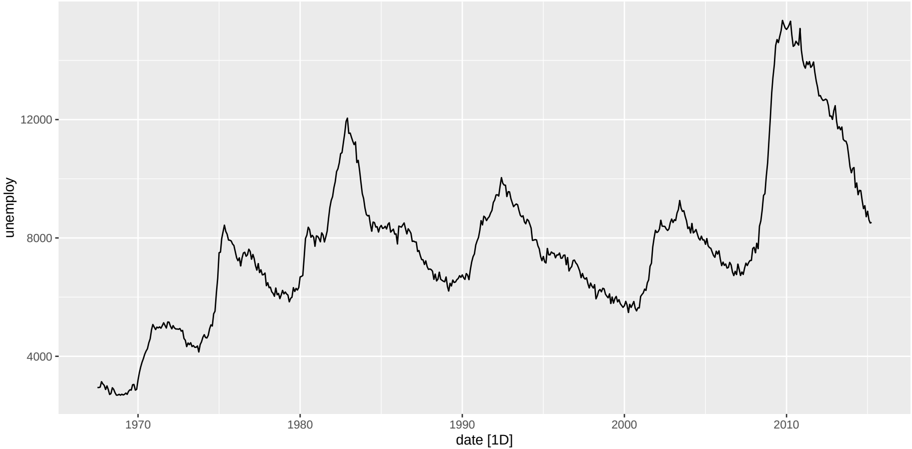
Visualizamos os valores iniciais da base de dados com o comando `head`:

```r
head(econ)
```

```
## # A tsibble: 6 x 6 [1D]
##   date          pce    pop psavert uempmed unemploy
##   <date>      <dbl>  <dbl>   <dbl>   <dbl>    <dbl>
## 1 1967-07-01   507. 198712    12.6     4.5     2944
## 2 1967-08-01   510. 198911    12.6     4.7     2945
## 3 1967-09-01   516. 199113    11.9     4.6     2958
## 4 1967-10-01   512. 199311    12.9     4.9     3143
## 5 1967-11-01   517. 199498    12.8     4.7     3066
## 6 1967-12-01   525. 199657    11.8     4.8     3018
```

Séries de tempo econômicas

```
econ %>% autoplot(unemploy) # Nº de desempregados (mil)
```



Teste ADF

No teste ADF, para verificar a existência de raíz unitária de um processo AR(p) devemos testar $H_0 : \gamma = 0$ (raíz unitária) usando a regressão:

$$\Delta Y_t = c_t + \beta t + \gamma Y_{t-1} + \sum_{i=1}^{p-1} \Delta Y_{t-i} + \varepsilon_t$$

Assim, uma vez definido o número de defasagens p[4], o teste ADF será dado por:

$$ADF = \frac{\hat{\gamma}}{std(\hat{\gamma})}$$

Onde $\hat{\gamma}$ é a estimativa do coeficiente γ na regressão acima

[4]Utilizamos critérios como o de Akaike para definir o número de lags.

Teste ADF

No R podemos utilizar o pacote urca para calcular o teste ADF[5]:

```
library(urca)
adf_test <- ur.df(
  log(econ$unemploy), type = "none", selectlags = "AIC"
)
summary(adf_test)
```

No argumento type é possível incluir um intercepto ou uma tendência na regressão estimada através das opções drift ou trend[6]

[5]Vamos aplicar o logaritmo natural nesta série antes para suavizar a variância.

[6]A opção trend inclui tanto um intercepto quanto uma constante, enquanto a opção none não inclui nenhum dos dois termos.

Teste ADF

```
...
## Call:
## lm(formula = z.diff ~ z.lag.1 - 1 + z.diff.lag)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.081534 -0.017945 -0.001709  0.017269  0.111692
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## z.lag.1    0.0001700  0.0001323   1.285  0.19942
## z.diff.lag 0.1202234  0.0415900   2.891  0.00399 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02813 on 570 degrees of freedom
## Multiple R-squared:  0.01813,    Adjusted R-squared:  0.01469
## F-statistic: 5.263 on 2 and 570 DF,  p-value: 0.005433
##
##
## Value of test-statistic is: 1.2847
##
## Critical values for test statistics:
##       1pct  5pct 10pct
## tau1 -2.58 -1.95 -1.62
...
```

Teste de Zivot e Andrews

O teste de Zivot e Andrews é um teste de raíz unitária robusto a quebras estruturais e estima a seguinte regressão:

$$\Delta Y_t = c + \beta t + \alpha Y_{t-1} + \gamma DU_t + \theta DT_t + \sum_{i=1}^{p} d_j \Delta Y_{t-j} + \varepsilon_t$$

Onde DU_t é uma *dummy* para a mudança na média e DT_t é uma *dummy* para a mudança na tendência, sendo a estatística do teste obtida a partir da estimativa $\hat{\alpha}$

- A hipótese nula é a de raíz unitária

Teste de Zivot e Andrews

No `R`, o teste está presente no pacote `urca` através da função `ur.za`:

```
za_test <- ur.za(
  log(econ$unemploy),  model = "both", lag = 1
)
summary(za_test)
```

Neste teste também podemos escolher entre acrescentar um intercepto, uma tendência ou ambos através do argumento `model`, e o número de lags da regressão através do argumento `lag`

Teste de Zivot e Andrews

```
...
## Call:
## lm(formula = testmat)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.081463 -0.018411 -0.000822  0.016257  0.115192
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.272e-01  3.882e-02   3.276  0.00112 **
## y.l1         9.858e-01  4.536e-03 217.306  < 2e-16 ***
## trend       -7.272e-07  1.014e-05  -0.072  0.94287
## y.dl1        6.704e-02  4.151e-02   1.615  0.10679
## du           3.669e-02  6.924e-03   5.300 1.67e-07 ***
## dt          -6.778e-04  1.275e-04  -5.317 1.52e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02731 on 566 degrees of freedom
##   (2 observations deleted due to missingness)
## Multiple R-squared:  0.9941, Adjusted R-squared:  0.9941
## F-statistic: 1.921e+04 on 5 and 566 DF,  p-value: < 2.2e-16
##
##
## Teststatistic: -3.1314
## Critical values: 0.01= -5.57 0.05= -5.08 0.1= -4.82
##
...
```

Teste de Phillips-Perron

- O teste de Phillips-Perron é um teste de raíz unitária não paramétrico
- É construído a partir de uma correção do teste ADF
- É robusto a autocorrelação mal especificada e heteroscedasticidade dos erros
- A hipótese nula é a de raíz unitária e a hipótese alternativa é a de estacionariedade

Teste de Phillips-Perron

Para estimar o teste de Phillips-Perron podemos utilizar a função `features` aliada a função `unitroot_pp` do pacote `feasts`[7]

```
econ %>%
  features(log(unemploy), unitroot_pp)
```

```
## # A tibble: 1 x 2
##   pp_stat pp_pvalue
##     <dbl>     <dbl>
## 1   -2.52       0.1
```

[7] É essencial definirmos nossa base de dados como um `tsibble` antes de utilizarmos estas funções.

Teste KPSS

- O teste KPSS tem **hipótese nula de estacionariedade** e hipótese alternativa de raíz unitária
- Inicialmente, o teste decompõe uma série em três componentes aditivos:
  1. Um componente de tendência
  2. Um componente de passeio aleatório
  3. Um ruído branco
- Os resíduos de uma regressão de Y_t explicado por estes componentes (e_t) são utilizados para a construção do teste:

$$KPSS = \sum_{t=1}^{T} \frac{\sum_{t=1}^{t} e_t^2}{T^2 \hat{\sigma}_e^2}$$

Teste KPSS

Para estimar o teste KPSS podemos utilizar a função `features` aliada a função `unitroot_kpss` do pacote `feasts`[8]

```
econ %>%
  features(log(unemploy), unitroot_kpss)
```

```
## # A tibble: 1 x 2
##   kpss_stat kpss_pvalue
##       <dbl>       <dbl>
## 1      3.52        0.01
```

[8] É essencial definirmos nossa base de dados como um `tsibble` antes de utilizarmos estas funções.

Testes de raíz unitária nas diferenças

O teste KPSS é utilizado como teste padrão para identificação de raíz unitária nos pacotes `feasts` e `fable`[9], sendo que a função `unitroot_ndiffs` já nos dá o número de diferenças necessárias para tornar a série estacionária

```
econ %>%
  features(log(unemploy), unitroot_ndiffs)
```

```
## # A tibble: 1 x 1
##   ndiffs
##    <int>
## 1      1
```

[9]Utilizaremos este pacote para estimação de modelos ARIMA.

Testes de raíz unitária nas diferenças

Vamos tirar a primeira diferença de `unemploy` e plotar ambas as séries

```
econ <- econ %>%
  mutate(var_unemploy = difference(log(unemploy)))
econ %>%
  slice(-1) %>%
  pivot_longer(
    c(unemploy, var_unemploy),
    names_to = "Variável",
    values_to = "Valor"
  ) %>%
  autoplot(Valor) +
  facet_wrap("Variável", scales = "free", ncol = 1) +
  theme(legend.position = "none") +
  xlab("")
```

Testes de raíz unitária nas diferenças

Algumas observações sobre o comando anterior:

1. As diferenças são calculadas pela função `difference`[10]
2. Passamos o logaritmo natural na série antes de calcularmos a diferença
3. Removemos a primeira observação com o comando `slice` pois perdemos um grau de liberdade ao calcular a primeira diferença
4. Organizamos os dados em formato de painel antes de plotarmos através da função `pivot_longer`
5. Criamos dois painéis diferenças com escalas livres através da função `facet_wrap`

[10] O argumento `differences` especifica o número diferenças, sendo uma diferença o padrão.

Testes de raíz unitária nas diferenças

Testes de raíz unitária nas diferenças

Podemos agora realizar o teste KPSS na primeira diferença da série de tempo para confirmar que a série é estacionária:

```
econon %>%
  features(var_unemploy, unitroot_ndiffs)
```

```
## # A tibble: 1 x 1
##   ndiffs
##    <int>
## 1      0
```

Raíz unitária sazonal

- Um outro tipo de raíz unitária é conhecido como **raíz unitária sazonal**, quando observamos persistência dos choques nos períodos sazonais
- Nesse caso, para removermos a raíz unitária é necessário tirar a s-ésima diferença, onde s é o período sazonal (*Ex*: 12 meses, 7 dias, etc)
- A função `unitroot_nsdiffs` pode ser utilizada para identificar o número de diferenças sazonais necessárias
- Esta função utiliza uma medida de persistência sazonal calculada através do modelo de decomposição STL

Raíz unitária sazonal

Podemos observar que não é necessária nenhuma diferença sazonal na série do número de desempregados dos Estados Unidos:

```
econ %>%
  features(log(unemploy), unitroot_nsdiffs)
```

```
## # A tibble: 1 x 1
##   nsdiffs
##     <int>
## 1       0
```