Hayashi

A Language for Applied Econometrics

Flávio de Vasconcellos Corrêa*

Versão 0.2.8-dev

9 de julho de 2026

*Mestre em Economia Aplicada pelo Programa de Pós-Graduação em Organizações e Mercados (PPGOM), Universidade Federal de Pelotas (UFPel).

Prefácio

Hayashi nasceu de uma insatisfação concreta: as ferramentas existentes para econometria aplicada exigem que o pesquisador escolha entre expressividade e desempenho, entre uma sintaxe próxima da notação estatística e a capacidade de automatizar análises complexas.

Stata tem a sintaxe certa, mas é proprietário, fechado e não é uma linguagem de propósito geral. R tem o ecossistema certo, mas a sintaxe é inconsistente e o modelo de dados é surpreendente para quem vem de Stata. Python tem a flexibilidade certa, mas os pacotes estatísticos tratam econometria como subcaso da aprendizagem de máquina.

Hayashi é uma linguagem interpretada, de propósito geral, construída especificamente para análise econométrica. A sintaxe homenageia Stata nos lugares certos e adota idiomas modernos onde Stata mostra sua idade. O motor é escrito em Rust — sem máquina virtual, sem *garbage collector*, sem tempo de inicialização perceptível.

Este livro documenta tanto a linguagem quanto o núcleo econométrico: da estrutura de tipos e controle de fluxo aos estimadores de séries temporais, inferência causal e métodos avançados. É um retrato do estado atual do projeto — um documento vivo, que evolui com o software.

Um projeto de um desenvolvedor — e de toda a comunidade

Hayashi é, até o momento, obra de um único desenvolvedor. Isso tem consequências práticas que o leitor deve conhecer.

Do lado positivo, há coerência de design: cada decisão de sintaxe, cada interface de estimador, cada mensagem de erro foi tomada com o mesmo conjunto de valores

em mente. Não há comitês, não há *bikeshedding*, não há camadas de compatibilidade acumuladas por décadas. O código é enxuto porque ninguém adicionou nada "só para não perder".

Do lado da limitação, um par de olhos — por mais criterioso que seja — não vê tudo. O projeto conta com três camadas de verificação: testes unitários cobrindo os primitivos da linguagem; scripts de integração que exercitam a Hayashi como caixa preta; e scripts *DGP* (processo gerador de dados) para cada estimador, onde o parâmetro verdadeiro é conhecido e a recuperação pode ser verificada numericamente. Ainda assim, casos de borda existem, e alguns certamente escaparam.

Licença. Hayashi é software livre, distribuído sob os termos da **GNU General Public License versão 3** (GPL v3). Isso significa que você pode usar, estudar, modificar e redistribuir o programa — inclusive para fins comerciais — desde que qualquer trabalho derivado seja igualmente livre. O código-fonte é público e pertence, no sentido mais literal, a quem o usa.

A escolha da GPL v3 não é acidental. Ferramentas econométricas proprietárias criam dependência institucional: universidades e laboratórios ficam reféns de licenças, de políticas de preço e de decisões corporativas sobre o que o software pode ou não fazer. Software livre quebra esse ciclo. Hayashi deve ser acessível ao pesquisador em qualquer lugar do mundo, independentemente de financiamento.

Colaboradores e testadores são bem-vindos. Se você encontrou um resultado inesperado, um estimador que não converge quando deveria, uma mensagem de erro obscura ou simplesmente uma sintaxe que poderia ser mais clara — isso é informação valiosa. Abra uma *issue* no repositório. Se quiser ir além, uma correção bem testada ou um novo capítulo de documentação é contribuição tão legítima quanto qualquer linha de código Rust.

O projeto precisa especialmente de pessoas dispostas a testar Hayashi em dados reais: séries de tempo com sazonalidade, painéis desbalanceados, modelos com multicolinearidade, dados com missings. Esses são os casos que os scripts DGP não cobrem, porque dados sintéticos são, por construção, bem-comportados.

Hayashi só se torna robusto com uso. E uso honesto — que inclui reportar o que quebra — é a forma mais direta de colaborar.

Junho de 2026.

Sumário

Parte I

A Linguagem

Capítulo 1

Introdução

1.1 O que é Hayashi

Hayashi é uma linguagem de programação interpretada projetada para análise econométrica aplicada. O nome homenageia Fumio Hayashi, cujo livro *Econometrics* (2000) é referência canônica na área.

A linguagem tem três objetivos centrais:

1. **Sintaxe familiar** para quem usa Stata — comandos como `generate`, `summarize`, `regress` funcionam como esperado.
2. **Programação de propósito geral** — funções, closures, estruturas de controle, módulos. Não é apenas um conjunto de comandos estatísticos.
3. **Desempenho sem configuração** — o interpretador é escrito em Rust; não há JVM, não há GC, não há dependências de runtime.

1.2 Filosofia de design

Semântica clara antes de sintaxe concisa

Quando uma operação muta um objeto, ela deve ser visivelmente imperativa. Quando retorna um valor novo, deve parecer funcional. Hayashi distingue essas duas formas de maneira consistente:

```
// imperativo: altera df no lugar (Stata-style)
generate df z = x^2

// funcional: retorna novo DataFrame, df inalterado
```

```
let df2 = generate(df, z = x^2)
```

Listing 1.1: Forma imperativa × forma funcional

Copy-on-Write para DataFrames

DataFrames usam semântica de *copy-on-write* (COW): atribuir um DataFrame a outra variável cria um alias barato. A cópia real só acontece no momento da mutação. Isso permite compartilhamento eficiente sem surpresas:

```
load "dados.csv" as df
let baseline = df       // alias, sem cópia

let transformado = df |> generate(z = x^2) |> filter(z > 10)
// baseline ainda não tem coluna z
```

Listing 1.2: COW em ação

Mensagens de erro como documentação

Erros em HAYASHI incluem o trecho de código que os causou, o número da linha e, quando possível, uma sugestão de correção (*did you mean?*). Um erro bem formatado é mais útil que um manual.

1.3 Hayashi e Stata

HAYASHI não é um clone de Stata. A sintaxe de comandos de dados é inspirada em Stata porque essa sintaxe é boa — próxima da linguagem que economistas usam naturalmente. Mas HAYASHI é uma linguagem completa: tem funções de primeira classe, closures, recursão, tratamento de erros estruturado.

A tabela abaixo resume as diferenças centrais:

Aspecto	**Stata**	**Hayashi**
Licença	Proprietário	GPL-3.0
Funções	Limitadas	Primeira classe
Closures	Não	Sim
Tipos	Implícitos	Explícitos em funções
DataFrames	Um por vez	Múltiplos, COW
Modularização	`do`	`source`
Desempenho	C	Rust

1.4 Hayashi e R / Python

R e Python têm ecossistemas imensos. HAYASHI não compete com eles no volume de pacotes disponíveis — compete na clareza da sintaxe para a tarefa específica de econometria aplicada. Um script de regressão com dados de painel em HAYASHI tem menos ruído visual que o equivalente em qualquer outra linguagem.

1.5 Programa de validação empírica

HAYASHI inclui um programa de validação reprodutível no diretório `validation/`. Cada caso executa um script HAYASHI e o compara, dentro de tolerâncias declaradas, com implementações de referência em R e Python (statsmodels). Os casos usam tanto datasets reais públicos quanto DGPs simulados retirados dos capítulos deste livro.

Para executar a validação completa:

```
python -m venv validation/.venv
validation/.venv/bin/pip install -r validation/requirements.
    txt
hay validate
```

Listing 1.3: Executar o programa de validação

O estado atual de cada caso está registrado em `validation/MATRIX.md`.

1.6 Estrutura deste livro

Este volume cobre a Parte I: a linguagem em si.

Capítulos 2–3 Instalação e uso do REPL.

Capítulos 4–12 Sistema de tipos, variáveis, operadores, controle de fluxo, funções, erros, strings, listas e dicionários.

Capítulos 13–15 DataFrames, manipulação de dados e o operador pipe.

Capítulo 16 Modularização com `source`.

A Parte II cobrirá estatística descritiva, estimadores de OLS, IV, painel, séries de tempo, modelos de variável dependente limitada e testes de hipótese.

Capítulo 2

Instalação

2.1 Requisitos

HAYASHI é distribuído como binário estático para Linux, macOS e Windows. Não há dependências de runtime: o executável `hay` é autossuficiente.

- Linux x86-64 ou ARM64
- macOS 12+ (Intel ou Apple Silicon)
- Windows 10+ (x86-64)

2.2 Via Cargo (recomendado)

Se Rust está instalado:

```
cargo install hayashi-lang
```

O binário `hay` será instalado em `$HOME/.cargo/bin`. Confirme que esse diretório está no `PATH`.

```
hay --version
```

2.3 Binários pré-compilados

Disponíveis na página de releases do repositório: https://github.com/sheep-farm/hayashi/releases

Baixe o arquivo correspondente à sua plataforma, descompacte e coloque o binário `hay` em algum diretório do `PATH`.

2.4 Compilação a partir do código-fonte

```
git clone https://github.com/sheep-farm/hayashi
cd hayashi
cargo build --release
# binário em target/release/hay
```

2.5 Verificando a instalação

```
hay --version
hay --help
```

Para executar um script:

```
hay meu_script.hay
```

Para entrar no REPL interativo:

```
hay
```

Nota

No Arch Linux com Omarchy/Hyprland, `$HOME/.cargo/bin` já costuma estar no `PATH` se o shell foi configurado com `rustup`. Caso contrário, adicione ao `.bashrc` ou `.zshrc`: `export PATH="$HOME/.cargo/bin:$PATH"`

Capítulo 3

O REPL

3.1 O que é o REPL

REPL (*Read-Eval-Print Loop*) é o modo interativo de Hayashi. Cada linha digitada é avaliada imediatamente e o resultado é exibido. É o ambiente ideal para explorar dados, testar expressões e prototipar análises.

```
hay
```

```
Hayashi 0.2.8-dev -  Applied Econometrics Language
In honor of Fumio Hayashi. Type 'exit' or Ctrl-D to quit.

hay>
```

3.2 Saindo do REPL

Comando	Efeito
`exit`	Sai do REPL
`Ctrl-D`	Sai do REPL (EOF)
`Ctrl-C`	Cancela a linha atual

3.3 Navegação e histórico

O REPL usa `rustyline` e suporta:

- Seta para cima/baixo: histórico de comandos
- `Ctrl-R`: busca no histórico

- `Ctrl-A` / `Ctrl-E`: início/fim da linha
- `Ctrl-C`: cancela a linha atual
- `Ctrl-D`: sai do REPL (equivalente a `exit`)

O histórico é persistido entre sessões em `$HOME/.hay_history`.

3.4 Expressões no REPL

Qualquer expressão válida pode ser digitada diretamente:

```
hay> 2 + 2
4

hay> let x = 10
hay> x * 3
30

hay> let s = "olá, mundo"
hay> s
"olá, mundo"
```

Listing 3.1: Sessão interativa básica

3.5 Executando scripts

Scripts `.hay` são arquivos de texto com uma instrução por linha. Para executar:

```
hay analise.hay
```

A saída vai para `stdout`. Erros vão para `stderr`.

3.6 Formato de erros

HAYASHI exibe erros com contexto visual:

```
error: line 3: undefined variable 'y' - did you mean 'x'?
   3   display y + 1
           ^^^^^^^^^^^^
```

O trecho de código, a linha e uma sugestão (quando disponível) facilitam a correção sem precisar contar linhas manualmente.

Capítulo 4

Tipos e Valores

Hayashi é dinamicamente tipado: variáveis não têm tipo declarado, mas cada valor carrega seu tipo em tempo de execução. O sistema de tipos é coerente — operações entre tipos incompatíveis produzem erros claros em vez de coerções silenciosas.

4.1 Tipos primitivos

4.1.1 Int

Inteiro de 64 bits com sinal. Literais inteiros são escritos sem ponto decimal:

```
let n = 42
let negativo = -7
let grande = 1_000_000    // _ como separador visual
```

4.1.2 Float

Ponto flutuante de 64 bits (IEEE 754). Requer ponto decimal ou notação científica:

```
let pi = 3.14159
let e  = 2.71828
let pequeno = 1.5e-10
```

Operações aritméticas entre `Int` e `Float` promovem para `Float`:

```
let x = 3 / 2      // 1  (divisão inteira)
let y = 3 / 2.0    // 1.5
let z = 3.0 / 2    // 1.5
```

Divisão de `Float` por zero retorna `inf` ou `-inf`, nunca um erro — seguindo IEEE 754:

```
display 1.0 / 0.0     // inf
display -1.0 / 0.0    // -inf
display 0.0 / 0.0     // nan
```

Nota

Divisão de `Int` por zero produz erro de runtime. Divisão de `Float` por zero segue IEEE 754 e retorna `inf`.

4.1.3 Bool

Verdadeiro ou falso. Literais: `true` e `false`.

```
let ativo = true
let encerrado = false
```

Operadores lógicos: `and`, `or`, `not`.

4.1.4 Nil

Ausência de valor. Algumas funções retornam `nil` quando não há resultado relevante (por exemplo, `push` que muta uma lista no lugar).

```
let nada = nil
display nada     // nil
```

4.2 Strings

Strings são sequências de caracteres Unicode delimitadas por aspas duplas:

```
let nome = "Hayashi"
let caminho = "/home/usuario/dados.csv"
```

Sequências de escape: `\n` (nova linha), `\t` (tab), `\"` (aspas), `\\` (barra).

F-strings permitem interpolação de expressões:

```
let ano = 2026
display f"Publicado em {ano}"    // Publicado em 2026
```

4.3 Listas

Coleções ordenadas de valores de qualquer tipo, delimitadas por colchetes:

```
let numeros = [1, 2, 3, 4, 5]
let misto   = [1, "dois", true, nil]
let vazia   = []
```

4.4 Dicionários

Mapeamentos chave-valor. Chaves e valores podem ser de qualquer tipo:

```
let config = {"host": "localhost", "porta": 5432}
let vazio  = {}
```

Acesso por chave:

```
config["host"]      // "localhost"
```

4.5 DataFrame

O tipo central de Hayashi. Um DataFrame é uma tabela bidimensional com colunas nomeadas. Cada coluna é um vetor de `Float` (valores ausentes representados por `NaN`).

```
input x y
  1  2
  3  4
  5  6
end
```

Listing 4.1: Criando um DataFrame inline

DataFrames são discutidos em profundidade no Capítulo 13.

4.6 Conversões

Hayashi não faz coerção implícita entre tipos. Conversões devem ser explícitas:

```
let s = str(42)        // "42"
let n = int("7")       // 7
let f = float("3.14")  // 3.14
```

Tentar somar um `Int` com uma `String` produz erro de tipo:

```
42 + "3"
// error: type mismatch - expected Int or Float, got String
```

4.7 Verificação de Tipos

Para verificar dinamicamente o tipo de um valor, HAYASHI oferece funções embutidas de verificação que retornam um booleano:

```
is_nil(nil)          // true
is_int(42)           // true
is_float(3.14)       // true
is_bool(true)        // true
is_str("ola")        // true  (ou is_string)
is_list([1, 2])      // true
is_dict({"a": 1})    // true
is_df(df)            // true  (ou is_dataframe)
is_fn(|x| x)         // true  (ou is_function)
```

Além disso, a função `type(x)` (ou `typeof(x)`) retorna o tipo do valor como uma string (ex: `"int"`, `"float"`, `"nil"`, etc.).

Capítulo 5

Variáveis e Escopos

5.1 Declaração com `let`

Toda variável é declarada com `let`. A declaração vincula um nome a um valor:

```
let x = 10
let nome = "João"
let ativo = true
```

`let` sem atribuição inicial não existe em HAYASHI — toda variável é inicializada no momento da declaração.

Redeclaração

Uma variável pode ser redeclarada com `let` no mesmo escopo. O novo valor substitui o anterior sem afetar outras variáveis que apontem para o valor antigo:

```
let x = 5
let y = x       // y aponta para 5
let x = 99      // x agora é 99; y continua 5
display y       // 5
```

5.2 Constantes com `const`

`const` declara um nome que não pode ser redeclarado nem atribuído no mesmo escopo. Útil para valores de configuração ou parâmetros fixos:

```
const ALPHA = 0.05
const NOME_PROJETO = "LAFI"
```

Tentar atribuir a uma constante produz erro de compilação:

```
const N = 100
N = 200    // error: cannot assign to constant 'N'
```

Nota

`const` em HAYASHI é avaliado em tempo de execução, não em tempo de compilação. A distinção de `let` é puramente de mutabilidade de ligação, não de desempenho.

5.3 Atribuição

Após declaração com `let`, a variável pode ser reatribuída com `=`:

```
let contador = 0
contador = contador + 1
contador = contador + 1
display contador    // 2
```

Operadores compostos de atribuição:

```
let n = 10
n += 5      // n = 15
n -= 3      // n = 12
n *= 2      // n = 24
n /= 4      // n = 6
```

5.4 Escopos

Escopos são delimitados por blocos `\{ \}`. Variáveis declaradas dentro de um bloco existem apenas até o final dele:

```
let x = 1

{
  let y = 2
  display x    // 1 -  x é visível aqui
  display y    // 2
}

display x    // 1
```

```
display y    // error: undefined variable 'y'
```

Listing 5.1: Variáveis locais a um bloco

Escopos em funções

Cada chamada de função cria um novo escopo. Parâmetros e variáveis locais são destruídos quando a função retorna:

```
fn soma(a, b) {
  let resultado = a + b
  return resultado
}

display soma(3, 4)     // 7
display resultado      // error: undefined variable 'resultado
    '
```

Captura de escopo exterior (closures)

Funções anônimas (closures) capturam variáveis do escopo onde foram definidas:

```
let base = 100

let adicionar = fn(x) {
  return x + base    // captura 'base' do escopo exterior
}

display adicionar(20)    // 120
```

5.5 Variáveis especiais

`_n` e `_N` em contextos de dados

Dentro de `generate` e `mutate`, duas variáveis implícitas estão disponíveis:

- `_n`: índice da linha atual (começando em 1)
- `_N`: número total de linhas do DataFrame

```
input x
  10  20  30
end

generate df id = _n        // [1, 2, 3]
generate df peso = 1/_N    // [0.333, 0.333, 0.333]
```

Listing 5.2: Uso de _n e _N

_ (descarte)

O identificador _ descarta um valor sem criar uma ligação:

```
for _ in 1..5 {
  display "iteração"
}
```

5.6 Blocos expressão

Um bloco pode ser usado como expressão. Ele avalia uma sequência de statements e retorna o valor da última expressão. Variáveis declaradas dentro do bloco são locais a ele:

```
let df = {
  let raw = load("dados.csv")
  generate raw y = log(x)
  keep(raw, ["date", "y"])
  raw
}
```

Listing 5.3: Bloco expressão retornando um valor

Após o bloco, `df` está disponível, mas `raw` não.

5.7 Controle de saída

`quietly on` suprime a saída automática de statements e estimadores. `print(...)` e `display ...` continuam aparecendo. `quietly off` restaura a saída normal. A flag respeita escopo:

```
quietly on

let df = {
  let a = load("a.csv")
  let b = load("b.csv")
  let m = merge(a, b, key=id, type=inner)
  generate m z = x - y
  m
}

quietly off

ols(z ~ x, df)
print("pronto")
```

Listing 5.4: quietly on/off com escopo

Dentro do bloco, `quietly off` habilitaria temporariamente a saída; ao sair do bloco, ela volta para `quietly on`.

Capítulo 6

Operadores

6.1 Aritméticos

Operador	Operação	Exemplo
`+`	Adição	`3 + 2` → `5`
`-`	Subtração	`5 - 3` → `2`
`*`	Multiplicação	`4 * 3` → `12`
`/`	Divisão	`7 / 2` → `3` (inteiros)
`^`	Potência	`2^10` → `1024`
`%`	Módulo	`7 % 3` → `1`

Divisão entre dois `Int` é inteira (truncamento em direção a zero). Para divisão real, ao menos um operando deve ser `Float`:

```
7 / 2      // 3
7 / 2.0    // 3.5
7.0 / 2    // 3.5
```

6.2 Comparação

Operador	Significado	Exemplo
`==`	Igual	`3 == 3` → `true`
`!=`	Diferente	`3 != 4` → `true`
`>`	Maior	`5 > 3` → `true`
`<`	Menor	`3 < 5` → `true`
`>=`	Maior ou igual	`3 >= 3` → `true`
`<=`	Menor ou igual	`2 <= 3` → `true`

6.3 Lógicos

```
true and false     // false
true or false      // true
not true           // false
```

`and` e `or` têm avaliação em curto-circuito: o segundo operando não é avaliado se o resultado já é determinado pelo primeiro.

6.4 Operador `in`

Verifica pertencimento a uma lista:

```
let regioes = ["norte", "sul", "centro"]
"sul" in regioes       // true
"leste" in regioes     // false
```

Funciona também com ranges:

```
5 in 1..10     // true
15 in 1..10    // false
```

6.5 Operador pipe `|>`

O operador `|>` passa o valor da esquerda como primeiro argumento da função à direita:

```
// sem pipe
display filter(df, x > 5)

// com pipe - mais legível
df |> filter(x > 5) |> display
```

O pipe tem semântica especial com DataFrames: quando usado sem atribuição explícita (`let`), o resultado é atribuído de volta à variável de origem:

```
let df = load("dados.csv")

// atualiza df no lugar
df |> filter(x > 0) |> sort(x)

// cria df2, df permanece inalterado
```

```
let df2 = df |> filter(x > 0)
```

Este comportamento é discutido em detalhe no Capítulo 15.

6.6 Precedência

Da menor para a maior prioridade:

1. `or`
2. `and`
3. `not`
4. `== != > < >= <=`
5. `+ -`
6. `* / %`
7. `^` (associa à direita)
8. Unário `-`
9. Chamada de função, indexação, acesso a campo

Use parênteses para clareza quando a precedência não for óbvia:

```
2 + 3 * 4       // 14  (multiplicação primeiro)
(2 + 3) * 4     // 20
```

Capítulo 7

Controle de Fluxo

7.1 `if` / `else`

```
let nota = 7.5

if nota >= 7 {
  display "aprovado"
} else if nota >= 5 {
  display "recuperação"
} else {
  display "reprovado"
}
```

Listing 7.1: Estrutura básica de if/else

`if` como expressão

`if` pode ser usado como expressão — retorna o valor do bloco escolhido:

```
let status = if nota >= 7 { "aprovado" } else { "reprovado" }
display status
```

Quando usado como expressão, os dois ramos devem ter o mesmo tipo (ou ambos `nil`).

7.2 `match`

`match` compara um valor contra múltiplos padrões. O primeiro padrão que casa é executado:

```
let codigo = 2

match codigo {
  1 => display "iniciante"
  2 => display "intermediário"
  3 => display "avançado"
  _ => display "desconhecido"
}
```

Listing 7.2: match com literais

O padrão _ é o caso padrão (*wildcard*) — casa com qualquer valor.

`match` com strings

```
let uf = "RS"

match uf {
  "RS" => display "Rio Grande do Sul"
  "SP" => display "São Paulo"
  "RJ" => display "Rio de Janeiro"
  _    => display f"UF desconhecida: {uf}"
}
```

`match` como expressão

```
let descricao = match codigo {
  1 => "iniciante"
  2 => "intermediário"
  _ => "avançado"
}
```

Múltiplos valores por braço

```
match codigo {
  1 | 2    => display "básico"
  3 | 4    => display "avançado"
  _        => display "outro"
}
```

7.3 Bloco de expressão

Um bloco `\{ \}` é uma expressão que avalia para o último valor produzido dentro dele:

```
let resultado = {
  let a = 10
  let b = 20
  a + b       // este é o valor do bloco
}
display resultado     // 30
```

Capítulo 8

Laços

8.1 `for`

Itera sobre uma sequência. A variável de iteração é visível dentro do bloco e *persiste* após o laço com o último valor assumido:

```
for i in 1..4 {
  display i    // 1, 2, 3  (4 não incluído)
}
display i    // 3 - variável persiste após o laço
```

Listing 8.1: for sobre range

Range exclusivo e inclusivo

HAYASHI segue a mesma convenção de Rust:

```
for i in 1..3  { }    // 1, 2      (exclusivo: não inclui o
   fim)
for i in 1..=3 { }    // 1, 2, 3   (inclusivo: inclui o fim)
```

Ranges também são expressões que avaliam para listas — ver Seção 12.2.

Iterando sobre listas

```
let frutas = ["maçã", "banana", "uva"]

for fruta in frutas {
  display fruta
}
```

Descartando a variável de iteração

Quando o índice não importa:

```
for _ in 1..3 {
  display "repetição"
}
```

8.2 `while`

Executa enquanto a condição for verdadeira:

```
let n = 1

while n <= 5 {
  display n
  n += 1
}
```

Listing 8.2: while básico

Atenção

HAYASHI não detecta laços infinitos. Um `while true { }` sem `break` executará indefinidamente. Use `Ctrl-C` para interromper.

8.3 `break` e `continue`

`break` interrompe o laço imediatamente. `continue` avança para a próxima iteração:

```
for i in 1..10 {
  if i == 3 { continue }    // pula o 3
  if i == 7 { break }       // para no 7
  display i
}
// exibe: 1, 2, 4, 5, 6
```

Listing 8.3: break e continue

8.4 Laços com DataFrames

`for` pode iterar sobre listas produzidas por funções de dados:

```
load "vendas.csv" as df

for uf in ["RS", "SP", "RJ"] {
  let sub = filter(df, estado == uf)
  display f"{uf}: {count(sub)} obs"
}
```

Listing 8.4: Iterando sobre grupos

Capítulo 9

Funções

9.1 Declaração

Funções são declaradas com `fn`:

```
fn quadrado(x) {
  return x^2
}

display quadrado(5)    // 25
```

Listing 9.1: Função básica

Retorno implícito

A última expressão de uma função é retornada automaticamente:

```
fn quadrado(x) {
  x^2    // return implícito
}
```

Múltiplos parâmetros

```
fn media(a, b) {
  (a + b) / 2.0
}

display media(10, 20)    // 15.0
```

9.2 Funções como valores

Funções são valores de primeira classe — podem ser atribuídas a variáveis, passadas como argumento e retornadas por outras funções:

```
fn aplicar(f, x) {
  f(x)
}

fn dobro(n) { n * 2 }

display aplicar(dobro, 7)    // 14
```

Listing 9.2: Função como argumento

9.3 Closures

Closures são funções anônimas que capturam o ambiente onde foram definidas:

```
let multiplicador = fn(fator) {
  fn(x) { x * fator }    // captura 'fator'
}

let triplo = multiplicador(3)
display triplo(10)    // 30
```

Listing 9.3: Closure básica

Closures como argumentos

```
fn aplicar_dois(f, a, b) {
  f(a) + f(b)
}

let soma_quadrados = aplicar_dois(fn(x) { x^2 }, 3, 4)
display soma_quadrados    // 9 + 16 = 25
```

9.4 Recursão

Funções podem chamar a si mesmas:

```
fn fat(n) {
  if n <= 1 { return 1 }
  n * fat(n - 1)
}

display fat(10)     // 3628800
```

Listing 9.4: Fatorial recursivo

Nota

HAYASHI não otimiza recursão em cauda (*tail-call optimization*). Para sequências longas, prefira laços.

9.5 Parâmetros opcionais e valores padrão

HAYASHI não tem parâmetros opcionais nativos, mas o padrão idiomático é usar `nil` como sentinela e verificar explicitamente:

```
fn formatar(valor, casas) {
  let c = if casas == nil { 2 } else { casas }
  // ...
}
```

9.6 Escopo de funções

Funções declaradas com `fn` no nível de módulo ficam disponíveis em todo o arquivo. Funções declaradas dentro de blocos são locais ao bloco:

```
{
  fn local(x) { x + 1 }
  display local(5)     // 6
}
display local(5)     // error: undefined function 'local'
```

9.7 `const` dentro de funções

`const` pode ser usado dentro de funções para fixar valores que não devem mudar durante a execução:

```
fn calcular_imc(peso, altura) {
  const FORMULA = "peso / altura^2"
  peso / altura^2
}
```

Capítulo 10

Tratamento de Erros

10.1 Modelo de erros

HAYASHI distingue dois tipos de erro:

Erros de parse Detectados antes da execução, ao analisar o código. Incluem linha e trecho do código.

Erros de runtime Ocorrem durante a execução — variável indefinida, tipo errado, divisão por zero, chave ausente.

Ambos os tipos exibem o trecho de código e a linha, facilitando a localização:

```
error: line 5: undefined variable 'rendimento' - did you mean
    'rendimento_bruto'?
   5   display rendimento * 12
         ^^^^^^^^^^^^^^^^^^^^^
```

10.2 `try` / `catch`

Erros de runtime podem ser capturados com `try`/`catch`:

```
try {
  let resultado = 10 / 0
  display resultado
} catch e {
  display f"Erro capturado: {e}"
}
```

Listing 10.1: try/catch básico

A variável `e` dentro do `catch` contém a mensagem de erro como string.

Capturando erros de arquivo

```
try {
  load "dados_inexistentes.csv" as df
} catch e {
  display f"Arquivo não encontrado: {e}"
}
```

`try` como expressão

`try`/`catch` pode retornar um valor:

```
let valor = try {
  int("não é um número")
} catch _ {
  -1
}
display valor    // -1
```

10.3 Erros comuns e suas mensagens

Situação	Mensagem típica
Variável não declarada	`undefined variable 'x' - did you mean 'y'?`
Função não existe	`undefined function 'foo' - did you mean 'for'?`
Tipo errado	`type mismatch - expected Int, got String`
Índice fora de limite	`index 5 out of bounds (len=3)`
Chave ausente em dict	`key 'nome' not found in dict`
Divisão por zero (Int)	`division by zero`
Arquivo não encontrado	`file not found: 'caminho.csv'`

10.4 Sugestões automáticas

Quando o nome de uma variável ou função é parecido com algo existente, HAYASHI sugere a correção:

```
error: undefined function 'regres' - did you mean 'regress'?
```

As sugestões usam distância de edição (*Levenshtein*) sobre uma lista de todos os nomes definidos no escopo e nas funções embutidas.

10.5 Re-lançamento de erros

Para propagar um erro capturado:

```
try {
  operacao_arriscada()
} catch e {
  display f"log: {e}"
  error(e)    // re-lança o erro
}
```

A função `error(msg)` lança um erro de runtime com a mensagem fornecida.

Capítulo 11

Strings e F-Strings

11.1 Literais de string

Strings são delimitadas por aspas duplas:

```
let s = "Olá, mundo"
let caminho = "/home/usuario/dados.csv"
let vazia = ""
```

Sequências de escape

Sequência	Significado
\n	Nova linha
\t	Tabulação
\r	Retorno de carro
\"	Aspas duplas literais
\\	Barra invertida literal

11.2 Operações com strings

Concatenação

```
let nome = "Hayashi"
let versao = "0.2"
let titulo = nome + " " + versao    // "Hayashi 0.2"
```

Comprimento

```
let s = "econometria"
display len(s)    // 11
```

Conversões

```
let n = str(42)         // "42"
let f = str(3.14)       // "3.14"
let b = str(true)       // "true"
```

Comparação

Strings são comparadas lexicograficamente com `==`, `!=`, `<`, `>`, `<=`, `>=`.

11.3 F-Strings

F-strings permitem interpolar expressões arbitrárias dentro de uma string. O prefixo `f` indica interpolação; expressões entre `{ }` são avaliadas e convertidas para string:

```
let municipio = "Pelotas"
let pop = 328_000

display f"{municipio} tem {pop} habitantes"
// Pelotas tem 328000 habitantes

display f"Média: {(10 + 20 + 30) / 3.0:.2f}"
// Média: 20.00
```

Listing 11.1: F-strings

Expressões em f-strings

Qualquer expressão pode ser interpolada — variáveis, operações aritméticas, chamadas de função:

```
let n = 5
display f"{n}^2 = {n^2}"
// 5^2 = 25

```

```
display f"raiz de {n} = {sqrt(n):.4f}"
// raiz de 5 = 2.2361
```

F-strings em `display`

`display` aceita opções `end=` e `sep=` para controlar o terminador e o separador entre múltiplos valores:

```
display "a", "b", "c" sep=", "    // a, b, c
display "linha" end=""             // linha (sem quebra)
```

11.4 Multiline strings

HAYASHI não tem literal de string multiline. Use concatenação explícita:

```
let sql = "SELECT * FROM dados " +
          "WHERE ano = 2026 " +
          "ORDER BY uf"
```

Capítulo 12

Listas e Dicionários

12.1 Listas

12.1.1 Criação

```
let nums  = [1, 2, 3, 4, 5]
let nomes = ["RS", "SP", "RJ"]
let misto = [1, "dois", true]
let vazia = []
```

12.1.2 Acesso por índice

Índices começam em 0. Índices negativos contam a partir do fim:

```
let xs = [10, 20, 30, 40]
display xs[0]     // 10
display xs[3]     // 40
display xs[-1]    // 40  (último)
display xs[-2]    // 30
```

Acesso fora dos limites produz erro de runtime:

```
xs[10]
// error: index 10 out of bounds (len=4)
```

12.1.3 Comprimento

```
len([1, 2, 3])    // 3
```

12.1.4 Mutação

`push` adiciona ao final da lista (muta no lugar, retorna `nil`):

```
let lista = [1, 2, 3]
push(lista, 4)
display lista    // [1, 2, 3, 4]
```

Atenção

`push` é uma função mutante — altera `lista` diretamente e retorna `nil`. Não faça `let nova = push(lista, 4)`.

`pop` remove e retorna o último elemento:

```
let ultimo = pop(lista)
```

12.1.5 Slicing

```
let xs = [0, 1, 2, 3, 4, 5]
display slice(xs, 1, 4)    // [1, 2, 3]  (índices 1..3)
```

12.1.6 Iteração

```
for x in [10, 20, 30] {
  display x * 2
}
```

12.1.7 Funções funcionais

```
let xs = [1, 2, 3, 4, 5]

let dobrados  = map(xs, fn(x) { x * 2 })
let pares     = filter(xs, fn(x) { x % 2 == 0 })
let soma      = reduce(xs, 0, fn(acc, x) { acc + x })
```

12.2 Ranges

Ranges são expressões que produzem listas de inteiros. A sintaxe segue Rust:

```
let a = 1..5       // [1, 2, 3, 4]    (exclusivo)
let b = 1..=5      // [1, 2, 3, 4, 5] (inclusivo)
```

Listing 12.1: Ranges como listas

Por serem listas normais, ranges funcionam com qualquer função que aceite lista:

```
map(1..=5, fn(x) { x^2 })             // [1, 4, 9, 16, 25]
filter(1..10, fn(x) { x % 2 == 0 }) // [2, 4, 6, 8]
len(1..100)                           // 99
```

12.2.1 chain

`chain` concatena múltiplas listas (ou ranges) em uma só:

```
chain(1..3, 9..=11)           // [1, 2, 9, 10, 11]
chain(1..=3, [10, 20], 5..7)  // [1, 2, 3, 10, 20, 5, 6]
```

Listing 12.2: chain com ranges e listas

Uso típico em laços sobre sequências não-contíguas:

```
for lag in chain(1..=3, 6..=8) {
  display f"lag {lag}"
}
```

12.2.2 Composição com closures

Ranges, `chain` e closures compõem naturalmente em uma única expressão. A sintaxe `|x| expr` define uma closure sem a cerimônia de `fn`:

```
let r = map(chain(1..=10, 5..9, 56..70), |n| n * 3)
```

Listing 12.3: Expressão composta — única linha

O pipe também funciona no cabeçalho do `for`, compondo transformações antes de entrar no corpo do laço:

```
for i in chain(1..=10, 20..=25) |> filter(|n| (n % 2) == 0) {
    display i
}
```

Listing 12.4: Pipeline no cabeçalho do for

12.2.3 Comparação com outras linguagens

A tabela abaixo mostra como as principais linguagens expressariam o mesmo pipeline:

Linguagem	Forma
Python	`filter(lambda n: n%2==0, chain(range(1,11), range(5,9), range(56,70)))`
Rust	`(1..=10).chain(5..9).chain(56..70).filter(|n| n%2==0)`
Haskell	`filter even $ [1..10] ++ [5..8] ++ [56..69]`
Julia	`Iterators.filter(n->n%2==0, Iterators.flatten([1:10,5:8,56:69]))`
R	`Filter(function(n) n%%2==0, c(1:10, 5:8, 56:69))`
Hayashi	`chain(1..=10,5..9,56..70) |> filter(|n| (n%2)==0)`

HAYASHI é a única linguagem em que o pipeline é estritamente da esquerda para a direita, sem imports, sem namespaces e com `chain` variádico que trata todos os argumentos simetricamente.

12.2.4 Nota sobre desempenho

`chain` materializa os ranges em uma lista antes de iterar. `for i in 1..N`, por outro lado, itera diretamente sem alocar nada.

`chain` é para sequências não-contíguas pequenas — lags, anos, grupos de variáveis. Esse é o seu único caso de uso legítimo. Qualquer sequência contígua é simplesmente `1..N` ou `1..=N`.

12.3 Dicionários

12.3.1 Criação

```
let config = {
  "host": "localhost",
  "porta": 5432,
  "ssl": true
}

let vazio = {}
```

Chaves podem ser strings ou inteiros. Valores podem ser de qualquer tipo.

12.3.2 Acesso

```
config["host"]      // "localhost"
config["porta"]     // 5432
```

Chave ausente produz erro:

```
config["usuario"]
// error: key 'usuario' not found in dict
```

Use `try`/`catch` para acesso seguro:

```
let user = try { config["usuario"] } catch _ { "anonimo" }
```

12.3.3 Inserção e atualização

```
let d = {"a": 1}
// d é imutável; d2 = {"a":1, "b":2}
let d2 = dict_set(d, "b", 2)
```

`dict_set` é funcional — retorna um novo dicionário sem mutar o original.

12.3.4 Verificando chave

```
"host" in config    // true
"senha" in config   // false
```

12.3.5 Dicionários como retorno de funções

Funções de análise frequentemente retornam dicionários com múltiplos resultados:

```
load "dados.csv" as df
let stats = summarize(df, renda)

display stats["mean"]    // média
display stats["sd"]      // desvio padrão
display stats["N"]       // número de observações
```

Listing 12.5: summarize retorna dicionário

12.3.6 Iteração sobre dicionários

```
let d = {"a": 1, "b": 2, "c": 3}

for key in keys(d) {
  display f"{key} = {d[key]}"
}
```

Capítulo 13

DataFrames

13.1 O que é um DataFrame

Um DataFrame é uma tabela bidimensional com colunas nomeadas. Em Hayashi, cada coluna é um vetor de ponto flutuante de 64 bits. Valores ausentes são representados por `NaN` (*Not a Number*).

O DataFrame é o tipo central da linguagem — praticamente toda análise econométrica começa com um DataFrame carregado de alguma fonte.

13.2 Semântica de cópia (COW)

DataFrames usam *copy-on-write* (COW): atribuir um DataFrame a outra variável cria um alias que compartilha os mesmos dados internos. A cópia real ocorre apenas no momento em que um dos dois é modificado.

```
load "painel.csv" as df

let baseline = df       // alias barato, sem cópia

let df2 = df |> generate(z = x^2)     // df2 tem 'z', df não
let df3 = df |> filter(ano > 2000)    // df3 filtrado, df
    intacto

// baseline == df original em qualquer ponto
```

Listing 13.1: COW: baseline preservado

Esta semântica permite criar variantes de análise sem custo de memória até que uma modificação real seja necessária.

13.3 Criando DataFrames

Bloco `input`

Para pequenos datasets inline — ideal para exemplos e testes:

```
input x y z
  1   2  10
  3   4  20
  5   6  30
end
```

Listing 13.2: input block

O bloco `input` cria um DataFrame chamado `df` por padrão. O número de colunas é definido pelo cabeçalho; cada linha de dados deve ter o mesmo número de valores.

Nota

`input` aceita apenas valores numéricos. Use `.` para representar valores ausentes. Para dados com colunas de texto, use `load`.

`load` — arquivos externos

```
load "dados.csv" as vendas
load "painel.dta" as painel          // formato Stata
load "resultado.parquet" as df
load "planilha.xlsx" as df           // primeira aba
load "planilha.xlsx" sheet="Dados" as df
```

Listing 13.3: Carregando arquivos

Formatos suportados: CSV, Parquet, Stata (`.dta`), Excel (`.xlsx`), SQLite.

13.4 Salvando DataFrames

```
save df "saida.csv"
save df "saida.parquet"
save df "saida.dta"
save df "saida.xlsx"
```

13.5 Inspecionando DataFrames

display

Exibe as primeiras linhas do DataFrame:

```
display df
display df, 20    // exibe até 20 linhas
```

describe

Exibe a estrutura: número de observações, variáveis e tipos:

```
describe df
```

Saída típica:

```
DataFrame: 1250 obs, 8 vars
  ano      Float  (0 missing)
  uf       Float  (0 missing)
  renda    Float  (12 missing)
  gini     Float  (3 missing)
```

count e nrow

Retorna o número de linhas:

```
let n = count(df)           // total de observações
let n2 = nrow(df)           // alias de count(df)
// contagem condicional
let n3 = count(df, renda > 1000)
```

13.6 Variáveis especiais

Dentro de comandos que operam linha a linha (`generate`, `mutate`, `filter`), duas variáveis implícitas estão disponíveis:

Número da linha atual (começa em 1).

Total de linhas do DataFrame.

```
generate df id = _n          // 1, 2, 3, ..., N
generate df peso = 1.0/_N    // peso uniforme
```

13.7 Operadores de séries temporais

Dentro de expressões de `generate` (forma imperativa), operadores especiais para séries temporais estão disponíveis:

Operador	Significado	Exemplo
`L.x`	Lag (valor anterior)	`L.pib` → pib_{t-1}
`F.x`	Lead (valor seguinte)	`F.taxa` → taxa_{t+1}
`D.x`	Diferença primeira	`D.pib` → pib_t - pib_{t-1}
`L2.x`	Lag de ordem 2	`L2.pib` → pib_{t-2}

```
generate df dpib = D.pib       // variação do PIB
generate df pib_lag = L.pib    // PIB defasado um período
```

Listing 13.4: Operadores de séries temporais

Capítulo 14

Manipulação de Dados

14.1 `generate` e `mutate`

Criam ou substituem colunas. As duas formas são equivalentes em resultado; diferem na semântica de mutação:

Forma	Semântica	Exemplo
`generate df col = expr`	Imperativa (muta `df`)	estilo Stata
`let df2 = generate(df, col = expr)`	Funcional (df inalterado)	estilo funcional
`df |> generate(col = expr)`	Pipe assign-back (muta `df`)	encadeamento

```
load "dados.csv" as df

generate df lrenda = log(renda)
generate df renda2 = renda^2
generate df dummy = if(renda > 1000, 1, 0)
```

Listing 14.1: generate: forma imperativa

```
let df2 = mutate(df, lrenda = log(renda), renda2 = renda^2)
// df não possui lrenda nem renda2
// df2 as possui
```

Listing 14.2: mutate: forma funcional

14.2 `filter`

Seleciona linhas que satisfazem uma condição:

```
let recentes = filter(df, ano >= 2010)
let ricos = filter(df, renda > 5000 and uf == "SP")
```

A forma imperativa usa `keep`:

```
keep df if ano >= 2010
```

14.3 select e drop

```
// mantém só essas colunas
let slim = select(df, ano, uf, renda)
let sem_id = drop(df, id, codigo)        // remove essas
    colunas
```

14.4 rename

```
rename df antiga nova
// renomeia renda_familiar → renda
rename df renda_familiar renda
```

14.5 sort

```
sort df ano                // ascendente
sort df renda desc         // descendente
sort df uf ano desc        // múltiplas colunas
```

14.6 dropna

Remove linhas com valores ausentes:

```
dropna df                  // remove linhas com qualquer NaN
// remove linhas com NaN em renda ou gini
dropna df renda gini
```

14.7 ffill

Forward-fill substitui `NaN` pelo último valor não ausente. Útil quando uma série mensal ou trimestral precisa ser expandida para frequência diária:

```
let daily = merge(asset, rf, key=date, type=left)
let daily = ffill(daily)
```

Listing 14.3: Preenchendo uma taxa mensal em dados diários

14.8 append

Empilha dois DataFrames verticalmente (equivalente a `rbind`):

```
let df_total = append(df_2024, df_2025)
```

As colunas devem ser compatíveis. Colunas ausentes em um dos DataFrames são preenchidas com `NaN`.

14.9 collapse e group_by

`group_by` agrega por grupos:

```
let media_uf = group_by(df, uf,
  renda_media = mean(renda),
  n           = count()
)
```

Listing 14.4: Agregação por grupo

`collapse` é a forma imperativa estilo Stata:

```
collapse df (mean) renda gini, by(uf ano)
```

14.10 pivot_longer e pivot_wider

```
let longo = pivot_longer(df,
  cols  = ["t1", "t2", "t3"],
  names = "trimestre",
  values = "valor"
)
```

Listing 14.5: Formato largo para longo

```
let largo = pivot_wider(df,
  id     = "municipio",
  names  = "ano",
  values = "pib"
)
```

Listing 14.6: Formato longo para largo

14.11 replace

Substitui valores condicionalmente:

```
// zera valores negativos
replace df renda = 0 if renda < 0
replace df gini = . if gini > 1          // marca como missing
```

14.12 tabulate

Tabela de frequências:

```
tabulate df uf
tabulate df uf ano    // tabulação cruzada
```

14.13 Auxiliares de arquivo

Pequenos auxiliares para scripts que persistem dados ou verificam cache:

```
ensure_dir("cache")                    // cria diretório se n
   ão existir
if file_exists("cache/dados.csv") {
    load "cache/dados.csv" as df
} else {
    let df = download(...)
    export(df, "csv", "cache/dados.csv")
}

write("texto bruto", "nota.txt")     // escreve string em
   arquivo
```

Listing 14.7: Auxiliares de cache

14.14 rolling

`rolling` estima OLS em uma janela móvel. É útil para visualizar coeficientes que variam no tempo ou testar a estabilidade de parâmetros. Uma coluna `date` opcional armazena o rótulo do final de cada janela.

```
let roll = rolling(y ~ x1 + x2, df, window=252, date=date)
let betas = tidy(roll)
print(betas)
```

Listing 14.8: OLS rolante com rótulos de data

14.15 tidy e glance

Essas funções convertem objetos de modelo em DataFrames. `tidy` retorna um DataFrame organizado com coeficientes, erros-padrão e intervalos de confiança. Para modelos `rolling`, retorna uma linha por janela com colunas `date`, `r2` e uma coluna de coeficiente por variável. `glance` retorna um resumo de uma linha com estatísticas de ajuste.

```
let m = ols(y ~ x1 + x2, df)
let coefs = tidy(m)
let summary = glance(m)
```

Listing 14.9: Tidy e glance

Capítulo 15

O Operador Pipe

15.1 Sintaxe básica

O operador `|>` passa o valor à sua esquerda como primeiro argumento da função à sua direita:

```
expr |> f(args...)
// equivale a: f(expr, args...)
```

Sem pipe:

```
display sort(filter(df, renda > 0), renda desc)
```

Com pipe:

```
df |> filter(renda > 0) |> sort(renda desc) |> display
```

O pipe permite ler transformações na ordem em que ocorrem, da esquerda para a direita (ou de cima para baixo), sem aninhamento de parênteses.

15.2 Semântica com DataFrames

O comportamento do pipe com DataFrames segue uma regra única:

Forma	Efeito
`df |> f(...)`	Assign-back: df recebe o resultado
`let df2 = df |> f(...)`	Funcional: df inalterado, df2 é novo

Assign-back

Quando o pipe começa com uma variável (`df |> ...`) e não há `let`, o resultado é atribuído de volta à variável de origem:

```
load "dados.csv" as df

df |> filter(ano >= 2010)
   |> dropna
   |> sort(renda desc)

// df agora é o resultado de todas as transformações
```

Listing 15.1: Assign-back: df é modificado

Forma funcional com `let`

```
load "dados.csv" as df

let df_limpo = df
  |> filter(ano >= 2010)
  |> dropna

// df está intacto; df_limpo é a versão filtrada
```

Listing 15.2: let preserva o original

15.3 COW e pipes

A semântica COW garante que o assign-back é eficiente: enquanto `df` e `df_limpo` compartilham os mesmos dados (antes de qualquer mutação), nenhuma cópia de memória ocorre. A cópia real acontece apenas quando os dados divergem.

```
load "painel.csv" as df
let baseline = df      // alias barato

df |> filter(tratamento == 1)   // modifica df, não baseline

display count(baseline)    // todas as obs originais
display count(df)          // só o grupo de tratamento
```

Listing 15.3: Preservando baseline com let

15.4 Encadeamento longo

O pipe é especialmente útil em pipelines de limpeza e transformação:

```
load "pnad.csv" as pnad

let amostra = pnad
  |> filter(idade >= 18 and idade <= 65)
  |> filter(horas_trab > 0)
  |> dropna
  |> generate(lrenda = log(renda + 1))
  |> generate(exp = idade - anos_estudo - 6)
  |> generate(exp2 = exp^2)
  |> sort(uf ano)

display count(amostra)
```

Listing 15.4: Pipeline completo de preparação

15.5 Uso de placeholders

Se você precisar passar o valor da esquerda para uma posição específica de argumento diferente da primeira, pode utilizar o caractere underscore (_) como placeholder:

```
expr |> f(a, _)
// equivale a: f(a, expr)
```

Isso é particularmente útil ao passar DataFrames para estimadores (como `ols` ou `iv`), onde o DataFrame não é o primeiro argumento:

```
let modelo = df |> ols(y ~ x1 + x2, _)
```

15.6 Pipe com closures

O operador `|>` também funciona com closures:

```
let resultado = 5 |> fn(x) { x^2 + 1 }
display resultado    // 26
```

Capítulo 16

Módulos, Pacotes e Plugins

16.1 O Sistema de Módulos do Hayashi

Em HAYASHI, o compartilhamento e a organização de código são feitos através de dois comandos principais: `source` e `import`.

16.1.1 O comando `source`

O comando `source` executa um arquivo `.hay` diretamente no contexto atual. Todas as definições (funções e variáveis) do arquivo ficam visíveis no escopo global de quem o executa, sem namespaces:

```
source "lib/utils.hay"
let resultado = minha_funcao_util(10)
```

Listing 16.1: Uso básico de source

16.1.2 O comando `import`

O comando `import` carrega um módulo isolando suas funções sob um **namespace** usando o delimitador `::`. Isso evita conflitos de nomes e organiza melhor o código:

```
// Importa e usa o nome do arquivo como namespace padrão
import("lib/matematica.hay")
let r1 = matematica::calcular_media([10, 20, 30])

// Importa definindo um alias customizado
import("lib/matematica.hay", as=mat)
let r2 = mat::calcular_media([10, 20, 30])

```

```
// Importa apenas funções específicas diretamente para o escopo local
import("lib/matematica.hay", only=["calcular_media"])
let r3 = calcular_media([10, 20, 30])
```

Listing 16.2: Importando com namespace

—

16.2 Gerenciador de Pacotes e Publicação no GitHub

O Hayashi utiliza o **GitHub diretamente como registro de pacotes**. Isso elimina a necessidade de um repositório centralizado proprietário e promove a distribuição descentralizada de código aberto.

16.2.1 Comandos do CLI

Para gerenciar pacotes de terceiros pelo terminal do sistema operacional, use:

```
hay install usuario/repositorio   -- Instala um pacote do GitHub (-y para pular prompt de sobrescrita)
hay remove  usuario/repositorio   -- Desinstala um pacote (deleta arquivos, pastas e metadados)
hay list                          -- Lista pacotes e plugins instalados
hay check-plugin [user/repo]      -- Verifica a integridade/versao contra o repositorio remoto
hay update [user/repo] [-y]       -- Atualiza o(s) pacote(s) para a versao mais recente (-y pula prompt)
```

16.2.2 Estrutura Padrão de um Pacote

Os pacotes do Hayashi são baseados em convenção e exigem configuração zero:

- **Pacotes de Script**: Basta fazer push de seus scripts `.hay` para a raiz de um repositório no GitHub (ex: `github.com/joao/financas/juros.hay`).
- **Plugins Nativos**: Configure um workflow do GitHub Actions para compilar e subir os binários gerados (`.so`, `.dll`, `.dylib` ou `.wasm`) como assets em um GitHub Release.

Depois de subir seu pacote no GitHub, qualquer usuário do mundo poderá instalar e usar no Hayashi:

```
// No terminal do sistema operacional:
// hay install joao/financas

// No seu script Hayashi:
import("joao/financas/juros")
let taxa = financas::calcular_taxa(1000, 12)
```

Listing 16.3: Instalando e importando um pacote

—

16.3 Desenvolvimento de Plugins Avançados

Para necessidades que exigem alta performance ou integração com outras linguagens, o Hayashi suporta três tipos de plugins sob a mesma sintaxe de `import`:

16.3.1 1. Plugins de Script (.hay)

Desenvolvidos em Hayashi puro. Úteis para funções econométricas compostas e utilitários rápidos.

16.3.2 2. Plugins Nativos (Exclusivos em Rust)

Os plugins nativos são desenvolvidos em Rust para garantir máxima performance e segurança de memória. Eles se comunicam com o Hayashi através de uma FFI C-string JSON estável e desacoplada.

Para facilitar o desenvolvimento, você deve usar o SDK oficial `hayashi-plugin-sdk`. O SDK fornece macros de procedimento para gerar automaticamente todo o boilerplate FFI unsafe e lidar com a serialização/deserialização JSON:

```
use hayashi_plugin_sdk::{hayashi_fn, hayashi_plugin};

// A macro #[hayashi_fn] lida automaticamente com a (de)
   serialização JSON.
// Você pode usar tipos padrão do Rust (Vec, HashMap, String,
    primitivos, etc.).
#[hayashi_fn]
pub fn distance_to_point(lats: Vec<f64>, lons: Vec<f64>,
   ref_lat: f64, ref_lon: f64) -> Vec<f64> {
```

```
    lats.iter()
        .zip(lons.iter())
        .map(|(&lat, &lon)| {
            // calculo da distancia de Haversine
            let r = 6371.0;
            let d_lat = (ref_lat - lat).to_radians();
            let d_lon = (ref_lon - lon).to_radians();
            let a = (d_lat / 2.0).sin().powi(2)
                + lat.to_radians().cos() * ref_lat.to_radians
                   ().cos() * (d_lon / 2.0).sin().powi(2);
            let c = 2.0 * a.sqrt().atan2((1.0 - a).sqrt());
            r * c
        })
        .collect()
}

// Gera as funções necessárias para limpeza de memória via
   FFI
hayashi_plugin!();
```

Listing 16.4: Desenvolvendo um plugin nativo usando o SDK

Compilado como um `cdylib`, você pode publicar seu plugin criando uma tag (ex: `v0.1.0`) no GitHub. O workflow de GitHub Actions irá compilar o plugin para Linux (`.so`), macOS (`.dylib`) e Windows (`.dll`), fazendo o upload deles como assets do release.

Qualquer usuário pode instalá-lo utilizando a CLI:

```
hay install john/spatial
```

O gerenciador de pacotes baixa automaticamente o binário correto correspondente ao sistema operacional e arquitetura do usuário.

Você pode então carregá-lo e nomear seu namespace no Hayashi:

```
import("john/spatial", as=geo)
let dist = geo::distance_to_point(df["lat"], df["lon"],
   -30.03, -51.22)
generate df dist_poa = dist
```

Considerações de Segurança e Memória: JSON ABI vs. Zero-Copy FFI

Embora a ABI JSON baseada em C-strings padrão ofereça um desacoplamento e segurança extremos (já que evoluções no esquema JSON não quebram layouts de

memória física), ela pode se tornar um gargalo ao compartilhar grandes volumes de dados (Big Data) devido ao overhead de serialização.

Para mitigar esse problema, o Hayashi suporta compartilhamento de memória zero-copy transparente utilizando o formato FFI do Apache Arrow para trocas complexas de arrays e dataframes. Isso é feito de forma automática passando ponteiros de structs FFI (`FFI_ArrowArray` e `FFI_ArrowSchema`). Contudo, desenvolvedores devem buscar o equilíbrio ideal:

- **JSON C String**: Oferece máximo desacoplamento, segurança e compatibilidade de versão binária entre diferentes compilações de compiladores. Indicado para rotinas de controle, pequenos volumes de dados e extensibilidade geral.
- **Apache Arrow Zero-Copy**: Oferece máxima velocidade e zero overhead de alocação de memória. Indicado para cálculos de uso intensivo de dados, inferência de redes neurais ou grandes estimações econométricas. Exige uma coordenação estrita do ciclo de vida de memória (prevenção manual de bugs de liberação dupla na fronteira FFI).

Esta dualidade representa uma decisão de projeto intencional da linguagem Hayashi. Limitar o suporte a zero-copy estritamente a DataFrames e colunas homogêneas protege o ecossistema contra o perigo de quebras de compatibilidade binária (ABI) em estruturas de controle e dados complexas, as quais são leves para serializar via JSON C String. Dessa forma, mantém-se a estabilidade e a facilidade de manutenção de longo prazo da linguagem sem comprometer a performance no tráfego de dados volumosos.

16.3.3 3. Plugins WebAssembly (WASM - Para demais linguagens)

Para desenvolvedores utilizando outras linguagens (como C, Go, Zig, etc.), a compilação deve ser feita obrigatoriamente para WebAssembly (`.wasm`). Esses arquivos são executados de forma totalmente isolada (sandboxed) através de um motor WASM integrado ao Hayashi, garantindo segurança total (qualquer falha no plugin não derruba o Hayashi) e portabilidade multiplataforma instantânea.

O plugin WASM deve exportar as funções `alloc(size: i32) -> i32` e `dealloc(ptr: i32, size: i32)` para gerenciar a passagem de strings JSON na memória linear do WASM, e a função do plugin deve retornar um `i64` contendo o ponteiro (nos 32 bits superiores) e o tamanho (nos 32 bits inferiores) da string JSON de retorno.

Parte II

Econometria Aplicada

Capítulo 17

Estatística Descritiva e Momentos

Este capítulo apresenta as ferramentas de estatística descritiva do Hayashi como expressões da teoria das probabilidades. Cada função mapeia diretamente para um conceito teórico, tornando o código legível tanto para quem vem da matemática quanto para quem vem do Stata ou R.

17.1 Valor esperado

O valor esperado $E(X)$ de uma variável aleatória discreta é a média ponderada de seus valores. Para dados observados, o estimador natural é a média amostral:

$$\hat{E}(X) = \bar{X} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Em Hayashi, `mean` aceita uma lista ou uma coluna de DataFrame:

```
// sobre lista
let xs = [2.0, 4.0, 4.0, 4.0, 5.0, 5.0, 7.0, 9.0]
display mean(xs)    // 5.0

// sobre coluna de DataFrame
load "dados.csv" as df
display mean(df, renda)
```

Listing 17.1: E(X) sobre lista e sobre coluna

17.2 Variância e desvio padrão

A variância amostral e o desvio padrão amostral são:

$$\widehat{\mathrm{Var}}(X) = s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{X})^2 \qquad s = \sqrt{s^2}$$

O divisor $n-1$ (correção de Bessel) produz um estimador não viesado de σ^2. Todas as funções de HAYASHI usam essa convenção.

```
display variance(df, renda)    // s²
display sd(df, renda)          // s
```

Listing 17.2: Variância e desvio padrão

Nota

`sd` e `std` são aliases — ambos calculam o desvio padrão amostral com divisor $n-1$.

Para obter ambos de uma vez, `summarize` retorna um dicionário com todos os momentos:

```
let stats = summarize(df, renda)

display stats["mean"]      // média
display stats["sd"]        // desvio padrão
display stats["variance"]  // variância
display stats["min"]       // mínimo
display stats["max"]       // máximo
display stats["N"]         // número de observações
```

17.3 Momentos condicionais

O valor esperado condicional $E(X \mid C)$ restringe o cálculo a um subconjunto da amostra sem criar um DataFrame auxiliar. A opção `if =` está disponível em todas as funções de agregação escalar:

```
// E(renda | setor == "publico")
display mean(df, renda, if = setor == "publico")

// Var(renda | setor == "publico")
display variance(df, renda, if = setor == "publico")

// DP(renda | setor == "publico")
display sd(df, renda, if = setor == "publico")
```

Listing 17.3: Momentos condicionais com if =

A condição é avaliada linha a linha no DataFrame — qualquer expressão booleana sobre colunas é válida:

```
display mean(df, salario, if = educ >= 12 and sexo == 1)
display sd(df, renda,   if = uf == "RS" or uf == "SC")
```

Listing 17.4: Condições compostas

Dica

`if =` não altera o DataFrame. Para criar uma amostra permanente use `filter`; para um cálculo pontual use `if =`.

17.4 Covariância

A covariância amostral entre X e Y mede a variação conjunta:

$$\widehat{\mathrm{Cov}}(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{X})(y_i - \bar{Y})$$

```
display cov(df, educ, salario)

// condicional: apenas trabalhadores formais
display cov(df, educ, salario, if = formal == 1)
```

Listing 17.5: Covariância entre duas colunas

Nota

$\mathrm{Cov}(X,X) = \mathrm{Var}(X)$. Para verificar: `cov(df, x, x)` devolve o mesmo que `variance(df, x)`.

17.5 Correlação

17.5.1 Escalar entre duas variáveis

O coeficiente de correlação de Pearson normaliza a covariância pelo produto dos desvios padrão:

$$\rho(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\mathrm{DP}(X)\,\mathrm{DP}(Y)} \in [-1,\,1]$$

`corr_pair` devolve esse escalar diretamente:

```
display corr_pair(df, educ, salario)

// condicional: setor privado
display corr_pair(df, educ, salario, if = setor == "privado")
```

Listing 17.6: Correlação escalar

17.5.2 Matriz de correlação

Para explorar todas as relações entre múltiplas variáveis use `correlate`:

```
// todas as variáveis numéricas
correlate(df)

// subconjunto explícito
correlate(df, educ, exp, salario)

// com p-values (* p<0.1  ** p<0.05  *** p<0.01)
pwcorr(df, educ, exp, salario)
```

Listing 17.7: Matriz de correlação completa

Nota

`correlate` imprime a matriz mas não retorna um valor — use `corr_pair` quando precisar do escalar em um cálculo.

17.6 Mediana e quantis

A mediana é o valor central da distribuição ordenada, menos sensível a valores extremos que a média.

```
display median(df, renda)

// condicional
display median(df, renda, if = regiao == "Sul")
```

Listing 17.8: Mediana

Quantis generalizam a mediana para qualquer ponto $p \in [0, 1]$ da distribuição:

```
display quantile(df, renda, 0.25)   // 1º quartil
display quantile(df, renda, 0.50)   // mediana
display quantile(df, renda, 0.75)   // 3º quartil
display quantile(df, renda, 0.90)   // percentil 90
```

Listing 17.9: Quantis

`quantile` aceita `if =`:

```
// Percentil 90 da renda entre trabalhadores com ensino
   superior
display quantile(df, renda, 0.90, if = educ >= 16)
```

Ambas as funções aceitam listas diretamente:

```
let xs = [3.0, 1.0, 4.0, 1.0, 5.0, 9.0, 2.0, 6.0]
display median(xs)           // 3.5
display quantile(xs, 0.75)   // 5.25
```

17.7 Sumário detalhado

`summarize` com `detail=true` expande o dicionário de retorno para incluir assimetria, curtose e toda a grade de percentis:

```
let s = summarize(df, renda, detail=true)

display s["mean"]     // média
display s["sd"]       // desvio padrão
display s["p25"]      // 1º quartil
display s["p50"]      // mediana
display s["p75"]      // 3º quartil
display s["p90"]      // percentil 90
display s["p99"]      // percentil 99
display s["skew"]     // assimetria
display s["kurt"]     // curtose
```

Listing 17.10: summarize com detail

17.8 Referência rápida

Notação	Hayashi	Formas
$E(X)$	`mean(df, x)`	lista / df / `if =`
$\mathrm{Var}(X)$	`variance(df, x)`	lista / df / `if =`
$\mathrm{DP}(X)$	`sd(df, x)`	lista / df / `if =`
$\mathrm{Cov}(X,Y)$	`cov(df, x, y)`	df / `if =`
$\rho(X,Y)$	`corr_pair(df, x, y)`	df / `if =`
$\mathrm{Med}(X)$	`median(df, x)`	lista / df / `if =`
$Q_p(X)$	`quantile(df, x, p)`	lista / df / `if =`
`correlate(df, x, y, ...)`		matriz de correlação (display)
`summarize(df, x)`		dicionário completo de momentos

17.9 Exemplo completo

O trecho abaixo ilustra o uso conjunto dessas funções em uma análise exploratória típica antes de estimar um modelo:

```
load "pnad.csv" as df

// visão geral
summarize(df, renda)
summarize(df, educ)
correlate(df, renda, educ, exp)

// momentos por grupo
display mean(df, renda, if = sexo == 1)      // homens
display mean(df, renda, if = sexo == 0)      // mulheres

display sd(df, renda, if = sexo == 1)
display sd(df, renda, if = sexo == 0)

// distância entre mediana e média: sinal de assimetria
display mean(df, renda)
display median(df, renda)

// associação linear
display cov(df, educ, renda)
display corr_pair(df, educ, renda)
```

```
22
23 // concentração no topo
24 display quantile(df, renda, 0.90)
25 display quantile(df, renda, 0.99)
```

Listing 17.11: Análise exploratória — renda e educação

Capítulo 18

Mínimos Quadrados Ordinários

18.1 O modelo

Considere o modelo de regressão linear:

$$y = X\beta + \varepsilon$$

onde y é um vetor $n \times 1$ de observações da variável dependente, X é uma matriz $n \times k$ de regressores (a primeira coluna é identicamente 1, correspondendo ao intercepto), β é um vetor $k \times 1$ de parâmetros desconhecidos, e ε é um vetor $n \times 1$ de erros.

Hipóteses de Gauss-Markov

1. **Linearidade:** $y = X\beta + \varepsilon$.
2. **Posto completo:** $\operatorname{rank}(X) = k$ (sem multicolinearidade perfeita).
3. **Exogeneidade estrita:** $E[\varepsilon \mid X] = 0$.
4. **Homocedasticidade:** $\operatorname{Var}[\varepsilon \mid X] = \sigma^2 I_n$.
5. **Não-autocorrelação:** $\operatorname{Cov}[\varepsilon_i, \varepsilon_j \mid X] = 0$ para $i \neq j$.

Sob H1–H3 o OLS é não-viesado. Sob H1–H5 é BLUE (Teorema de Gauss-Markov).

18.2 O estimador

O estimador OLS minimiza a soma dos quadrados dos resíduos:

$$\hat{\beta} = \arg\min_{\beta} (y - X\beta)'(y - X\beta)$$

A condição de primeira ordem $X'(y - X\beta) = 0$ — as *equações normais* — tem solução analítica:

$$\boxed{\hat{\beta} = (X'X)^{-1}X'y}$$

Os resíduos são $\hat{e} = y - X\hat{\beta} = M_X y$, onde $M_X = I - X(X'X)^{-1}X'$ é a matriz de resíduos (simétrica e idempotente).

18.3 Propriedades

Sob H1–H3:

$$E[\hat{\beta}] = \beta$$

Sob H1–H5, a variância condicional do estimador é:

$$\mathrm{Var}[\hat{\beta} \mid X] = \sigma^2 (X'X)^{-1}$$

O estimador não-viesado da variância do erro é:

$$s^2 = \frac{\hat{e}'\hat{e}}{n - k}$$

e portanto $\widehat{\mathrm{Var}}[\hat{\beta}] = s^2 (X'X)^{-1}$.

18.4 Verificação por DGP

A prova mais direta de que o estimador está correto é definir um DGP (*data-generating process*) com parâmetros conhecidos, gerar dados a partir dele e verificar que o OLS os recupera.

Escolha os parâmetros verdadeiros:

$$\beta_0 = 2{,}5 \qquad \beta_1 = 1{,}8 \qquad \beta_2 = -0{,}7$$

e gere $n = 1000$ observações *sem* termo de erro ($\varepsilon = 0$):

$$y_i = 2{,}5 + 1{,}8\, x_{1i} - 0{,}7\, x_{2i}$$

Com $\varepsilon = 0$ as hipóteses de Gauss-Markov são satisfeitas de forma exata, e o OLS

deve recuperar $(\beta_0, \beta_1, \beta_2)$ com precisão de máquina.

```
seed(42)
input df
id
1
end
for i in 1..=10 { df = append(df, df) }
df |> filter(_n <= 1000)
generate df x1 = rnormal()
generate df x2 = rnormal()
generate df y  = 2.5 + 1.8*x1 - 0.7*x2
ols(y ~ x1 + x2, df)
```

Listing 18.1: DGP sem ruído — verificação numérica

Saída real do interpretador:

```
========================== OLS Regression Results ==========================
Dep. Variable:                      y || R-squared:                     1.0000
Model:                            OLS || Adj. R-squared:                1.0000
Covariance Type:          Non-Robust || F-statistic:                    0.0000
No. Observations:               1000 || Prob (F-statistic):           1.0000e0
Df Residuals:                    997 || Log-Likelihood:             31513.9419
AIC:                       -63021.8837 || BIC:                       -63007.1605

------------------------------------------------------------------------------
Variable   |       coef |    std err |        t |    P>|t| | [0.025   0.975]
------------------------------------------------------------------------------
const      |     2.5000 |     0.0000 | 6.17e+15 |    0.000 |  2.5000  2.5000
x1         |     1.8000 |     0.0000 | 3.36e+15 |    0.000 |  1.8000  1.8000
x2         |    -0.7000 |     0.0000 |-1.28e+15 |    0.000 | -0.7000 -0.7000
==============================================================================
```

O OLS recupera exatamente $(\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2) = (2{,}5;\ 1{,}8;\ -0{,}7)$. O erro padrão zero e o $R^2 = 1$ são esperados: sem ruído, os dados estão exatamente sobre o hiperplano definido por β, e o estimador é numericamente exato até a precisão de ponto flutuante.

Nota

A estatística F aparece como `0.0` e o p-valor como `1.0` neste caso — artefato numérico de 0/0 quando SQR $= 0$. As estatísticas t são da ordem de 10^{15} pelo mesmo motivo: numerador finito dividido por denominador de precisão de máquina. Em presença de ruído ($\varepsilon \neq 0$) todas as estatísticas assumem valores finitos e interpretáveis.

Efeito de má-especificação

O mesmo DGP permite verificar o que acontece ao omitir x_2. Estimam-se dois modelos e comparam-se com `esttab`:

```
let m1 = ols(y ~ x1 + x2, df)
let m2 = ols(y ~ x1, df)
esttab(m1, m2)
```

Listing 18.2: Modelo correto vs. x_2 omitido

```
                            (1)             (2)
                            OLS             OLS

x1                       1.8000***       1.7422***
                          (0.0000)        (0.0224)
x2                      -0.7000***
                          (0.0000)
Constant                 2.5000***       2.1810***
                          (0.0000)        (0.0129)

N                             1000            1000
R²                          1.0000          0.8585
Adj. R²                     1.0000          0.8583

* p<0.10   ** p<0.05   *** p<0.01
```

O coeficiente de x_1 mal se altera ($1.8000 \to 1.7422$) porque x_1 e x_2 são gerados independentemente — a correlação amostral é próxima de zero. Já a constante distorce de 2.5000 para 2.1810 porque absorve $\hat{\beta}_2 \, \bar{x}_2$, e a média amostral de x_2 não é exatamente zero mesmo com $n = 1000$.

Nota

Viés de variável omitida (OVB) *requer correlação* entre o regressor incluído e o omitido. Sem correlação, a omissão apenas infla os erros padrão e reduz o R^2 (perda de eficiência, não de consistência). Para o tratamento formal de endogeneidade e instrumentos, veja o Capítulo 19.

18.5 Erros padrão

Clássico

Assume homocedasticidade. A matriz de covariância estimada é $s^2(X'X)^{-1}$ e o erro padrão do coeficiente j é $\text{se}_j = s\sqrt{[(X'X)^{-1}]_{jj}}$.

Robusto à heterocedasticidade

Quando $\text{Var}[\varepsilon_i \mid x_i] = \sigma_i^2$ (heterocedasticidade), o estimador sanduíche fornece inferência válida.

HC1 (White com correção de graus de liberdade):

$$\hat{V}_{\text{HC1}} = \frac{n}{n-k}\,(X'X)^{-1}\left(\sum_{i=1}^{n}\hat{e}_i^2\,x_i x_i'\right)(X'X)^{-1}$$

HC3 (mais conservador, recomendado em amostras pequenas):

$$\hat{V}_{\text{HC3}} = (X'X)^{-1}\left(\sum_{i=1}^{n}\frac{\hat{e}_i^2}{(1-h_{ii})^2}\,x_i x_i'\right)(X'X)^{-1}$$

onde $h_{ii} = x_i'(X'X)^{-1}x_i$ é o *leverage* da observação i.

Clusterizado

Quando os erros são correlacionados dentro de grupos $g = 1, \ldots, G$:

$$\hat{V}_{\text{cl}} = \frac{G(n-1)}{(G-1)(n-k)}\,(X'X)^{-1}\left(\sum_{g=1}^{G}X_g'\hat{e}_g\hat{e}_g'X_g\right)(X'X)^{-1}$$

18.6 Inferência

Teste t sobre um coeficiente

Sob H0: $\beta_j = \beta_j^0$:

$$t_j = \frac{\hat{\beta}_j - \beta_j^0}{\text{se}(\hat{\beta}_j)} \;\xrightarrow{d}\; t_{n-k}$$

Teste F sobre restrições lineares

Para $R\beta = r$ com q restrições:

$$F = \frac{(R\hat{\beta} - r)' \left[R\,(X'X)^{-1} R' \right]^{-1} (R\hat{\beta} - r)}{q\, s^2} \xrightarrow{d} F_{q,\,n-k}$$

Nota

Com erros robustos ou clusterizados, $\hat{V}$ substitui $s^2(X'X)^{-1}$ no cálculo do teste F.

18.7 Sintaxe

```
ols(fórmula, df)
ols(fórmula, df, cov=tipo)
ols(fórmula, df, cov=cluster, cluster=coluna)
ols(fórmula, df, if = condição)
```

Alias: `reg()` é equivalente a `ols()`.

18.8 Fórmula

A fórmula segue a notação `y ~ x1 + x2 + ...`:

```
input df
Y X
10 2
12 3
8  1
15 5
11 2
14 4
end

ols(Y ~ X, df)
```

Listing 18.3: OLS básico

A constante é incluída automaticamente. Para múltiplas variáveis:

```
ols(Y ~ X1 + X2 + X3, df)
```

A fórmula pode ser passada como string:

```
let formula = "Y ~ X1 + X2"
ols(formula, df)
```

18.8.1 Transformações

Qualquer função suportada pela linguagem pode ser usada diretamente no RHS da fórmula — o interpretador avalia a expressão *antes* de passar os dados ao estimador, de modo que o Greeners recebe apenas colunas numéricas planas.

```
// log de uma variável
ols(Y ~ log(X), df)

// raiz quadrada
ols(Y ~ sqrt(X), df)

// múltiplas transformações
ols(Y ~ log(K) + log(L), df)

// termo quadrático (equivalente a I(X^2))
ols(Y ~ X + X^2, df)
```

Listing 18.4: Transformações em regressores

O nome do coeficiente na tabela de resultados reflete a expressão original: `log(K)`, `sqrt(X)`, etc.

18.8.2 Interações

O operador `:` cria um termo de interação (produto *element-wise*) entre dois regressores. Cada lado pode ser uma expressão arbitrária:

```
// interação entre variáveis simples
ols(Y ~ X1 + X2 + X1:X2, df)

// interação de transformações - log(K)*log(L)
ols(Y ~ log(K) + log(L) + log(K):log(L), df)

// apenas o termo de interação
ols(Y ~ log(K):log(L), df)
```

Listing 18.5: Interações

O operador `*` é açúcar sintático para expansão completa (`x1*x2` ≡ `x1 + x2 + x1:x2`):

```
// equivalente: ols(Y ~ X1 + X2 + X1:X2, df)
ols(Y ~ X1 * X2, df)
```

18.8.3 `I(expr)` — expressões aritméticas arbitrárias

A função `I(…)` isola uma expressão aritmética composta dentro da fórmula, protegendo os operadores `+`, `-` e `*` do significado especial que têm na gramática de fórmulas:

```
// quadrado de X
ols(Y ~ X + I(X^2), df)

// interação linear construída explicitamente
ols(Y ~ X1 + X2 + I(X1 * X2), df)

// combinação linear de variáveis
ols(Y ~ I(X1 + X2), df)
```

Listing 18.6: Termos arbitrários com `I()`

Nota

`I(X^2)` e `X^2` são equivalentes quando `X^2` aparece sozinho como termo (fora de interações e adições de nível superior). Use `I(…)` para eliminar qualquer ambiguidade ou quando a expressão contém `+` ou `*`.

18.8.4 Variável categórica: `C()`

`C(var)` instrui o estimador a tratar `var` como categórica, gerando dummies automáticas (categoria de referência = primeira por ordem lexicográfica):

```
ols(Y ~ X + C(regiao), df)
```

18.9 Opções de erro padrão

Opção	Fórmula
(padrão)	$s^2(X'X)^{-1}$
`cov=robust`	HC1 (sanduíche White)
`cov=HC1`	idem
`cov=HC3`	HC3 (leverage-ajustado)
`cov=cluster, cluster=col`	clusterizado por `col`

```
ols(Y ~ X, df)                                  // clássico
ols(Y ~ X, df, cov=robust)                      // HC1
ols(Y ~ X, df, cov=HC3)                         // HC3
ols(Y ~ X, df, cov=cluster, cluster=id)         // clusterizado
```

Listing 18.7: Variantes de erro padrão

18.10 Amostra condicional

A opção `if =` restringe a estimação a um subconjunto sem modificar o DataFrame:

```
ols(Y ~ X, df, if = grupo == 1)
```

18.11 Capturando o resultado

```
let m = ols(Y ~ X, df)
display m
```

Quando atribuído, `ols()` não imprime a tabela — apenas armazena o modelo.

18.12 Pós-estimação

18.12.1 `predict`

```
let m = ols(Y ~ X, df)

// $\hat{y} = X\hat{\beta}$
let yhat = predict(m, "xb")
// $\hat{e} = y - X\hat{\beta}$
let ehat = predict(m, "residuals")
```

Aliases aceitos: `"fitted"` para valores ajustados; `"resid"` ou `"e"` para resíduos.

18.12.2 `vif`

O VIF do regressor j é $\text{VIF}_j = 1/(1 - R_j^2)$, onde R_j^2 é o R^2 da regressão de x_j sobre os demais regressores. Valores acima de 10 indicam multicolinearidade severa.

```
let m = ols(Y ~ X1 + X2 + X3, df)
vif(m)
```

18.12.3 `test`

Teste F de restrições lineares $H_0 : R\beta = r$:

```
let m = ols(Y ~ X1 + X2 + X3, df)
test(m, X2 = 0, X3 = 0)     // H0: X2 = X3 = 0
```

18.12.4 `lincom`

Combinação linear $c'\hat{\beta}$ com erro padrão $\sqrt{c'\hat{V}c}$:

```
let m = ols(Y ~ X1 + X2, df)
lincom(m, X1 + X2)
```

18.12.5 `margins`

Efeitos marginais médios. No OLS linear $\partial E[y]/\partial x_j = \beta_j$, equivalente aos coeficientes.

```
let m = ols(Y ~ X, df)
margins(m)
```

18.13 Tabela de resultados com `esttab`

```
let m1 = ols(Y ~ X, df)
let m2 = ols(Y ~ X, df, cov=robust)
let m3 = ols(Y ~ X1 + X2, df)

esttab(m1, m2, m3)
```

Listing 18.8: Comparação de especificações

```
eststo(ols(Y ~ X, df))
eststo(ols(Y ~ X1 + X2, df))
esttab()
```

18.14 Bootstrap

Erros padrão por bootstrap — alternativa não-paramétrica:

```
bootstrap(ols, Y ~ X, df, n=1000)
bootse(m, n=500)
```

Capítulo 19

Variáveis Instrumentais

19.1 O problema de endogeneidade

Considere o modelo estrutural:

$$y = X_1\beta_1 + X_2\beta_2 + \varepsilon$$

onde X_1 são regressores exógenos ($n \times k_1$) e X_2 são regressores potencialmente endógenos ($n \times k_2$). O OLS é viesado e inconsistente quando:

$$E[\varepsilon \mid X_2] \neq 0 \quad \Longleftrightarrow \quad \operatorname{plim} \hat{\beta}_{\text{OLS}} \neq \beta$$

As causas mais comuns são:

- **Variável omitida** correlacionada com X_2: se $y = X\beta + W\gamma + u$ e W é omitida, então $\varepsilon = W\gamma + u$ e $\operatorname{Cov}(X_2, \varepsilon) \neq 0$.
- **Simultaneidade**: y e X_2 são determinados conjuntamente.
- **Erro de medida clássico**: $X_2 = X_2^* + \nu$, com X_2^* verdadeiro; o atenuante é $\operatorname{plim} \hat{\beta} = \beta \cdot \sigma^2_{X^*}/\sigma^2_X < \beta$.

19.2 Condições de identificação

Seja Z a matriz $n \times L$ de instrumentos (incluindo X_1 como instrumentos incluídos). Um instrumento válido satisfaz duas condições:

1. **Relevância:** $\operatorname{rank}\Big(E[z_i x_i']\Big) = k$, onde $k = k_1 + k_2$. Na prática: X_2 é correlacionado com Z na primeira etapa.

2. **Exogeneidade (restrição de exclusão):** $E[z_i \varepsilon_i] = 0$. Os instrumentos excluídos afetam y *somente* através de X_2.

O modelo é identificado quando $L \geq k$ (condição de ordem). Para $L = k$ (identificação exata) o estimador IV é único; para $L > k$ (sobreidentificação) o 2SLS é eficiente na classe dos estimadores IV.

19.3 O estimador 2SLS

Defina a matriz de projeção sobre o espaço coluna de Z:

$$P_Z = Z(Z'Z)^{-1}Z'$$

O estimador de Mínimos Quadrados em Dois Estágios (2SLS) é:

$$\boxed{\hat{\beta}_{\text{2SLS}} = (X'P_Z X)^{-1} X'P_Z y}$$

Interpretação em dois estágios

Primeiro estágio — projeta cada coluna de X_2 sobre Z:

$$X_2 = Z\hat{\Pi} + \hat{V}, \qquad \hat{\Pi} = (Z'Z)^{-1}Z'X_2$$

Segundo estágio — regride y sobre $\hat{X} = P_Z X = [X_1,\ \hat{X}_2]$:

$$\hat{\beta}_{\text{2SLS}} = (\hat{X}'X)^{-1}\hat{X}'y$$

Atenção

Nunca execute os dois estágios manualmente como OLS sequencial. Os erros padrão do segundo estágio estarão errados porque $\hat{X}_2$ é estimado. Use sempre `iv()`, que corrige a variância automaticamente.

19.4 Variância assintótica

Sob exogeneidade e relevância, $\hat{\beta}_{\text{2SLS}}$ é consistente e assintoticamente normal:

$$\sqrt{n}\,(\hat{\beta}_{\text{2SLS}} - \beta) \xrightarrow{d} \mathcal{N}\!\left(0,\ \sigma^2\, Q_{XZ}^{-1} Q_{ZZ} Q_{XZ}^{-1\prime}\right)$$

onde $Q_{XZ} = E[x_i z_i']$ e $Q_{ZZ} = E[z_i z_i']$.

O estimador da variância é:

$$\widehat{\mathrm{Var}}[\hat{\beta}_{\mathrm{2SLS}}] = s^2_{\mathrm{IV}}\,(X'P_ZX)^{-1}, \qquad s^2_{\mathrm{IV}} = \frac{\hat{e}'_{\mathrm{2SLS}}\,\hat{e}_{\mathrm{2SLS}}}{n-k}$$

As mesmas correções sanduíche do OLS (HC1, HC3, clustered) aplicam-se substituindo X por $\hat{X} = P_ZX$.

19.5 Instrumentos fracos

Um instrumento é fraco quando a correlação com X_2 é baixa. O viés do 2SLS em relação ao OLS é aproximadamente:

$$\frac{\mathrm{Viés}_{\mathrm{2SLS}}}{\mathrm{Viés}_{\mathrm{OLS}}} \approx \frac{1}{F+1}$$

onde F é a estatística F da primeira etapa. Para $F < 10$ (regra de Staiger & Stock, 1997) o instrumento é considerado fraco.

A **estatística de Cragg-Donald** generaliza para múltiplos regressores endógenos:

$$CD = \frac{n-L}{L-k_1}\cdot\lambda_{\min}$$

onde $\lambda_{\min}$ é o menor autovalor da matriz de concentração. Stock & Yogo (2005) tabelam valores críticos para viés máximo relativo de 5%, 10%, 20% e 30% em relação ao OLS.

19.6 Verificação por DGP

O mesmo método do Capítulo 18 se aplica ao IV: define-se um DGP com endogeneidade conhecida, estima-se OLS e IV, e verifica-se que o IV corrige o viés.

Estrutura do DGP

Sejam z e v variáveis aleatórias independentes $\sim \mathcal{N}(0,1)$. Defina ε como o *resíduo* da projeção de v sobre $[1,\ z]$:

$$\varepsilon = v - \delta_0 - \delta_1 z, \qquad \delta_1 = \frac{\mathrm{Cov}_n(z,v)}{\mathrm{Var}_n(z)}, \quad \delta_0 = \bar{v} - \delta_1\bar{z}$$

Por construção $\sum \varepsilon_i = 0$ e $\sum z_i\varepsilon_i = 0$, ou seja, $Z'\varepsilon = 0$ *exatamente* na amostra. Então:

$$x = 0{,}8\,z + \varepsilon \qquad y = 1{,}5 + 2{,}0\,x + 0{,}7\,\varepsilon$$

O erro estrutural $0{,}7\varepsilon$ cria endogeneidade ($\mathrm{Cov}(x, 0{,}7\varepsilon) \neq 0$), mas como $Z'\varepsilon = 0$ exatamente, as condições de momento do IV são satisfeitas sem ruído amostral. Resultado: IV recupera (β_0, β_1) com precisão de máquina — o mesmo que OLS fez sem ruído no Capítulo 18.

O viés do OLS é previsível:

$$\operatorname{plim} \hat{\beta}_{\mathrm{OLS}} = \beta_1 + \frac{\mathrm{Cov}(x, \varepsilon)}{\mathrm{Var}(x)} \approx 2{,}0 + \frac{0{,}7}{0{,}64 + 1} \approx 2{,}43$$

Script

```
seed(42)
input df
id
1
end
for i in 1..=10 { df = append(df, df) }
df |> filter(_n <= 1000)
generate df z   = rnormal()
generate df v   = rnormal()
let d1          = cov(df, z, v) / variance(df, z)
let d0          = mean(df, v) - d1 * mean(df, z)
generate df eps = v - d0 - d1*z
generate df x   = 0.8*z + eps
generate df y   = 1.5 + 2.0*x + 0.7*eps
let m_ols = ols(y ~ x, df)
let m_iv  = iv(y ~ x, ~ z, df)
esttab(m_ols, m_iv)
```

Listing 19.1: DGP com endogeneidade — OLS vs. IV

Saída real do interpretador:

```
                            (1)             (2)
                            OLS         IV/2SLS

x                     2.4303***       2.0000***
                       (0.0108)        (0.0280)
const                                 1.5000***
                                       (0.0129)
```

```
Constant                1.3273***
                         (0.0058)

N                           1000              1000
R²                        0.9807
Adj. R²                   0.9807

* p<0.10   ** p<0.05   *** p<0.01
```

O IV recupera $(\hat{\beta}_0, \hat{\beta}_1) = (1{,}5000,\ 2{,}0000)$ com precisão de máquina. O OLS superestima β_1 em $+0{,}43$ (2.4303 vs. 2.0000), consistente com o viés assintótico previsto de $+0{,}43$. O R^2 não é reportado para IV pois não tem interpretação padrão sob endogeneidade.

19.7 Sintaxe

```
iv(fórmula_estrutural, fórmula_instrumentos, df)
iv(fórmula_estrutural, fórmula_instrumentos, df, cov=tipo)
```

A *fórmula estrutural* especifica a equação de interesse. A *fórmula dos instrumentos* lista os instrumentos excluídos após `~`:

```
// equação estrutural: y ~ x  (x é endógeno)
// instrumento excluído: z
iv(y ~ x, ~ z, df)
```

Listing 19.2: IV básico — um instrumento

Para múltiplos regressores endógenos e instrumentos:

```
iv(y ~ x1 + x2, ~ z1 + z2, df)
```

19.7.1 Transformações e interações nas fórmulas

As mesmas transformações e interações disponíveis no OLS (Seção 18.8.1–18.8.3) funcionam igualmente nas fórmulas estrutural e de instrumentos do IV:

```
// regressor endógeno transformado
iv(Y ~ log(X2), ~ Z, df)

// interação de transformações na equação estrutural
iv(Y ~ log(K) + log(L) + log(K):log(L), ~ Z1 + Z2, df)
```

```
6
7 // instrumento transformado
8 iv(Y ~ X2, ~ log(Z), df)
```

Listing 19.3: IV com transformações e interações

Nota

Os regressores exógenos incluídos (X_1) não precisam ser listados na fórmula dos instrumentos — são adicionados automaticamente como seus próprios instrumentos.

19.8 Opções de erro padrão

As mesmas opções de `cov=` disponíveis no OLS aplicam-se ao IV:

```
1 iv(y ~ x, ~ z, df)                                  // clássico
2 iv(y ~ x, ~ z, df, cov=robust)                      // HC1
3 iv(y ~ x, ~ z, df, cov=HC3)                         // HC3
4 iv(y ~ x, ~ z, df, cov=cluster, cluster=id)         // clusterizado
```

19.9 Capturando o resultado

```
1 let m_iv = iv(y ~ x, ~ z, df)
2 display m_iv
```

19.10 Diagnóstico de instrumentos fracos

```
1 weak_iv(y ~ x, ~ z, df)
```

Listing 19.4: Teste de instrumentos fracos

A função reporta:

- Estatística F da primeira etapa (por variável endógena)
- Estatística de Cragg-Donald CD
- Valores críticos de Stock & Yogo (2005) para viés relativo de 5%–30%
- Referência: $F > 10$ (Staiger & Stock, 1997)

19.11 Valores ajustados

```
let m_iv = iv(y ~ x, ~ z, df)
// $\hat{y} = X\hat{\beta}_{\mathrm{2SLS}}$
let yhat = predict(m_iv, "xb")
```

19.12 Tabela comparativa

O fluxo típico quantifica o viés de endogeneidade comparando OLS e IV:

```
load "dados.csv" as df

let m_ols = ols(salario ~ educ, df)
let m_iv  = iv(salario ~ educ, ~ distancia, df, cov=robust)

// diagnóstico da primeira etapa
weak_iv(salario ~ educ, ~ distancia, df)

// tabela comparativa
esttab(m_ols, m_iv)
```

Listing 19.5: Fluxo completo de análise IV

Se $\hat{\beta}^{\text{IV}} > \hat{\beta}^{\text{OLS}}$, o viés do OLS é negativo — o que é consistente com erro de medida atenuante ou com variável omitida que deprime tanto X_2 quanto y.

Capítulo 20

Dados em Painel

20.1 O modelo

Um painel observa n indivíduos ao longo de T períodos. O modelo geral é:

$$y_{it} = \alpha_i + X_{it}\beta + \varepsilon_{it}, \qquad i = 1, \ldots, n; \quad t = 1, \ldots, T$$

onde α_i é o **efeito individual não observado** — características fixas no tempo que diferem entre indivíduos (habilidade, cultura, localização). O OLS em corte transversal ou empilhado ignora α_i, tratando-o como parte do erro:

$$\operatorname{plim} \hat{\beta}_{\text{OLS}} = \beta + \frac{\operatorname{Cov}(x_{it},\, \alpha_i)}{\operatorname{Var}(x_{it})}$$

O viés só desaparece se $\operatorname{Cov}(x_{it}, \alpha_i) = 0$ — hipótese forte na maioria das aplicações.

20.2 Efeitos Fixos

O estimador **within** elimina α_i subtraindo a média temporal de cada indivíduo:

$$\tilde{y}_{it} = y_{it} - \bar{y}_i, \qquad \tilde{X}_{it} = X_{it} - \bar{X}_i$$

O estimador de Efeitos Fixos é então o OLS sobre as variáveis desmediadas:

$$\boxed{\hat{\beta}_{\text{FE}} = (\tilde{X}'\tilde{X})^{-1}\tilde{X}'\tilde{y}}$$

Como a transformação within anula α_i para qualquer valor de $\operatorname{Cov}(x_{it}, \alpha_i)$, o estimador é consistente mesmo sob endogeneidade do efeito individual. A variância

é:

$$\widehat{\mathrm{Var}}[\hat{\beta}_{\mathrm{FE}}] = s^2_{\mathrm{FE}}(\tilde{X}'\tilde{X})^{-1}, \qquad s^2_{\mathrm{FE}} = \frac{\hat{\varepsilon}'\hat{\varepsilon}}{nT - n - k}$$

Atenção

Variáveis que não variam no tempo (sexo, raça, região de nascimento) são eliminadas pela transformação within. FE não pode estimar seu efeito. Use RE ou estimadores híbridos nesses casos.

20.3 Efeitos Aleatórios

Se $\alpha_i \sim \mathrm{IID}(0, \sigma^2_\alpha)$ e $\mathrm{Cov}(x_{it}, \alpha_i) = 0$, α_i é parte do erro composto $v_{it} = \alpha_i + \varepsilon_{it}$. A estrutura de erros implica GLS com quasi-desmediação:

$$y^*_{it} = y_{it} - \theta\bar{y}_i, \qquad \theta = 1 - \frac{\sigma_\varepsilon}{\sqrt{\sigma^2_\varepsilon + T\sigma^2_\alpha}} \in [0, 1)$$

O estimador RE é o OLS sobre y^*_{it} e X^*_{it}. É mais eficiente que FE sob H_0 : $\mathrm{Cov}(x_{it}, \alpha_i) = 0$, mas *inconsistente* quando a hipótese falha.

20.4 Teste de Hausman

O teste de Hausman compara FE (consistente sempre) com RE (consistente sob H_0, eficiente):

$$H_0 : \mathrm{Cov}(x_{it}, \alpha_i) = 0 \quad \Longrightarrow \quad \mathrm{plim}(\hat{\beta}_{\mathrm{FE}} - \hat{\beta}_{\mathrm{RE}}) = 0$$

A estatística é:

$$H = (\hat{\beta}_{\mathrm{FE}} - \hat{\beta}_{\mathrm{RE}})'\left[\widehat{\mathrm{Var}}(\hat{\beta}_{\mathrm{FE}}) - \widehat{\mathrm{Var}}(\hat{\beta}_{\mathrm{RE}})\right]^{-1}(\hat{\beta}_{\mathrm{FE}} - \hat{\beta}_{\mathrm{RE}}) \xrightarrow{d} \chi^2(k)$$

Rejeitar H_0 indica que RE é inconsistente — use FE.

20.5 Verificação por DGP

Recuperação exata (sem ruído)

O DGP usa 100 indivíduos × 10 períodos ($n = 1000$). O efeito fixo $\alpha_i = 0{,}05 \cdot i$ é determinístico e correlacionado com x:

$$x_{it} = \alpha_i + u_{it}, \quad u \sim \mathcal{N}(0, 1) \qquad y_{it} = 1{,}0 + 2{,}0\, x_{it} + \alpha_i \quad (\varepsilon = 0)$$

Sem ruído idiossincrático a transformação within elimina α_i exatamente, e FE recupera β com precisão de máquina:

```
seed(42)
input df
id
1
end
for i in 1..=10 { df = append(df, df) }
df |> filter(_n <= 1000)
generate df id    = (_n - 1) % 100 + 1
generate df alpha = id * 0.05
generate df u     = rnormal()
generate df x     = alpha + u
generate df y     = 1.0 + 2.0*x + alpha
let m_ols = ols(y ~ x, df)
let m_fe  = fe(y ~ x, df, id=id)
esttab(m_ols, m_fe)
```

Listing 20.1: FE com efeito fixo correlacionado — sem ruído

```
                            (1)             (2)
                            OLS              FE

x                      2.9663***       2.0000***
                        (0.0060)        (0.0000)
Constant               0.6002***
                        (0.0202)

N                           1000            1000
R²                        0.9959
Adj. R²                   0.9959

* p<0.10   ** p<0.05   *** p<0.01
```

O OLS superestima β em +0,97 porque $\text{Cov}(x, \alpha) = \text{Var}(\alpha) > 0$. O FE recupera 2,0000 exatamente.

Hausman com ruído

Com ruído idiossincrático $\varepsilon \sim \mathcal{N}(0, 1)$, o FE permanece consistente e o RE é inconsistente (pois α_i ainda é correlacionado com x). O Hausman deve rejeitar H_0:

```
// mesmo DGP; adiciona ruído idiossincrático
generate df e  = rnormal()
generate df y2 = 1.0 + 2.0*x + alpha + e
let m_fe2 = fe(y2 ~ x, df, id=id)
let m_re2 = re(y2 ~ x, df, id=id)
esttab(m_fe2, m_re2)
hausman(m_fe2, m_re2)
```

Listing 20.2: Hausman — FE vs RE com efeito correlacionado

```
                            (1)             (2)
                             FE              RE

x                       2.0977***       2.9347***
                         (0.0338)        (0.0118)
const                                   1.1930***
                                         (0.0399)

N                           1000               0

* p<0.10   ** p<0.05   *** p<0.01


 Hausman Test: FE vs RE
 H: efeitos individuais não correlacionados com regressores

 Coeficientes Comuns
   Variável          _FE        _RE Δ
   x              2.0977     2.9347    -0.8369
 Estatística
   ²(1) = 698.1711    p = 0.0000   ***
 Conclusão
   Rejeita  H → use EFEITOS FIXOS (RE pode ser inconsistente)
```

O RE absorve o efeito de α_i no coeficiente de x (2.9347), replicando o viés do OLS. O Hausman rejeita H_0 com $p < 0{,}001$ — a endogeneidade do efeito individual é detectada.[1]

[1] O FE recupera 2.0977, não 2.0000, porque a transformação within elimina α_i mas não ε_{it}. Com ruído idiossincrático presente, o estimador é consistente mas não exato — o mesmo motivo pelo qual OLS e IV divergem de seus parâmetros verdadeiros quando $\varepsilon \neq 0$.

20.6 Sintaxe

```
fe(fórmula, df, id=col)
re(fórmula, df, id=col)
hausman(modelo_fe, modelo_re)
```

A coluna `id` identifica o indivíduo. O FE remove automaticamente o intercepto (absorvido pelos efeitos individuais).

```
load "painel.csv" as df

let m_fe = fe(salario ~ educ + exp, df, id=id)
let m_re = re(salario ~ educ + exp, df, id=id)

esttab(m_fe, m_re)
hausman(m_fe, m_re)
```

Listing 20.3: Fluxo completo

20.7 Opções de erro padrão

```
fe(y ~ x, df, id=id)                          // clássico
fe(y ~ x, df, id=id, cov=robust)              // HC1
// cluster por indivíduo
fe(y ~ x, df, id=id, cov=cluster, cluster=id)
```

Nota

Em painéis, erros clusterizados por indivíduo (`cluster=id`) são o padrão recomendado: corrigem heterocedasticidade e correlação serial dentro do indivíduo simultaneamente.

20.8 Capturando resultados

```
let m = fe(y ~ x, df, id=id)
display m
let yhat = predict(m, "xb")
```

Capítulo 21

Diferenças em Diferenças

21.1 O modelo

Diferenças em Diferenças (DiD) estima o efeito causal de um tratamento comparando a variação temporal do grupo tratado com a do grupo controle:

$$\hat{\delta}_{\text{DiD}} = \underbrace{(\bar{Y}_{1,\text{pós}} - \bar{Y}_{1,\text{pré}})}_{\text{variação tratado}} - \underbrace{(\bar{Y}_{0,\text{pós}} - \bar{Y}_{0,\text{pré}})}_{\text{variação controle}}$$

A tabela 2×2 resume os quatro grupos:

	Pré	**Pós**
Controle ($D = 0$)	μ_{00}	μ_{01}
Tratado ($D = 1$)	μ_{10}	μ_{11}

$$\hat{\delta} = (\mu_{11} - \mu_{10}) - (\mu_{01} - \mu_{00})$$

21.2 Forma de regressão

O estimador DiD é equivalente ao coeficiente da interação em:

$$y_{it} = \beta_0 + \beta_1 \,\text{Post}_t + \beta_2 \,\text{Treated}_i + \delta \,(\text{Treated}_i \times \text{Post}_t) + \varepsilon_{it}$$

onde $\hat{\delta} = \hat{\beta}_3$ é o efeito médio do tratamento sobre os tratados (ATT — *Average Treatment effect on the Treated*).

21.3 Hipótese de tendências paralelas

A identificação do ATT repousa sobre:

$$E\left[Y_{it}^{(0)} - Y_{is}^{(0)} \mid D_i = 1\right] = E\left[Y_{it}^{(0)} - Y_{is}^{(0)} \mid D_i = 0\right] \qquad \forall\, t \neq s$$

Na ausência do tratamento, tratados e controles teriam a *mesma* tendência temporal. A suposição é não testável no período pós-tratamento, mas pode ser verificada nos períodos pré-tratamento quando há mais de um período disponível.

Atenção

A hipótese de tendências paralelas é mais forte do que parece. Grupos com tendências de crescimento estruturalmente diferentes (ex. regiões em expansão vs. regiões estagnadas) violam a hipótese mesmo que os níveis sejam similares no pré-tratamento.

21.4 Verificação por DGP

Recuperação exata (sem ruído)

O DGP usa 500 indivíduos $\times$ 2 períodos ($n = 1000$). Os primeiros 250 são tratados, os demais são controle. O efeito verdadeiro é $\delta = 3{,}0$:

$$y_{it} = 1{,}0 + 0{,}5 \cdot \text{Post}_t + 3{,}0 \cdot (D_i \times \text{Post}_t)$$

As quatro médias de célula são $\mu_{00} = 1{,}0$, $\mu_{01} = 1{,}5$, $\mu_{10} = 1{,}0$ e $\mu_{11} = 4{,}5$, de modo que $\hat{\delta} = (4{,}5 - 1{,}0) - (1{,}5 - 1{,}0) = 3{,}0$ exatamente.

```
seed(42)
input df
id
1
end
for i in 1..=10 { df = append(df, df) }
df |> filter(_n <= 1000)
generate df id      = (_n - 1) % 500 + 1
generate df post    = _n > 500
generate df treated = id <= 250
generate df att     = treated * post
generate df y       = 1.0 + 0.5*post + 3.0*att
did(y ~ treated + post, df)
```

Listing 21.1: DGP DiD sem ruído — recuperação exata

```
================= Difference-in-Differences (2x2 Canonical) ================
ATT (Effect):                 3.0000 || R-squared:
1.0000
Std. Error:                   0.0000 || P-value:
0.0000
t-statistic:                     inf || Observations:
1000

------------------------------- Group Means --------------------------------
Control Group (Pre):      1.0000 | Control Group (Post):     1.5000
Treated Group (Pre):      1.0000 | Treated Group (Post):     4.5000
Parallel Trend Diff:      0.0000 (If > 0, Control grew more than Treated Pre
============================================================================
```

Com ruído idiossincrático

Adicionando $\varepsilon \sim \mathcal{N}(0,1)$, o DiD permanece consistente:

```
generate df e = rnormal()
generate df y = 1.0 + 0.5*post + 3.0*att + e
did(y ~ treated + post, df)
```

Listing 21.2: DGP DiD com ruído

```
================= Difference-in-Differences (2x2 Canonical) ================
ATT (Effect):                 3.0114 || R-squared:
0.9645
Std. Error:                   0.0358 || P-value:
0.0000
t-statistic:                 84.0942 || Observations:
1000

------------------------------- Group Means --------------------------------
Control Group (Pre):      1.4842 | Control Group (Post):     2.0099
Treated Group (Pre):      1.4880 | Treated Group (Post):     5.0251
Parallel Trend Diff:      0.0000 (If > 0, Control grew more than Treated Pre
============================================================================
```

O ATT de 3,0114 está próximo do verdadeiro 3,0; o desvio é variância amostral. O *Parallel Trend Diff* próximo de zero confirma que a hipótese de tendências paralelas é satisfeita por construção.

21.5 Sintaxe

```
did(outcome ~ treated + post, df)
did(outcome ~ treated + post, df, cov=robust)
```

A fórmula requer exatamente duas variáveis no lado direito: a indicadora de grupo (`treated`) e a indicadora de período (`post`). O Hayashi calcula a interação internamente.

```
load "dados.csv" as df

// cria indicadoras
generate df post    = ano >= 2020
generate df treated = estado == "RS"

let m = did(salario ~ treated + post, df, cov=robust)
display m
```

Listing 21.3: Fluxo típico de DiD

21.6 Capturando o resultado

```
let m = did(y ~ treated + post, df)
display m
```

Nota

Para estudos de evento com múltiplos períodos pré e pós-tratamento, o DiD canônico 2×2 pode ser insuficiente. Nesses casos, estime o modelo de regressão com interações período a período e verifique a ausência de efeitos pré-tratamento como teste de tendências paralelas.

Capítulo 22

Logit e Probit

22.1 O modelo de variável latente

Quando a variável dependente é binária ($y \in \{0, 1\}$), o modelo linear produz previsões fora do intervalo $[0, 1]$ e erros heterocedásticos por construção. A solução é modelar a probabilidade de $y = 1$ via uma função de distribuição acumulada:

$$P(y_i = 1 \mid x_i) = F(x_i'\beta)$$

A motivação vem do modelo de variável latente:

$$y_i^* = x_i'\beta + \varepsilon_i, \qquad y_i = \mathbf{1}[y_i^* > 0]$$

A distribuição de ε determina o modelo:

Modelo	**Distribuição de** ε	$F(\cdot)$
Logit	Logística$(0, \pi^2/3)$	$\Lambda(z) = \dfrac{1}{1+e^{-z}}$
Probit	Normal$(0, 1)$	$\Phi(z) = \displaystyle\int_{-\infty}^{z} \phi(t)\,dt$

A estimação é por Máxima Verossimilhança (MLE). A log-verossimilhança é:

$$\ell(\beta) = \sum_{i=1}^{n} \Big[y_i \ln F(x_i'\beta) + (1 - y_i) \ln\Big(1 - F(x_i'\beta)\Big) \Big]$$

22.2 Interpretação dos coeficientes

Os coeficientes do Logit e Probit **não são efeitos marginais** — são parâmetros da variável latente, sem unidade interpretável diretamente. Para x_j contínuo, o efeito marginal sobre a probabilidade é:

$$\frac{\partial P(y=1 \mid x)}{\partial x_j} = f(x'\beta)\,\beta_j$$

onde $f = F'$ é a densidade da distribuição escolhida. Como $f(x'\beta)$ varia entre observações, reporta-se o **Efeito Marginal Médio** (AME):

$$\text{AME}_j = \frac{1}{n}\sum_{i=1}^{n} f(x_i'\beta)\,\beta_j$$

Nota

A relação entre os coeficientes de Logit e Probit é aproximadamente:

$$\hat{\beta}_{\text{Logit}} \approx \frac{\pi}{\sqrt{3}}\,\hat{\beta}_{\text{Probit}} \approx 1{,}81\,\hat{\beta}_{\text{Probit}}$$

Os AMEs dos dois modelos são tipicamente muito próximos, mesmo com coeficientes em escalas diferentes.

22.3 Verificação por DGP

Consistência do MLE

Ao contrário dos modelos lineares dos capítulos anteriores, Logit e Probit *não admitem recuperação exata* dos parâmetros. A variável binária y é gerada por amostragem de Bernoulli — há aleatoriedade irredutível mesmo sem adicionar ruído extra. O que se verifica aqui é **consistência**: com $n = 1000$ o MLE deve convergir próximo dos parâmetros verdadeiros.

O DGP usa a parametrização logística com $\beta_0 = -1{,}0$ e $\beta_1 = 2{,}0$:

$$x_i \sim \mathcal{N}(0,1), \qquad p_i = \Lambda(-1{,}0 + 2{,}0\,x_i), \qquad y_i \sim \text{Bernoulli}(p_i)$$

Bernoulli individual é gerado por $y_i = \mathbf{1}[u_i < p_i]$, com $u_i \sim \text{Uniforme}(0,1)$:

```
seed(42)
input df
id
1
end
for i in 1..=10 { df = append(df, df) }
df |> filter(_n <= 1000)
generate df x  = rnormal()
generate df lp = -1.0 + 2.0*x
```

```
generate df p  = 1 / (1 + exp(-lp))
generate df u  = runiform()
generate df y  = u < p
let m_l = logit(y ~ x, df)
let m_p = probit(y ~ x, df)
esttab(m_l, m_p)
```

Listing 22.1: DGP logístico — Logit e Probit simultâneos

Os verdadeiros parâmetros são $(\beta_0, \beta_1) = (-1{,}0;\ 2{,}0)$ para o DGP Logit com $x_i \sim \mathcal{N}(0,1)$. As estimativas Probit estão na escala normal do índice latente; portanto, não devem ser numericamente idênticas aos coeficientes Logit mesmo quando os dois modelos descrevem a mesma relação qualitativa.

Efeitos marginais médios

Os coeficientes são apenas parâmetros da variável latente. Para interpretação econômica, usa-se `margins`:

```
margins(m_l, df)
```

Listing 22.2: AME do Logit

O efeito marginal é positivo e varia com a probabilidade ajustada; ele é reportado por `margins(m_l, df)` na escala de probabilidade.

22.4 Sintaxe

```
logit(y ~ x1 + x2, df)
probit(y ~ x1 + x2, df)
margins(modelo, df)
```

```
load "dados.csv" as df

let m_l = logit(empregado ~ educ + exp + sexo, df)
let m_p = probit(empregado ~ educ + exp + sexo, df)

esttab(m_l, m_p)
margins(m_l, df)
```

Listing 22.3: Fluxo típico — escolha discreta

22.5 Previsão de probabilidades

```
let m = logit(y ~ x, df)
let phat = predict(m, "pr")      // P(y=1|x) para cada obs
let yhat = predict(m, "class")   // classe prevista (0 ou 1)
```

Nota

Use `predict(m, "class")` com limiar padrão de 0,5. Em dados desbalanceados (prevalência baixa), ajuste o limiar com `predict(m, "class", threshold=0.3)`.

Capítulo 23

Regressão Quantílica

23.1 O modelo

O OLS estima $E[y \mid x]$ — a média condicional. A Regressão Quantílica (QR) estima $Q_\tau[y \mid x]$ — o τ-ésimo quantil condicional para qualquer $\tau \in (0, 1)$:

$$Q_\tau(y \mid x) = x'\beta(\tau)$$

O vetor $\beta(\tau)$ varia com τ: cada quantil tem seu próprio conjunto de coeficientes. Com um único modelo é possível descrever a distribuição condicional inteira de y.

23.2 O estimador

O estimador minimiza a **função de perda assimétrica** (check function):

$$\hat{\beta}(\tau) = \arg\min_{\beta} \sum_{i=1}^{n} \rho_\tau(y_i - x_i'\beta)$$

onde

$$\rho_\tau(u) = \begin{cases} \tau\, u & \text{se } u \geq 0 \\ (\tau - 1)\, u & \text{se } u < 0 \end{cases}$$

Para $\tau = 0{,}5$: $\rho_{0,5}(u) = |u|/2$ — minimização da soma dos desvios absolutos (LAD/mediana). Para $\tau \neq 0{,}5$, resíduos positivos e negativos recebem pesos assimétricos, deslocando a estimativa para o τ-ésimo quantil.

Nota

Ao contrário do OLS, a QR não tem solução de forma fechada — o problema é um programa linear resolvido por simplex ou métodos de ponto interior. Por isso a inferência padrão usa bootstrap (`boot=`).

23.3 Vantagens sobre o OLS

- **Heterocedasticidade:** quando $\mathrm{Var}[y \mid x]$ varia com x, os slopes $\beta_j(\tau)$ diferem entre quantis. O OLS captura apenas a média ponderada implícita.
- **Robustez:** a função check é linear nos resíduos, não quadrática. Outliers em y influenciam menos $\hat{\beta}(\tau)$ do que $\hat{\beta}_{\mathrm{OLS}}$.
- **Distribuição assimétrica:** salários, renda e preços de ativos têm caudas pesadas à direita. A QR descreve efeitos nas caudas sem suposições sobre a forma da distribuição.

23.4 Verificação por DGP

O DGP abaixo introduz heterocedasticidade multiplicativa: a dispersão de y cresce com x, de modo que o efeito de x é maior nos quantis superiores do que nos inferiores.

$$y_i = 2{,}0 + 1{,}5\,x_i + (1{,}0 + 0{,}8\,x_i)\,\varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0,1)$$

O quantil condicional $Q_\tau(y \mid x)$ é crescente em x, e a inclinação $\partial Q_\tau / \partial x$ aumenta com τ: indivíduos nos quantis superiores são mais sensíveis a x do que os do quantil inferior.

```
seed(42)
input df
id
1
end
for i in 1..=10 { df = append(df, df) }
df |> filter(_n <= 1000)
generate df x = rnormal()
generate df e = rnormal()
generate df y = 2.0 + 1.5*x + (1.0 + 0.8*x)*e
let m_ols = ols(y ~ x, df)
let m_q25 = qreg(y ~ x, df, tau=0.25)
```

```
let m_q50 = qreg(y ~ x, df, tau=0.50)
let m_q75 = qreg(y ~ x, df, tau=0.75)
esttab(m_ols, m_q25, m_q50, m_q75)
```

Listing 23.1: DGP heterocedástico — OLS vs. quatro quantis

	(1)	(2)	(3)	(4)
	OLS	QReg (=0.25)	QReg (=0.50)	QReg (=0.75)
x	1.9947***	1.8739***	2.0149***	2.2462***
	(0.0454)	(0.0687)	(0.0833)	(0.0579)
const		2.1659***	2.4525***	2.6637***
		(0.0302)	(0.0433)	(0.0369)
Constant	2.4538***			
	(0.0261)			
N	1000	0	0	0
R^2	0.6594			
Adj. R^2	0.6591			

* p<0.10 ** p<0.05 *** p<0.01

O slope de x cresce monotonicamente: 1,87 no Q25, 2,01 no Q50 e 2,25 no Q75. O OLS (1,99) é uma média ponderada implícita desses efeitos heterogêneos — resume em um único número o que a QR decompõe em toda a distribuição condicional.

A constante também aumenta com τ (2,17 $\rightarrow$ 2,45 $\rightarrow$ 2,66), refletindo que os quantis superiores de y estão acima da média para qualquer valor de x.

23.5 Sintaxe

```
qreg(y ~ x1 + x2, df, tau=0.5)
qreg(y ~ x1 + x2, df, tau=0.5, boot=500)
```

O parâmetro `tau` define o quantil desejado ($\tau \in (0,1)$, padrão 0,5). O parâmetro `boot` ativa bootstrap para os erros padrão (padrão 200 réplicas).

```
load "pnad.csv" as df

let q10 = qreg(sal ~ edu + exp, df, tau=0.10, boot=500)
let q50 = qreg(sal ~ edu + exp, df, tau=0.50, boot=500)
let q90 = qreg(sal ~ edu + exp, df, tau=0.90, boot=500)
let ols = ols(sal ~ edu + exp, df)

esttab(ols, q10, q50, q90)
```

Listing 23.2: Fluxo típico — salários em múltiplos quantis

23.6 Capturando o resultado

```
let m = qreg(y ~ x, df, tau=0.75)
display m
let yhat = predict(m, "xb")
```

Capítulo 24

Regressão com Descontinuidade

24.1 O modelo

A Regressão com Descontinuidade (RDD) identifica efeitos causais quando o tratamento é atribuído mecanicamente com base em um limiar de uma variável contínua observável — a **variável de corrida** (*running variable*) x:

$$D_i = \mathbf{1}[x_i \geq c]$$

O estimador explora o fato de que indivíduos imediatamente abaixo e acima do limiar c são comparáveis em tudo, exceto no tratamento recebido. O efeito identificado é o **LATE no limiar** (*Local Average Treatment Effect at the cutoff*):

$$\tau_{\text{RDD}} = \lim_{x \downarrow c} E[y \mid x] - \lim_{x \uparrow c} E[y \mid x]$$

Nota

O RDD *Sharp* (acima) assume que D muda deterministicamente em c. O RDD *Fuzzy* generaliza para quando D aumenta *descontinuamente* em c, sem salto de 0 a 1 — requer um primeiro estágio e equivale a um IV local.

24.2 Identificação

A hipótese central é a **continuidade dos resultados potenciais** em c:

$$E[Y^{(0)} \mid x] \quad \text{e} \quad E[Y^{(1)} \mid x] \quad \text{são contínuas em } x = c$$

Isso garante que qualquer descontinuidade em $E[y \mid x]$ no ponto c é causada pelo salto em D, e não por outro fator que também muda em c.

A hipótese é violada se agentes *manipulam* x para ficarem acima ou abaixo do limiar — estudantes que revisam respostas para cruzar o corte de aprovação, por exemplo. O teste de densidades de McCrary (teste de continuidade da densidade de x em c) verifica se há aglomeração suspeita de observações de um lado do limiar.

24.3 O estimador

O estimador padrão é a **regressão local linear** separada em cada lado do limiar, dentro de uma janela (*bandwidth*) h:

$$\hat{\tau} = \hat{\mu}_+ - \hat{\mu}_-$$

onde $\hat{\mu}_+$ e $\hat{\mu}_-$ são os interceptos das regressões locais lineares no lado direito ($x \in [c, c+h)$) e esquerdo ($x \in (c-h, c)$), respectivamente, ponderadas pelo kernel triangular:

$$K\left(\frac{x-c}{h}\right) = \left(1 - \frac{|x-c|}{h}\right)\mathbf{1}[|x-c| \le h]$$

O *bandwidth* h controla o viés-variância: janelas largas reduzem variância mas aumentam viés (linearidade local pode ser má aproximação); janelas estreitas são mais locais mas têm menos observações. O Hayashi seleciona h automaticamente pelo critério MSE-ótimo (Imbens-Kalyanaraman) quando `bw` não é especificado.

24.4 Verificação por DGP

Recuperação exata (sem ruído)

O DGP usa a variável de corrida $x \sim \text{Uniforme}(-1, 1)$, limiar em $c = 0$ e efeito verdadeiro $\tau = 2{,}0$. A função de resultados é linear em x dos dois lados do limiar, sem ruído, de modo que a RDD deve recuperar τ exatamente.

$$y_i = 1{,}0 + 0{,}5\,x_i + 2{,}0\,D_i, \qquad D_i = \mathbf{1}[x_i \ge 0]$$

```
seed(42)
input df
id
1
end
for i in 1..=10 { df = append(df, df) }
```

```
df |> filter(_n <= 1000)
generate df x = runiform() * 2 - 1
generate df d = x >= 0
generate df y = 1.0 + 0.5*x + 2.0*d
rd(y ~ x, 0.0, df)
```

Listing 24.1: DGP RDD sem ruído — recuperação exata

```
Regressão Descontínua -   Sharp -   Local Linear (p=1)

Outcome: y                    Running var: x
Cutoff: 0.0000   Bandwidth: 0.0571   Kernel: Triangular
Obs total: 72   N esquerda: 37   N direita: 35

Efeito de Tratamento ^():
      2.0000   SE     0.0000   z 2.443e15   P>|z|   0.0000   ***
IC 95%: [2.0000, 2.0000]

*** p<0.01   ** p<0.05   * p<0.10
```

O estimador recupera $\hat{\tau} = 2{,}0000$ exatamente. O bandwidth de 0,0571 foi selecionado automaticamente — das 1000 observações, 72 estão dentro da janela em torno do cutoff.

Com ruído idiossincrático

Adicionando ε idiossincrático, o LATE permanece identificado:

```
generate df e = rnormal()
generate df y = 1.0 + 0.5*x + 2.0*d + e
rd(y ~ x, 0.0, df)
```

Listing 24.2: DGP RDD com ruído

```
Regressão Descontínua -   Sharp -   Local Linear (p=1)

Outcome: y                    Running var: x
Cutoff: 0.0000   Bandwidth: 0.0741   Kernel: Triangular
Obs total: 94   N esquerda: 43   N direita: 51
```

```
Efeito de Tratamento ^():
      2.0116    SE      0.1191    z    16.893    P>|z|    0.0000   ***
IC 95%: [1.7782, 2.2449]

*** p<0.01   ** p<0.05   * p<0.10
```

Com ruído, $\hat{\tau} = 2{,}0116$ — o desvio de 2,0 é variância amostral. O IC de 95% [1,78; 2,24] cobre o verdadeiro $\tau = 2{,}0$. O bandwidth foi 0,0741: com mais variância, o critério MSE-ótimo prefere janela mais estreita para controlar o viés.

24.5 Sintaxe

```
rd(y ~ x, cutoff, df)
rd(y ~ x, cutoff, df, bw=0.3)
rd(y ~ x, cutoff, df, bw=0.3, poly=2)
```

Parâmetro	Padrão	Descrição
`cutoff`	obrigatório	valor do limiar em x
`bw`	automático (IK)	largura da janela h
`poly`	1	grau do polinômio local
`kernel`	`triangular`	`uniform`, `epanechnikov`

```
load "escola.csv" as df

// pontuação centralizada no limiar 60
generate df xc = nota - 60

let m = rd(aprovado ~ xc, 0.0, df)
display m
```

Listing 24.3: Fluxo típico — nota de corte e bolsa

24.6 Capturando o resultado

```
let m = rd(y ~ x, 0.0, df)
display m
```

Nota

Em aplicações reais, valide o RDD com testes de falsificação: (1) estime o modelo usando *covariáveis pré-tratamento* como desfecho — não deve haver descontinuidade em c; (2) teste descontinuidades em *limiares placebo* ($c \pm \delta$) — não devem ser significativas; (3) verifique a densidade de x em c (McCrary) para detectar manipulação da variável de corrida.

24.7 Validação empírica

O DGP deste capítulo com variável de corrida uniforme e limiar em 0 é usado no caso de validação `rdd`$_{b}$*ookem*`validation/cases/`.$OcasoestimaoefeitodetratamentoporRDDem$`HAYAS`

Capítulo 25

Método Generalizado dos Momentos

25.1 O framework

O Método Generalizado dos Momentos (GMM) unifica os estimadores dos capítulos anteriores sob um único princípio: a identificação via **condições de momento** populacionais.

Um modelo GMM especifica L condições de momento:

$$E[g_i(\beta)] = 0, \qquad g_i(\beta) \in \mathbb{R}^L$$

OLS e IV são casos especiais:

Estimador	**Condição de momento**	**Situação**
OLS	$E[x_i\,\varepsilon_i] = 0$	$L = K$, identificado
IV/2SLS	$E[z_i\,\varepsilon_i] = 0$	$L = K$, identificado
GMM	$E[z_i\,\varepsilon_i] = 0$	$L > K$, sobreidentificado

Com $L = K$ momentos, resolve-se $g(\hat{\beta}) = 0$ exatamente - equivalente ao IV just-identified. Com $L > K$, o sistema é sobredeterminado e não existe β que zere todos os momentos simultaneamente; o GMM minimiza uma forma quadrática ponderada:

$$\hat{\beta}_{\text{GMM}} = \arg\min_{\beta} \; n\,\bar{g}(\beta)'\hat{W}^{-1}\bar{g}(\beta)$$

onde $\bar{g}(\beta) = \frac{1}{n}\sum_i g_i(\beta)$ e $\hat{W}$ é a estimativa consistente da variância assintótica de $\sqrt{n}\,\bar{g}(\beta)$.

25.2 O estimador dois passos

1. **Primeiro passo:** estima $\hat{\beta}_1$ com $W = I$ (peso uniforme).
2. **Matriz de peso eficiente:** calcula $\hat{W} = \frac{1}{n}\sum_i z_i z_i' \hat{\varepsilon}_i^2$ com resíduos do primeiro passo.
3. **Segundo passo:** reestima $\hat{\beta}_2$ com $\hat{W}^{-1}$ - o estimador GMM eficiente.

O estimador dois passos é assintoticamente eficiente na classe de estimadores GMM com as mesmas condições de momento.

25.3 Teste J de sobreidentificação

Quando $L > K$, é possível testar se os momentos excedentes são válidos. A estatística de Hansen (*J-test*):

$$J = n\,\bar{g}(\hat{\beta})'\hat{W}^{-1}\bar{g}(\hat{\beta}) \xrightarrow{d} \chi^2(L-K) \quad \text{sob } H_0 : E[g_i(\beta_0)] = 0$$

H_0 é a hipótese de que *todos* os L instrumentos são exógenos. Rejeitar H_0 indica que pelo menos um instrumento é correlacionado com ε - o modelo está mal especificado ou um instrumento é inválido.

Atenção

O teste J é necessário mas não suficiente: não rejeitar não prova que todos os instrumentos são válidos, apenas que as restrições de sobreidentificação são consistentes com os dados.

25.4 Verificação por DGP

OLS, IV e GMM lado a lado

O DGP repete a estrutura do capítulo 19: x é endógeno ($\text{Cov}(x, \varepsilon) \neq 0$), com dois instrumentos válidos z_1 e z_2. O ε é ortogonalizado contra z_1 in-sample - IV just-identified recupera exatamente; GMM com dois instrumentos é sobreidentificado ($L = 2, K = 1$).

```
seed(42)
input df
id
1
```

```
end
for i in 1..=10 { df = append(df, df) }
df |> filter(_n <= 1000)
generate df z1  = rnormal()
generate df z2  = rnormal()
generate df v   = rnormal()
let d1          = cov(df, z1, v) / variance(df, z1)
let d0          = mean(df, v) - d1 * mean(df, z1)
generate df eps = v - d0 - d1*z1
generate df x   = 0.8*z1 + 0.4*z2 + eps
generate df y   = 1.5 + 2.0*x + 0.7*eps
let m_ols = ols(y ~ x, df)
let m_iv  = iv(y ~ x, ~ z1, df)
let m_gmm = gmm(y ~ x, ~ z1 + z2, df)
display m_ols
display m_iv
display m_gmm
```

Listing 25.1: DGP GMM — OLS viesado, IV exato, GMM eficiente

```
Dep. Variable: y  R²=0.9798  N=1000  F=48418.53
const   1.2757***  (0.0077)    x   2.3736***  (0.0108)

Dep. Variable: y  Estimator: 2SLS  N=1000
const   1.5000***  (0.0170)    x   2.0000***  (0.0264)

Dep. Variable: y  GMM Two-Step  N=1000  J=2.5012  Prob(J)=0.1138  DF=1
x0   1.4875***  (0.0145)    x1   2.0206***  (0.0226)
```

OLS é viesado (endogeneidade): $\hat{\beta}_1 = 2{,}37 \neq 2{,}0$. IV com z_1 recupera exatamente $2{,}0000$ (ortogonalização in-sample). GMM com $z_1 + z_2$ converge a $2{,}0206$ - consistente, não exato, porque usa dois momentos para estimar um parâmetro. O teste J ($J = 2{,}50$, $p = 0{,}11$) não rejeita H_0: ambos os instrumentos são válidos.

Nota

O Hayashi exibe os coeficientes como x0 (constante) e x1 (slope) na saída do GMM. Isso é uma particularidade do display interno - os coeficientes correspondem exatamente à ordem das variáveis da fórmula.

J-test rejeita — instrumento inválido

Se z_2 for contaminado com ε, o teste J detecta a violação:

```
// z2b correlacionado com eps: instrumento inválido
generate df z2b = z2 + 0.5*eps
let m_bad = gmm(y ~ x, ~ z1 + z2b, df)
display m_bad
```

Listing 25.2: Instrumento inválido — J-test rejeita

```
Dep. Variable: y  GMM Two-Step  N=1000  J=152.9417  Prob(J)=0.0000  DF=1
x0   1.3738***  (0.0089)    x1   2.2125***  (0.0124)
```

$J = 152{,}94$ $(p = 0{,}000)$ rejeita H_0 com alta significância - e os coeficientes são viesados ($\hat{\beta}_1 = 2{,}21$). O teste J funcionou como alarme: antes de confiar nas estimativas GMM, verifique se J não rejeita.

25.5 Sintaxe

```
gmm(y ~ x1 + x2, ~ z1 + z2 + z3, df)
gmm(y ~ x1 + x2, ~ z1 + z2 + z3, df, cov=robust)
```

O primeiro lado direito (~ x1 + x2) lista os regressores; o segundo (~ z1 + z2 + z3) lista os instrumentos. Variáveis exógenas em x devem aparecer também como instrumentos de si mesmas.

```
load "dados.csv" as df

// edu é exógena (instrumento de si mesma)
// exp é endógena; z1 e z2 são instrumentos para exp
let m = gmm(sal ~ edu + exp, ~ edu + z1 + z2, df)
display m
```

Listing 25.3: Fluxo típico com variável exógena inclusa

25.6 Capturando o resultado

```
let m = gmm(y ~ x, ~ z1 + z2, df)
display m
let yhat = predict(m, "xb")
```

Parte III

Séries Temporais

Capítulo 26

AR, MA e ARMA

26.1 Processos estocásticos e estacionariedade

Um **processo estocástico** $\{y_t\}$ é estacionário (em covariância) quando:

$$E[y_t] = \mu, \qquad \mathrm{Var}[y_t] = \sigma^2, \qquad \mathrm{Cov}(y_t, y_{t-k}) = \gamma_k$$

para todo t - média, variância e autocovariâncias dependem apenas da defasagem k, não do tempo t.

O caso mais simples é o **ruído branco** $\varepsilon_t \sim \mathrm{WN}(0, \sigma^2)$: $E[\varepsilon_t] = 0$, $\mathrm{Var}[\varepsilon_t] = \sigma^2$ e $\mathrm{Cov}(\varepsilon_t, \varepsilon_s) = 0$ para $t \neq s$. Ruído branco é o bloco de construção de todos os modelos desta seção.

26.2 Processo AR(p)

O modelo autorregressivo de ordem p (**AR(p)**) expressa y_t como combinação linear de seus próprios valores passados mais um choque:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

Para o AR(1), a condição de estacionariedade é $|\phi_1| < 1$. Quando satisfeita, a média incondicional é $\mu = c/(1-\phi_1)$ e a autocorrelação decai geometricamente:

$$\rho_k = \phi_1^k, \qquad k = 0, 1, 2, \ldots$$

A **Função de Autocorrelação Parcial** (PACF) corta em zero após defasagem p - característica diagnóstica do AR.

26.3 Processo MA(q)

O modelo de médias móveis de ordem q (**MA(q)**) expressa y_t como combinação linear de choques presentes e passados:

$$y_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$$

O MA(q) é sempre estacionário. A **Função de Autocorrelação** (ACF) corta em zero após defasagem q - em contraste com o AR, cuja ACF decai geometricamente. A condição de **invertibilidade** ($|\theta_1| < 1$ para MA(1)) garante representação AR(∞) única.

Nota

Diagnóstico padrão: ACF decai lentamente e PACF corta → AR(p); ACF corta e PACF decai lentamente → MA(q); ambas decaem → ARMA.

26.4 Processo ARMA(p, q)

O ARMA(p,q) combina os dois mecanismos:

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

É estacionário quando as raízes do polinômio AR estão fora do círculo unitário e invertível quando as raízes do polinômio MA também estão.

26.5 Verificação por DGP

Como não existe gerador de defasagem em `generate`, os processos são construídos com um laço `while` que preenche y_t linha a linha, usando `mean(..., if t == k)` como acessor de linha k.

AR(1) com $\phi = 0{,}7$

```
seed(42)
input df
id
1
end
```

```
for i in 1..=9 { df = append(df, df) }
df |> filter(_n <= 500)
generate df e = rnormal()
generate df y = 0.0
generate df t = _n
let n = nrow(df)
let i = 2
while i <= n {
    let lag_y = mean(df, y, if = t == i - 1)
    let et    = mean(df, e, if = t == i)
    replace df y = 0.7 * lag_y + et if t == i
    i = i + 1
}
let m = arima(df, y, p=1, d=0, q=0)
display m
```

Listing 26.1: DGP AR(1) e estimação

```
================== ARIMA(1,0,0) via Hannan-Rissanen ==================
Observations:                 494
AIC:                    157.7072   BIC:                    166.1123
Parameters:
  intercept    0.5454   std=0.0557   z=9.786   p=0.000   [0.4362, 0.6547]
  ar.L1        0.6638   std=0.0335   z=19.814  p=0.000   [0.5981, 0.7295]
======================================================================
```

$\hat{\phi}_1 = 0{,}6638$ com verdadeiro $\phi_1 = 0{,}7$; o IC 95% $[0{,}60;\ 0{,}73]$ cobre o valor verdadeiro.

MA(1) com $\theta = 0{,}5$

```
generate df y = e
let i = 2
while i <= n {
    let lag_e = mean(df, e, if = t == i - 1)
    replace df y = y + 0.5 * lag_e if t == i
    i = i + 1
}
let m = arima(df, y, p=0, d=0, q=1)
display m
```

Listing 26.2: DGP MA(1) e estimação

```
================== ARIMA(0,0,1) via Hannan-Rissanen ==================
Observations:                 493
AIC:                     162.5430   BIC:                     170.9441
Parameters:
  intercept    0.7297    std=0.0128    z=57.014   p=0.000    [0.7046, 0.7548]
  ma.L1        0.5087    std=0.0456    z=11.163   p=0.000    [0.4194, 0.5980]
======================================================================
```

$\hat{\theta}_1 = 0{,}5087 \approx 0{,}5$; IC 95% $[0{,}42;\ 0{,}60]$ cobre $\theta_1 = 0{,}5$.

ARMA(1,1) com $\phi = 0{,}6$ e $\theta = 0{,}4$

```
generate df y = e
let i = 2
while i <= n {
    let lag_y = mean(df, y, if = t == i - 1)
    let lag_e = mean(df, e, if = t == i - 1)
    let et    = mean(df, e, if = t == i)
    replace df y = 0.6*lag_y + et + 0.4*lag_e if t == i
    i = i + 1
}
let m = arima(df, y, p=1, d=0, q=1)
display m
```

Listing 26.3: DGP ARMA(1,1) e estimação

```
================== ARIMA(1,0,1) via Hannan-Rissanen ==================
Observations:                 493
AIC:                     159.4217   BIC:                     172.0233
Parameters:
  intercept    0.8073    std=0.0660    z=12.227   p=0.000    [0.6779, 0.9367]
  ar.L1        0.5259    std=0.0380    z=13.831   p=0.000    [0.4514, 0.6004]
  ma.L1        0.4829    std=0.0591    z=8.167    p=0.000    [0.3670, 0.5987]
======================================================================
```

Ambos os parâmetros são recuperados com boa precisão: $\hat{\phi} = 0{,}53$ (verdadeiro $0{,}6$) e $\hat{\theta} = 0{,}48$ (verdadeiro $0{,}4$).

26.6 Seleção de ordem

O Hayashi reporta AIC e BIC automaticamente. O critério prático:

1. Estime modelos com ordens (p, q) em uma grade pequena (ex. $0 \leq p, q \leq 3$).
2. Selecione o modelo com menor AIC (preferência por ajuste) ou BIC (preferênci por parcimônia).
3. Verifique que os resíduos são ruído branco.

```
let m00 = arima(df, y, p=0, d=0, q=0)
let m10 = arima(df, y, p=1, d=0, q=0)
let m01 = arima(df, y, p=0, d=0, q=1)
let m11 = arima(df, y, p=1, d=0, q=1)
esttab(m00, m10, m01, m11)
```

Listing 26.4: Seleção de ordem por AIC

26.7 Sintaxe

```
arima(df, y, p=1, d=0, q=0)
arima(df, y, p=0, d=0, q=1)
arima(df, y, p=1, d=1, q=1)
```

Parâmetro	Padrão	Descrição
p	0	ordem autorregressiva
d	0	grau de diferenciação
q	0	ordem de médias móveis

```
load "ipca.csv" as df

let m = arima(df, inflacao, p=1, d=0, q=1)
display m
let yhat = predict(m, "xb")
```

Listing 26.5: Fluxo típico com dados reais

Nota

O laço `while` dos DGPs acima é $O(n^2)$ e serve apenas para ilustrar a estrutura do processo. Em uso real, os dados são carregados de arquivo com `load` e o laço não é necessário.

26.8 Validação empírica

Os DGPs deste capítulo são usados como casos de validação automática em validation/cases/: ar$_b ook$,

Capítulo 27

ARIMA e Raiz Unitária

27.1 Processos integrados

Um processo $\{y_t\}$ é **integrado de ordem** d - denotado $I(d)$ - quando precisa ser diferenciado d vezes para se tornar estacionário. O caso mais comum em economia é $I(1)$: a primeira diferença é estacionária, mas o nível não.

O caso mais simples de processo $I(1)$ é o **passeio aleatório** (*random walk*):

$$y_t = y_{t-1} + \varepsilon_t, \qquad \varepsilon_t \sim \mathrm{WN}(0, \sigma^2)$$

A variância cresce com t ($\mathrm{Var}[y_t] = t\sigma^2$), de modo que o processo não é estacionário. A primeira diferença $\Delta y_t = y_t - y_{t-1} = \varepsilon_t$ é ruído branco - portanto $I(0)$.

O passeio aleatório *com deriva* adiciona uma constante μ:

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

com tendência determinística $E[y_t] = y_0 + \mu t$ - modelo básico para preços de ativos, câmbio e muitas séries macroeconômicas.

27.2 Teste de Dickey-Fuller Aumentado

O teste ADF (*Augmented Dickey-Fuller*) testa formalmente a presença de raiz unitária. O modelo auxiliar estimado é:

$$\Delta y_t = \alpha + \delta y_{t-1} + \sum_{j=1}^{k} \gamma_j \Delta y_{t-j} + \varepsilon_t$$

com hipóteses:

$$H_0 : \delta = 0 \quad \text{(raiz unitária)} \qquad H_1 : \delta < 0 \quad \text{(estacionário)}$$

A estatística t de $\hat{\delta}$ não segue a distribuição t padrão sob H_0 - usa-se a distribuição de Dickey-Fuller com valores críticos tabelados.

Atenção

Os valores críticos do ADF são **negativos**: rejeita-se H_0 quando a estatística é suficientemente *negativa*. Um valor positivo (ou próximo de zero) não rejeita H_0 - evidência de raiz unitária.

27.3 O modelo ARIMA

O ARIMA(p,d,q) generaliza o ARMA para séries não estacionárias, aplicando d diferenciações antes da estimação:

$$\underbrace{\phi(L)(1-L)^d y_t}_{\text{AR em diferenças}} = c + \underbrace{\theta(L)\varepsilon_t}_{\text{MA}}$$

onde L é o operador de defasagem ($Ly_t = y_{t-1}$) e $(1-L)^d$ é o operador de diferença de ordem d.

Modelo	Notação	Processo
Passeio aleatório	ARIMA(0,1,0)	$\Delta y_t = c + \varepsilon_t$
AR(1) em diferenças	ARIMA(1,1,0)	$\Delta y_t = c + \phi\,\Delta y_{t-1} + \varepsilon_t$
MA(1) em diferenças	ARIMA(0,1,1)	$\Delta y_t = c + \varepsilon_t + \theta\,\varepsilon_{t-1}$

27.4 Verificação por DGP

Random walk: ADF não rejeita, AR(1): ADF rejeita

O DGP contrasta os dois casos: passeio aleatório (I(1)) gerado por cumsum (e) versus AR(1) estacionário ($\phi = 0{,}7$) gerado por laço.

```
seed(42)
input df
id
1
end
```

```
for i in 1..=9 { df = append(df, df) }
df |> filter(_n <= 500)
generate df e  = rnormal()
generate df rw = cumsum(e)
generate df y  = 0.0
generate df t  = _n
let n = nrow(df)
let i = 2
while i <= n {
    let ly = mean(df, y, if = t == i - 1)
    let et = mean(df, e, if = t == i)
    replace df y = 0.7 * ly + et if t == i
    i = i + 1
}
adf(df, rw)
adf(df, y)
```

Listing 27.1: ADF em passeio aleatório vs. AR(1)

```
=============== Augmented Dickey-Fuller Test ===============
  Variable: rw   Lags used: 7
  H: series has a unit root (non-stationary)
  Test statistic:      -0.0980
  Critical values:  1%=-3.430  5%=-2.860  10%=-2.570
  Conclusion: Não rejeita  H - raiz unitária presente

=============== Augmented Dickey-Fuller Test ===============
  Variable: y   Lags used: 7
  H: series has a unit root (non-stationary)
  Test statistic:      -6.4616
  Critical values:  1%=-3.430  5%=-2.860  10%=-2.570
  Conclusion: REJEITA  H - estacionária
```

O passeio aleatório tem estatística $-0{,}10$ - acima dos valores críticos negativos, não rejeita H_0. O AR(1) com $\phi = 0{,}7$ tem estatística $-6{,}46$ - muito abaixo do valor crítico de 1% ($-3{,}43$), rejeita H_0 com alta confiança.

Estimação ARIMA(0,1,0) e seleção de ordem

Com a raiz unitária identificada, estima-se com $d = 1$. Três ordens concorrentes para o passeio aleatório verdadeiro:

```
let m010 = arima(df, rw, p=0, d=1, q=0)
let m110 = arima(df, rw, p=1, d=1, q=0)
let m011 = arima(df, rw, p=0, d=1, q=1)
display m010
display m110
display m011
```

Listing 27.2: Seleção de ordem para série I(1)

```
-- ARIMA(0,1,0): AIC=156.77  BIC=160.97
   intercept   0.4863  std=0.0128  z=38.14  p=0.000

-- ARIMA(1,1,0): AIC=158.70  BIC=167.10
   intercept   0.4805  std=0.0253  z=18.96  p=0.000
   ar.L1       0.0120  std=0.0450  z=0.266  p=0.791  [insig.]

-- ARIMA(0,1,1): AIC=157.80  BIC=166.19
   intercept   0.4855  std=0.0128  z=38.07  p=0.000
   ma.L1       0.0058  std=0.0453  z=0.128  p=0.898  [insig.]
```

O modelo verdadeiro ARIMA(0,1,0) tem o menor AIC e BIC. Os termos AR e MA adicionais são insignificantes ($p > 0{,}7$) - o BIC, que penaliza mais fortemente a complexidade, confirma a parcimônia do modelo simples.

27.5 Fluxo de trabalho

O protocolo padrão para séries temporais univariadas:

1. **Inspecionar** o nível da série - tendência visível sugere não-estacionariedade.
2. **Testar** com ADF - não rejeitar H_0 indica $d \geq 1$.
3. **Diferenciar** uma vez e re-testar com ADF na série diferenciada - confirma $d = 1$.
4. **Selecionar** (p, q) pela grade AIC/BIC sobre a série diferenciada.
5. **Validar** resíduos: devem ser ruído branco.

27.6 Sintaxe

```
adf(df, y)
adf(df, y, lags=4)
arima(df, y, p=0, d=1, q=0)
arima(df, y, p=1, d=1, q=1)
```

```
load "cambio.csv" as df

adf(df, taxa)

// se não rejeitar H0: estimar com d=1
let m = arima(df, taxa, p=1, d=1, q=0)
display m
let yhat = predict(m, "xb")
```

Listing 27.3: Fluxo típico com dados reais

27.7 Validação empírica

O DGP de passeio aleatório deste capítulo é usado no caso de validação arima_book_em validation/cases/. O caso estima um $ARIMA(1,1,0)$ em HAYASHI e o compara com a si

Capítulo 28

VAR e Causalidade de Granger

28.1 O modelo VAR

O modelo Autorregressivo Vetorial (**VAR(p)**) generaliza o AR para sistemas multivariados. Com K variáveis e p defasagens:

$$\mathbf{y}_t = \mathbf{c} + \mathbf{A}_1 \mathbf{y}_{t-1} + \cdots + \mathbf{A}_p \mathbf{y}_{t-p} + \boldsymbol{\varepsilon}_t$$

onde $\mathbf{y}_t = (y_{1t}, \ldots, y_{Kt})'$ é o vetor de variáveis, $\mathbf{A}_j$ são matrizes $K \times K$ de coeficientes e $\boldsymbol{\varepsilon}_t \sim (0, \Sigma_u)$ é o vetor de choques não correlacionados no tempo mas potencialmente correlacionados entre si.

O VAR é estimado equação por equação via OLS - cada equação é um AR(p) nos K defasados. A condição de estacionariedade exige que todas as raízes características de $\mathbf{A}_1 + \cdots + \mathbf{A}_p$ estejam dentro do círculo unitário.

28.2 Causalidade de Granger

x **causa Granger** y se as defasagens de x têm poder preditivo para y além do que as defasagens de y sozinhas fornecem. Formalmente, testa-se a hipótese:

$$H_0 : A_{12}(1) = A_{12}(2) = \cdots = A_{12}(p) = 0$$

na equação de y_1, onde $A_{12}(j)$ é o coeficiente de $y_{2,t-j}$.

O teste é um teste F padrão na equação auxiliar:

$$F = \frac{(RSS_r - RSS_u)/p}{RSS_u/(T - 2p - 1)} \sim F(p,\, T - 2p - 1)$$

Nota

Causalidade de Granger é *previsibilidade*, não causalidade estrutural. x que antecede y e correlaciona com ela passa no teste, mesmo sem relação causal direta.

28.3 Função de Resposta ao Impulso

A **IRF** (*Impulse Response Function*) rastreia o efeito de um choque unitário em y_k sobre todas as variáveis ao longo do tempo.

A representação MA(∞) do VAR é $\mathbf{y}_t = \sum_{h=0}^{\infty} \mathbf{\Phi}_h \boldsymbol{\varepsilon}_{t-h}$, com $\mathbf{\Phi}_0 = I$ e $\mathbf{\Phi}_h = \mathbf{A}_1\mathbf{\Phi}_{h-1} + \cdots + \mathbf{A}_p\mathbf{\Phi}_{h-p}$. A IRF ortogonalizada usa a decomposição de Cholesky de Σ_u para isolar choques não correlacionados - a **ordenação de Cholesky** (ordem das variáveis) importa.

28.4 Verificação por DGP

O DGP usa um VAR(1) bivariado com estrutura de causalidade assimétrica: y_2 causa y_1 (coeficiente $0{,}2$), mas y_1 causa y_2 fracamente (coeficiente $0{,}1$):

$$\begin{pmatrix} y_{1t} \\ y_{2t} \end{pmatrix} = \begin{pmatrix} 0{,}5 & 0{,}2 \\ 0{,}1 & 0{,}6 \end{pmatrix} \begin{pmatrix} y_{1,t-1} \\ y_{2,t-1} \end{pmatrix} + \begin{pmatrix} \varepsilon_{1t} \\ \varepsilon_{2t} \end{pmatrix}$$

```
seed(42)
input df
id
1
end
for i in 1..=9 { df = append(df, df) }
df |> filter(_n <= 300)
generate df e1 = rnormal()
generate df e2 = rnormal()
generate df y1 = 0.0
generate df y2 = 0.0
generate df t  = _n
let n = nrow(df)
let i = 2
while i <= n {
    let l1 = mean(df, y1, if = t == i - 1)
    let l2 = mean(df, y2, if = t == i - 1)
```

```
    let a  = mean(df, e1, if t == i)
    let b  = mean(df, e2, if t == i)
    replace df y1 = 0.5*l1 + 0.2*l2 + a if t == i
    replace df y2 = 0.1*l1 + 0.6*l2 + b if t == i
    i = i + 1
}
let m = var(df, y1, y2, lags=1)
display m
granger(df, y1, y2, lags=1)
granger(df, y2, y1, lags=1)
irf(m, steps=6)
```

Listing 28.1: DGP VAR(1) — Granger e IRF

Ajuste do modelo

```
VAR(1)  N=299  AIC=-5.0269  BIC=-4.9526

Sigma_u:
  [ 0.0819  0.0046 ]
  [ 0.0046  0.0772 ]
```

A matriz de covariância residual $\hat{\Sigma}_u$ é quase diagonal - os choques ε_1 e ε_2 são praticamente ortogonais, como especificado no DGP.

Causalidade de Granger

```
================== Granger Causality Test ==================
  H: y2 não causa Granger y1   (lags=1)
  F(1, 296) = 9.4437   p = 0.0023
  Conclusion: REJEITA  H - y2 causa Granger y1

================== Granger Causality Test ==================
  H: y1 não causa Granger y2   (lags=1)
  F(1, 296) = 4.5768   p = 0.0332
  Conclusion: REJEITA  H - y1 causa Granger y2
```

Ambas as direções rejeitam H_0: $y_2 \to y_1$ com $p = 0{,}002$ e $y_1 \to y_2$ com $p = 0{,}033$. O DGP impõe $\phi_{12} = 0{,}2$ e $\phi_{21} = 0{,}1$ - a causalidade bidirecional é detectada, embora $y_2 \to y_1$ seja mais forte.

Resposta ao impulso

```
IRF - VAR(1) - 6 passos

  Impulso: y1

       h           y1           y2

       1       0.2862       0.0161
       2       0.1430       0.0382
       3       0.0758       0.0364
       4       0.0426       0.0286
       5       0.0251       0.0207
       6       0.0154       0.0145


  Impulso: y2

       h           y1           y2

       1       0.0000       0.2774
       2       0.0405       0.1595
       3       0.0432       0.0958
       4       0.0352       0.0595
       5       0.0260       0.0378
       6       0.0183       0.0243
```

Um choque em y_1 tem efeito rápido sobre y_1 (decai de $0{,}29$ para $0{,}015$ em 6 períodos) e efeito pequeno sobre y_2. Um choque em y_2 transmite-se progressivamente a y_1 - atinge $0{,}043$ no período 3, depois decai - capturando o canal $\phi_{12} = 0{,}2$ do DGP.

28.5 Seleção de defasagens

```
let m1 = var(df, y1, y2, lags=1)
let m2 = var(df, y1, y2, lags=2)
let m3 = var(df, y1, y2, lags=3)
esttab(m1, m2, m3)
```

Listing 28.2: Comparação por número de defasagens

Seleciona-se a defasagem com menor BIC (parcimônia) ou AIC (ajuste). Com o DGP VAR(1), espera-se que $p = 1$ minimize ambos.

28.6 Sintaxe

```
var(df, y1, y2, lags=1)
var(df, y1, y2, y3, lags=2)
granger(df, y, x, lags=1)
irf(m, steps=10)
```

```
load "macro.csv" as df

adf(df, pib)
adf(df, ipca)

let m = var(df, pib, ipca, lags=2)
granger(df, pib, ipca, lags=2)
granger(df, ipca, pib, lags=2)
irf(m, steps=12)
```

Listing 28.3: Fluxo típico — PIB e inflação

Nota

O VAR exige séries *estacionárias*. Se as variáveis forem I(1) e cointegradas, o modelo adequado é o VECM (cap. 29). Se forem I(1) mas não cointegradas, diferencie antes de estimar.

28.7 Validação empírica

O DGP VAR(1) deste capítulo corresponde ao caso de validação var_lookemvalidation/

Capítulo 29

Cointegração e VECM

29.1 Relação de longo prazo entre séries I(1)

Duas séries $I(1)$ são **cointegradas** se existe uma combinação linear entre elas que é $I(0)$. Formalmente, y_t e x_t são cointegradas se existe β tal que:

$$u_t = y_t - \beta x_t \sim I(0)$$

A intuição é a de uma *correia*: séries que individualmente seguem passeios aleatórios podem ser atraídas por uma força de equilíbrio de longo prazo. Preços de ativos no mesmo mercado, taxa de câmbio e paridade de poder de compra, juros de curto e longo prazo são exemplos clássicos.

Séries $I(1)$ não cointegradas divergem indefinidamente - a regressão em nível produz **regressão espúria**: R^2 alto e t-estatísticas significativas sem relação causal real.

29.2 Teste de Engle-Granger

O procedimento de Engle-Granger (1987) testa cointegração em dois passos:

1. Estima a **relação de longo prazo** por OLS: $\hat{u}_t = y_t - \hat{\beta} x_t$
2. Aplica o **teste ADF** nos resíduos $\hat{u}_t$ - sob H_0 os resíduos têm raiz unitária (sem cointegração).

Os valores críticos do teste são mais negativos do que os do ADF padrão, pois $\hat{u}_t$ é resíduo estimado, não série diretamente observada.

29.3 Modelo de Correção de Erros (VECM)

Se y_t e x_t são cointegradas, o VECM (*Vector Error Correction Model*) decompõe a dinâmica em componente de curto prazo e ajuste ao equilíbrio de longo prazo:

$$\Delta \mathbf{y}_t = \boldsymbol{\alpha} \underbrace{(\boldsymbol{\beta}' \mathbf{y}_{t-1})}_{\text{desvio do equilíbrio}} + \sum_{j=1}^{p-1} \boldsymbol{\Gamma}_j \Delta \mathbf{y}_{t-j} + \boldsymbol{\varepsilon}_t$$

- $\boldsymbol{\beta}$ - **vetor cointegrante**: define a relação de equilíbrio de longo prazo $\boldsymbol{\beta}' \mathbf{y}_t = 0$.
- $\boldsymbol{\alpha}$ - **coeficientes de ajuste**: velocidade com que cada variável corrige desvios do equilíbrio. Sinal negativo em α_y significa que y cai quando está acima do equilíbrio.
- $\boldsymbol{\Gamma}_j$ - dinâmica de curto prazo.

O VECM é estimado por ML via procedimento de Johansen, que testa o número de vetores cointegrantes (*rank*) via estatística de traço (*trace*) ou máximo autovalor.

29.4 Verificação por DGP

Caso 1 — cointegrado: $y_t = 2x_t + \varepsilon_t$

x_t é um passeio aleatório ($I(1)$); $\varepsilon_t \sim I(0)$. Logo $y_t - 2x_t = \varepsilon_t \sim I(0)$: as séries são cointegradas com vetor $[1, -2]$.

```
seed(42)
input df
id
1
end
for i in 1..=9 { df = append(df, df) }
df |> filter(_n <= 300)
generate df e1 = rnormal()
generate df e2 = rnormal()
generate df x  = cumsum(e1)
generate df y  = 2.0*x + e2
coint(df, y, x)
```

Listing 29.1: DGP cointegrado e teste de Engle-Granger

```
============= Engle-Granger Cointegration Test =============
  Variables: y, x
  H: sem cointegração
  ADF statistic:    -6.4808
  Critical values:  1%=-3.900  5%=-3.340  10%=-3.040
  Vetor cointegrante: [0.4778, 2.0002]
  Conclusion: REJEITA  H - séries cointegradas
```

A estatística $-6{,}48$ está muito abaixo do valor crítico de 1% ($-3{,}90$). O vetor cointegrante estimado $[0{,}48;\ 2{,}00]$ recupera $\beta = 2{,}0$ com precisão.

Caso 2 — não cointegrado: dois passeios aleatórios independentes

```
generate df y = cumsum(e2)
coint(df, y, x)
```

Listing 29.2: Dois random walks — sem cointegração

```
============= Engle-Granger Cointegration Test =============
  Variables: y, x
  H: sem cointegração
  ADF statistic:    -3.1058
  Critical values:  1%=-3.900  5%=-3.340  10%=-3.040
  Conclusion: Não rejeita  H - sem cointegração
```

VECM — dinâmica de ajuste

Com cointegração confirmada, estima-se o VECM:

```
generate df y = 2.0*x + e2
let m = vecm(df, y, x, lags=1)
display m
```

Listing 29.3: VECM com rank 1

```
======================== VECM (Johansen ML) - Rank 1 ========================
No. Variables:                2
Observations:               299
```

```
------------------------ Cointegration Vectors (Beta) ------------------------
[ 3.6175, -7.2357 ] →   normalizado: [ 1, -2.000 ]

---------------------- Adjustment Coefficients (Alpha) ----------------------
  y:  -0.3133   (corrige desvios do equilíbrio)
  x:  -0.0179   (quase exógena - ajuste mínimo)

----------------------- Johansen Eigenvalues (Lambda) -----------------------
     = 0.5017        = 0.0004
================================================================================
```

O vetor cointegrante normalizado $[1, -2{,}00]$ recupera o verdadeiro $\beta = 2{,}0$. O coeficiente de ajuste $\hat{\alpha}_y = -0{,}313$: quando $y_{t-1} > 2x_{t-1}$ (acima do equilíbrio), Δy_t é negativo - y se corrige para baixo. x quase não ajusta ($\hat{\alpha}_x \approx -0{,}02$), pois é a variável exógena que define o equilíbrio.

29.5 Sintaxe

```
coint(df, y, x)
coint(df, y, x, lags=4)
vecm(df, y, x, lags=1)
vecm(df, y1, y2, y3, lags=2)
```

```
load "cambio.csv" as df

adf(df, spot)
adf(df, forward)

coint(df, spot, forward)

// se cointegradas:
let m = vecm(df, spot, forward, lags=2)
display m
```

Listing 29.4: Fluxo típico — taxa de câmbio e paridade

Nota

O VECM pressupõe que todas as variáveis são $I(1)$ e cointegradas. Verifique com ADF antes de estimar. Se as séries forem $I(1)$ mas não cointegradas, use o VAR em primeiras diferenças (cap. 28).

29.6 Validação empírica

O DGP cointegrado deste capítulo é usado no caso de validação coint$_{b}ookemvalidati$

Capítulo 30

GARCH e Volatilidade Condicional

30.1 Heterocedasticidade condicional

Séries financeiras exibem um fenômeno recorrente: períodos de alta volatilidade tendem a ser seguidos de mais volatilidade, e períodos calmos persistem. Esse *agrupamento de volatilidade* (*volatility clustering*) viola a hipótese de variância condicional constante dos modelos ARMA - a variância de ε_t dado o passado varia ao longo do tempo.

O modelo GARCH(*p,q*) (Engle 1982; Bollerslev 1986) captura essa dinâmica explicitamente. A especificação GARCH(1,1) é:

$$\varepsilon_t \mid \mathcal{F}_{t-1} \sim (0,\, h_t) \qquad h_t = \omega + \alpha_1\, \varepsilon_{t-1}^2 + \beta_1\, h_{t-1}$$

onde h_t é a *variância condicional*, α_1 captura o efeito ARCH (reação ao choque passado) e β_1 captura a persistência (memória da volatilidade).

Restrição	Condição	Significado
Variância positiva	$\omega > 0,\ \alpha_1 \geq 0,\ \beta_1 \geq 0$	$h_t > 0$ garantido
Estacionariedade	$\alpha_1 + \beta_1 < 1$	variância não explode
Variância incondicional	$\sigma^2 = \omega/(1-\alpha_1-\beta_1)$	nível de longo prazo

O caso $\alpha_1+\beta_1 = 1$ é chamado IGARCH (*integrated*): a volatilidade tem raiz unitária e choques têm efeito permanente.

30.2 Teste ARCH-LM

Antes de estimar um modelo de volatilidade, testa-se se existem efeitos ARCH. O teste de Engle (1982) regride ε_t^2 em seus próprios lags e avalia

a significância conjunta:

$$\varepsilon_t^2 = a_0 + a_1\,\varepsilon_{t-1}^2 + \cdots + a_m\,\varepsilon_{t-m}^2 + u_t$$

$$H_0\colon a_1 = \cdots = a_m = 0 \quad \text{(sem efeitos ARCH)}$$

A estatística é $LM = N \cdot R^2_{\text{aux}} \xrightarrow{d} \chi^2(m)$ sob H_0.

30.3 Verificação por DGP

O DGP gera um processo ARCH(1) verdadeiro ($\beta = 0$):

$$h_t = 0{,}3 + 0{,}5\,\varepsilon_{t-1}^2 \qquad \varepsilon_t = \sqrt{h_t}\,z_t \qquad z_t \sim (0,1)$$

A inovação é gerada diretamente como sorteio normal padrão: $z = \text{rnormal}()$, portanto $E[z] = 0$ e $E[z^2] = 1$.

```
seed(42)
input df
id
1
end
for i in 1..=9 { df = append(df, df) }
df |> filter(_n <= 500)
generate df z  = rnormal()                  // z: E[z]=0, E[z²]=1
generate df e  = z
generate df t  = _n
let n = nrow(df)
let i = 2
while i <= n {
    let le = mean(df, e, if = t == i - 1)
    let lz = mean(df, z, if = t == i)
    replace df e = sqrt(0.3 + 0.5 * le * le) * lz if t == i
    i = i + 1
}

archtest(df, e, lags=5)      // deve rejeitar H
let m = garch(df, e, p=1, q=1)
display m
```

```
23 archtest(m, lags=5)          // deve não rejeitar (GARCH
       captou o ARCH)
```

Listing 30.1: DGP ARCH(1) e estimação GARCH

Teste ARCH-LM — série bruta

A série bruta deve rejeitar a hipótese nula de ausência de ARCH, confirmando variância condicional variável no processo simulado.

Estimação GARCH(1,1)

O estimador MLE recupera o DGP verdadeiro ($\omega = 0{,}3$, $\alpha = 0{,}5$, $\beta = 0$) até a variação de amostra finita. Um $\hat{\beta}$ próximo de zero identifica que o processo é ARCH puro - sem componente GARCH de persistência.[1]

ARCH-LM nos resíduos padronizados

Após ajustar o GARCH, os resíduos padronizados $\hat{z}_t = \varepsilon_t / \sqrt{\hat{h}_t}$ devem ser ruído branco em z_t^2:

Depois do ajuste, o teste ARCH-LM nos resíduos padronizados não deve rejeitar se a especificação de volatilidade captou a heterocedasticidade condicional.

30.4 Variantes: EGARCH e GJR-GARCH

O GARCH simétrico trata choques positivos e negativos igualmente. Em ativos financeiros, choques negativos tendem a aumentar mais a volatilidade do que positivos de mesma magnitude (*efeito de alavancagem*).

O **EGARCH** (Nelson 1991) modela $\ln h_t$:

$$\ln h_t = \omega + \beta_1 \ln h_{t-1} + \alpha_1 \left| \frac{\varepsilon_{t-1}}{\sqrt{h_{t-1}}} \right| + \gamma_1 \frac{\varepsilon_{t-1}}{\sqrt{h_{t-1}}}$$

O parâmetro $\gamma_1 < 0$ captura a assimetria (efeito de alavancagem).

O **GJR-GARCH** (Glosten-Jagannathan-Runkle 1993) adiciona um termo indicador:

$$h_t = \omega + \alpha_1 \varepsilon_{t-1}^2 + \gamma_1 \varepsilon_{t-1}^2 \mathbf{1}[\varepsilon_{t-1} < 0] + \beta_1 h_{t-1}$$

[1] Com $N = 500$, estimativas em amostras finitas não precisam coincidir exatamente com os parâmetros do DGP; com N maior, a convergência melhora.

O coeficiente $\gamma_1 > 0$ implica que choques negativos elevam mais a volatilidade.

30.5 Sintaxe

```
garch(df, y, p=1, q=1)
garch(df, y, p=1, q=1, dist=t)    // erros Student-t
egarch(df, y, p=1, q=1)
gjrgarch(df, y, p=1, q=1)
archtest(df, y, lags=5)            // ARCH-LM na série bruta
archtest(modelo, lags=5)           // ARCH-LM nos resíduos
   padronizados
```

Parâmetro	Padrão	Descrição
p	1	ordem ARCH (α)
q	1	ordem GARCH (β)
dist	normal	distribuição: normal ou t
lags	5	lags para ARCH-LM

```
load "retornos.csv" as df

// 1. Verificar agrupamento de volatilidade
archtest(df, ret, lags=10)

// 2. Estimar modelo de volatilidade
let m_garch  = garch(df, ret, p=1, q=1)
let m_gjr    = gjrgarch(df, ret, p=1, q=1)

// 3. Verificar diagnóstico
archtest(m_garch, lags=10)

// 4. Variância condicional estimada
display m_garch
```

Listing 30.2: Fluxo típico — retornos de ativos

Nota

O Hayashi estima GARCH por MLE assumindo inovações gaussianas (dist=normal) ou Student-t (dist=t). Para séries com caudas muito pesadas (criptomoedas, câmbio em crises), a distribuição t geralmente melhora o ajuste. Comparar AIC/BIC entre especificações.

30.6 Validação empírica

O DGP ARCH(1) deste capítulo corresponde ao caso de validação garch$_b$*ookem*validati

Parte IV

Inferência Causal

Capítulo 31

Propensity Score Matching

31.1 Seleção em observáveis

Em muitos estudos observacionais, a atribuição ao tratamento não é aleatória - indivíduos se selecionam ao tratamento com base em características observáveis X. Se X afeta também o resultado Y, a comparação direta entre tratados e controles produz estimativas viesadas.

O **Propensity Score Matching** (PSM) explora o teorema de Rosenbaum e Rubin (1983): se a atribuição ao tratamento é independente dos resultados potenciais dado X (*ignorabilidade*), então a mesma independência vale dado o propensity score $p(X) = P(D = 1 \mid X)$:

$$(Y^0, Y^1) \perp D \mid X \quad \Longrightarrow \quad (Y^0, Y^1) \perp D \mid p(X)$$

Isso reduz o problema de dimensão alta (condicionar em X) para um problema unidimensional (condicionar em $\hat{p}$). O PSM:

1. Estima $\hat{p}(X_i)$ via logit (score de propensão).

2. Emparelha cada tratado com o(s) controle(s) mais próximo(s) em $\hat{p}$.

3. Computa o ATT como diferença de médias no grupo emparelhado.

A condição de suporte comum (*overlap*): $0 < p(X) < 1$ para todo X é essencial - unidades com probabilidade 0 ou 1 de tratamento não têm contrafactual no outro grupo.

31.2 Verificação por DGP

O DGP cria confusão via x_1: quem tem x_1 alto tem maior probabilidade de tratamento e também maior resultado em média. O ATT verdadeiro é $\tau = 2$.

$$P(D = 1 \mid x_1) = 0{,}3 + 0{,}4\,x_1 \in [0{,}30,\ 0{,}70] \qquad y_i = 1 + 2\,d_i + 3\,x_{1i} + \varepsilon_i$$

```
seed(42)
input df
id
1
end
for i in 1..=10 { df = append(df, df) }
df |> filter(_n <= 1000)
generate df x1 = runiform()                    // x1 ~ U(0,1)
generate df u  = runiform()                    // ruído de
    atribuição
generate df d  = (0.3 + 0.4*x1) > u           // P(D=1|x1)
    [0.30, 0.70]
generate df e  = rnormal()
generate df y  = 1.0 + 2.0*d + 3.0*x1 + e    // ATT verdadeiro
     = 2
```

Listing 31.1: DGP com seleção em observáveis

OLS ingênuo vs. OLS com controles

```
let m_ols = ols(y ~ d, df)           // sem controle: viesado
let m_adj = ols(y ~ d + x1, df)     // com controle: não
    viesado
display m_ols
display m_adj
```

Listing 31.2: Comparação OLS sem e com controle de x1

```
 OLS sem controle
const   2.9135***   (0.0398)     d   2.2215***   (0.0562)
N=1000   R²=0.6105
```

```
 OLS com x1
const   1.5289***   (0.0193)     d   1.9762***   (0.0180)
x1      3.0053***   (0.0317)
N=1000   R²=0.9610
```

O OLS sem controle superestima τ em $+0{,}22$ (2.22 vs. 2.0) - viés de variável omitida. Controlando x_1 o OLS recupera $\hat{\tau} = 1{,}98$, indistinguível de 2.

PSM sem caliper

```
let m_psm = psm(y ~ d + x1, df, k=1, boot=200)
display m_psm
```

Listing 31.3: PSM 1:1 sem caliper

Sem caliper, o matching aceita pares distantes em $\hat{p}$. O balanceamento pode melhorar pouco e a estimativa pode permanecer viesada se tratados forem pareados com controles ruins.

PSM com caliper

O caliper limita a distância máxima admitida no matching. A regra prática de Cochran e Rubin (1973) é caliper $= 0{,}2 \times \sigma_{\hat{p}}$, o que corresponde aproximadamente a $0{,}05$ para scores uniformes.

```
let m_cal = psm(y ~ d + x1, df, k=1, caliper=0.05, boot=200)
display m_cal
```

Listing 31.4: PSM com caliper $= 0.05$

Com caliper $= 0{,}05$, a estimativa deve se aproximar do ATT verdadeiro ao custo de descartar tratados sem par aceitável. O caliper também deve melhorar o balanceamento ao rejeitar matches distantes no propensity score.[1]

31.3 Sintaxe

[1] O aviso `SMD > 0.10` persiste porque a regra convencional exige SMD $\leq 0{,}10$ (Austin 2009). Neste DGP a confusão por x_1 é intensa ($\beta_{x_1} = 3$) e $N = 1000$; com mais observações o caliper poderia ser estreitado para melhorar ainda mais o balanço.

```
psm(outcome ~ treatment + cov1 + cov2, df)
psm(outcome ~ treatment + cov1 + cov2, df, k=1, caliper=0.05,
    replace=false, boot=200)
```

A fórmula: `outcome ~ treatment + cov1 + cov2 + ....` O primeiro regressor após `~` é a variável de tratamento; os demais são as covariáveis usadas no modelo de propensão.

Opção	Padrão	Descrição
k	1	número de controles por tratado
caliper	nenhum	distância máxima no score de propensão
replace	false	matching com reposição
boot	200	réplicas bootstrap para o SE

```
load "beneficiarios.csv" as df

// Covariáveis de seleção
let m = psm(renda ~ tratado + idade + educ + regiao, df,
            k=1, caliper=0.05, boot=500)
display m
```

Listing 31.5: Fluxo típico — avaliação de programa

Nota

O PSM é consistente sob *ignorabilidade* (seleção em observáveis): $\{Y^0, Y^1\} \perp D \mid X$. Variáveis omitidas que afetam simultaneamente D e Y invalidam a identificação - o mesmo problema que torna IV necessário em cap. 19. Ao contrário do DiD (cap. 21), o PSM não elimina confusores não observados constantes no tempo.

31.4 Validação empírica

O dataset jtrain3 do Wooldridge é usado no caso de validação psm$jtrain3em$validation/ca

Capítulo 32

Controle Sintético

32.1 O problema do caso único

O DiD (cap. 21) e o PSM (cap. 31) dependem de múltiplas unidades tratadas e controles comparáveis. Em muitas aplicações de política, porém, apenas uma unidade recebe o tratamento - um país que adota um programa econômico, um estado que muda sua legislação - e as unidades potencialmente comparáveis são poucas e heterogêneas.

O **Controle Sintético** (Abadie, Diamond e Hainmueller, ADH 2010) constrói um grupo de controle artificial - o *controle sintético* - como uma combinação convexa de unidades doadoras que replica a trajetória da unidade tratada *antes* do tratamento. A ideia central:

$$\hat{Y}_{1t}^{(0)} = \sum_{j=2}^{J+1} w_j^* \, Y_{jt} \qquad w_j^* \geq 0, \quad \sum_{j=2}^{J+1} w_j^* = 1$$

Os pesos $W^* = (w_2^*, \ldots, w_{J+1}^*)$ minimizam o erro quadrático médio de predição (*RMSPE*) no período pré-tratamento:

$$W^* = \arg\min_{W \in \Delta^J} \sum_{t=1}^{T_0 - 1} \left(Y_{1t} - \sum_{j \geq 2} w_j Y_{jt} \right)^2$$

onde Δ^J é o simplexo unitário ($w_j \geq 0$, $\sum w_j = 1$). O efeito do tratamento no período $t \geq T_0$ é:

$$\hat{\tau}_t = Y_{1t} - \hat{Y}_{1t}^{(0)}$$

A validade do método repousa sobre a qualidade do ajuste pré-tratamento: se o controle sintético acompanha a unidade tratada no período pré-tratamento,

a hipótese de que teria continuado a fazê-lo na ausência do tratamento é plausível - uma forma não paramétrica da hipótese de tendências paralelas.

Nota

A restrição de convexidade ($w_j \geq 0$, $\sum w_j = 1$) evita extrapolação - o controle sintético fica dentro do suporte dos doadores. Isso o distingue do DiD com pesos, que pode atribuir pesos negativos para alcançar o balanço.

32.2 Verificação por DGP

O DGP usa painel de 10 unidades $\times$ 20 períodos. A unidade 1 é tratada a partir do ano 11; as unidades 2--10 são doadoras. O efeito verdadeiro é $\tau = 3{,}0$ (permanente, a partir de $T_0 = 11$).

$$y_{it} = \alpha_i + 0{,}3\,t + 3{,}0 \cdot \mathbf{1}[i = 1, t \geq 11] + \varepsilon_{it} \qquad \alpha_i \in \{2, 3, \ldots, 10\}, \quad \alpha_1 = 5$$

O intercepto da unidade tratada ($\alpha_1 = 5$) coincide com o da unidade 5 (doadora), garantindo suporte comum. As inovações $\varepsilon_{it} \sim U(0,1)$ introduzem ruído que impede o SC de usar apenas a unidade 5 como doadora perfeita.

```
seed(42)
input df
id
1
end
for i in 1..=8 { df = append(df, df) }
df |> filter(_n <= 200)

generate df unit = (_n - 1) % 10 + 1
generate df year = (_n - (_n - 1) % 10 - 1) / 10 + 1

// interceptos por unidade: doadores recebem alpha = unit;
   tratado alpha = 5
generate df alpha = 0.0
let j = 2
while j <= 10 {
    replace df alpha = j * 1.0 if unit == j
    j = j + 1
```

```
}
replace df alpha = 5.0 if unit == 1

generate df d = (unit == 1) * (year >= 11)
generate df e = rnormal()
generate df y = alpha + 0.3 * year + 3.0 * d + e

let m = synth("y", 1, 11, df, id="unit", time="year")
display m
```

Listing 32.1: DGP: painel 10×20, tratamento na unidade 1 a partir de t=11

Construção do painel em Hayashi

A função `generate` opera sobre o vetor `_n` (linha corrente, 1-indexado). Para um painel $N \times T$ em formato longo (empilhado por unidade), as fórmulas são:

Variável	Expressão
Unidade $(1, \ldots, N)$	`(_n - 1)% N + 1`
Período $(1, \ldots, T)$	`(_n - (_n-1)%N - 1)/ N + 1`

Resultado

```
Controle Sintético -   Abadie, Diamond, Hainmueller (2010)

Unidade tratada : 1   Outcome: y
T (1ª pós-trat): 11   Doadores: 9   T pré: 10   T pós: 10
RMSPE pré   : 0.3347
RMSPE pós   : 2.7056   razão pós/pré: 8.083

Pesos dos doadores (w > 0.001):
  6                          0.4420
  3                          0.2153
  8                          0.1747
  2                          0.1406
  7                          0.0274
```

	Período	Real	Sintético	Efeito
	1	5.8266	6.0335	
	2	6.5321	6.1112	
	3	6.7458	6.5023	
	4	6.6240	6.8732	
	5	7.4867	7.2037	
	6	7.5689	7.4293	
	7	8.0670	7.6433	
	8	7.8780	8.0722	
	9	8.0202	8.3241	
	10	8.0878	8.6960	
*	11	11.4437	8.8167	2.6270
*	12	12.1888	9.4361	2.7527
*	13	12.0728	9.8984	2.1745
*	14	12.9748	9.5753	3.3995
*	15	12.7653	10.2134	2.5519
*	16	13.2120	10.2424	2.9696
*	17	13.3167	10.6701	2.6465
*	18	13.7205	11.0759	2.6446
*	19	14.3320	11.5282	2.8039
*	20	14.0020	11.7106	2.2913

* pós-tratamento

Leitura dos resultados

Ajuste pré-tratamento. O RMSPE pré-tratamento é $0{,}33$ em uma série cujo desvio padrão é da ordem de 2.5 - um ajuste razoável dado o ruído do DGP ($N = 200$, apenas 10 doadores). A razão pós/pré = 8.1 sinaliza que a divergência pós-tratamento é *oito vezes* maior do que o erro de ajuste pré-tratamento, rejeitando a hipótese de efeito nulo.

Pesos. O algoritmo distribui os pesos principalmente entre as unidades 6, 3, 8 e 2, em vez de concentrar todo o peso na unidade 5 (que tem $\alpha_5 = 5 = \alpha_1$). Isso ocorre porque o ruído amostral ε_{it} torna a trajetória realizada da unidade 5 uma combinação linear imprecisa da unidade 1 no período pré-tratamento. O SC usa múltiplos doadores para cancelar o ruído.

Efeitos estimados. A média dos efeitos pós-tratamento é:

$$\bar{\hat{\tau}} = \frac{1}{10} \sum_{t=11}^{20} \hat{\tau}_t = 2{,}69$$

O valor verdadeiro é $\tau = 3{,}0$; a subestimação de $\sim 10\%$ reflete a imprecisão do ajuste pré-tratamento com apenas 10 doadores e 10 períodos pré.

32.3 Sintaxe

```
synth("outcome", treated_id, t0, df, id="col_unidade", time="
    col_tempo")
synth("outcome", treated_id, t0, df, id="col", time="col",
    covs=["x1","x2"])
```

Argumento	Tipo	Descrição
outcome	string	nome da variável de resultado
treated_id	num/str	identificador da unidade tratada
t0	número	primeiro período pós-tratamento
df	DataFrame	painel em formato longo
id=	opção	coluna de identificação de unidade
time=	opção	coluna de período
covs=	lista	covariáveis adicionais para o ajuste

O argumento covs permite incluir médias de covariáveis pré-tratamento no critério de minimização, além dos valores da variável de resultado. Útil quando os doadores têm trajetórias similares mas características estruturais distintas.

```
load "estados.csv" as df   // id, ano, pib, pop, educ

let m = synth("pib", "SP", 2008, df,
              id="id", time="ano",
              covs=["pop", "educ"])
display m
```

Listing 32.2: Fluxo típico — avaliação de política estadual

Nota

O CS não fornece inferência assintótica padrão (sem erros-padrão, sem p-values). A inferência é feita via *testes de placebo*: aplica-se o SC a cada unidade doadora "como se"fosse tratada e compara a razão pós/pré. Se a razão para a unidade verdadeiramente tratada for excepcional no conjunto de placebos, o efeito é estatisticamente incomum. O HAYASHI exibe a razão pós/pré como heurística descritiva; testes de placebo são implementados manualmente via loop sobre as unidades.

32.4 Validação empírica

O caso de validação synth$_s$*mokingem*validation/cases/*estimaumcontrolesintticosobreumpaine*
*tratamentomdiodo*HAYASHI*comumaimplementaoderefernciaemPython.Execute*hay validate*paraver*

Parte V

Resposta Qualitativa e Dependência Limitada

Capítulo 33

Modelos de Contagem

33.1 Dados de contagem e a inadequação do OLS

Variáveis de resultado que assumem apenas valores inteiros não negativos ($Y \in \{0, 1, 2, \ldots\}$) - número de filhos, visitas ao médico, patentes registradas - violam as premissas do OLS. O OLS pode produzir predições negativas e é ineficiente: a variância de Y cresce com a média (cauda assimétrica à direita), violando homocedasticidade.

Poisson

O modelo de regressão Poisson especifica:

$$Y_i \mid x_i \sim \text{Poisson}(\lambda_i) \qquad \lambda_i = E[Y_i \mid x_i] = \exp(\beta_0 + \beta_1 x_i)$$

A função de log-verossimilhança é:

$$\ell(\beta) = \sum_{i=1}^{n} \left[y_i \left(\beta_0 + \beta_1 x_i \right) - \exp(\beta_0 + \beta_1 x_i) - \ln(y_i!) \right]$$

Uma propriedade útil: o estimador de Poisson converge para β mesmo que Y não seja estritamente Poisson, contanto que $E[Y \mid x] = \exp(x'\beta)$ (estimador quasi-MLE; Gourieroux, Monfort e Trognon 1984).

Binomial Negativa

O modelo Poisson impõe $E[Y \mid x] = \text{Var}[Y \mid x]$ (equidispersão). Quando a variância supera a média (*superdispersão*), o modelo de Binomial Negativa (NB2) generaliza:

$$\mathrm{Var}[Y \mid x] = E[Y \mid x] + \alpha \left(E[Y \mid x]\right)^2$$

O parâmetro $\alpha > 0$ captura a superdispersão. Poisson é o caso-limite $\alpha \to 0$.

Zero-Inflated Negative Binomial (ZINB)

Muitas aplicações apresentam zeros em excesso além do que qualquer modelo de contagem simples prevê - processos de dois estágios: primeiro decide-se se há evento ($D = 1$ com prob. π_i), depois conta-se a frequência. O ZINB combina:

$$P(Y_i = 0) = (1 - \pi_i) + \pi_i \left(1 + \alpha\lambda_i\right)^{-1/\alpha} \qquad P(Y_i = k) = \pi_i \, P_{\mathrm{NB}}(k \mid \lambda_i, \alpha) \quad (k \geq 1)$$

33.2 Verificação por DGP

O DGP cria contagens aproximadamente Poisson com $\lambda_i = e^{0{,}5 + x_i}$, $x \sim U(0,1)$. O estimador deve recuperar $\beta_0 \approx 0{,}5$ e $\beta_1 \approx 1{,}0$.

```
seed(42)
generate df x   = runiform()                          // x ~ U(0,1)
generate df lam = exp(0.5 + 1.0 * x)
generate df u   = runiform()
generate df y   = floor(lam + u)                      // contagem 0

let m_pois = poisson(y ~ x, df)
display m_pois

let m_nb   = nbreg(y_nb ~ x, df)
display m_nb

let m_zinb = zinb(y_zi ~ x, df)
display m_zinb
```

Listing 33.1: Poisson, NegBin e ZINB

Poisson

```
========================= Poisson Regression Results ========================
Dep. Variable:      y   || Log-Likelihood:    102.0744  AIC:   -200.1488
Method:          IRLS   || Pseudo R-sq:         0.7709  N:           500
-----------------------------------------------------------------------------
                   coef    std err         z    P>|z|
const            0.5246     0.0602     8.709    0.000       ***
x                0.9733     0.0960    10.141    0.000       ***
```

$\hat{\beta}_0 = 0{,}52 \approx 0{,}5$ e $\hat{\beta}_1 = 0{,}97 \approx 1{,}0$ - estimativas próximas dos valores verdadeiros.

Binomial Negativa

```
================ Negative Binomial Regression (alpha=0.0010) ===============
                   coef    std err         z    P>|z|
const            0.4613     0.0177    26.025    0.000       ***
x                1.1950     0.0276    43.249    0.000       ***
```

$\hat{\alpha} \approx 0{,}001$ - mínima superdispersão, NegBin colapsa para Poisson. Com dados genuinamente superdispersos $\hat{\alpha}$ é significativamente positivo.

ZINB

```
=============== Zero-Inflated Negative Binomial (ZINB) Results =============
---- Count Model ----
const   0.2994 (0.1347)   **
x       1.2435 (0.2051)   ***
---- Inflate Model (Logit) ----
z0      0.4392 (0.2264)   *
z1      0.5121 (0.3828)
```

A equação de inflação de zeros estima a probabilidade de um zero estrutural; a equação de contagem modela o processo condicional ao evento ocorrer.

33.3 Sintaxe

```
poisson(y ~ x1 + x2, df)
nbreg(y ~ x1 + x2, df)
zinb(y ~ x1 + x2, df)
```

```
zinb(y ~ x1 + x2, df, inflate = z1 + z2)  // equação de inflação separada
```

Estimador	Quando usar
poisson	contagens com equidispersão; quasi-MLE robusto
nbreg	superdispersão ($\hat{\alpha}$ sig. positivo)
zinb	excesso de zeros além do esperado por NB

Nota

Coeficientes de Poisson e NB são log-razões: $\beta_1 = 1$ implica que um aumento unitário em x multiplica a contagem esperada por $e^1 \approx 2{,}72$. Para obter *efeitos marginais médios* (AME): use `margins(modelo)`.

33.4 Validação empírica

O dataset fertil2 do Wooldridge é usado nos casos de validação $\text{poisson}_{f}ertil2e$

Capítulo 34

Modelos de Resposta Limitada

34.1 Censura e truncagem

Em muitas aplicações o resultado só é observado para parte da amostra: salários de mulheres que trabalham (seleção), horas de lazer truncadas em zero (censura), gastos com bens de luxo (muitos zeros, mas não zeros estruturais). O OLS aplicado à variável censurada é viesado mesmo com erros gaussianos.

Tobit — censura em zero

O modelo Tobit (Tobin 1958) especifica uma variável latente:

$$y_i^* = x_i'\beta + \varepsilon_i, \quad \varepsilon_i \sim N(0, \sigma^2) \qquad y_i = \max(0,\ y_i^*)$$

O OLS sobre y_i (ignorando a censura) subestima β porque os zeros "achatam"a distribuição condicional. O estimador de máxima verossimilhança (MLE) usa a função de verossimilhança completa:

$$\ell(\beta, \sigma) = \sum_{y_i = 0} \ln \Phi\left(-\frac{x_i'\beta}{\sigma}\right) + \sum_{y_i > 0} \ln \phi\left(\frac{y_i - x_i'\beta}{\sigma}\right) - \ln \sigma$$

Heckman — seleção amostral

O modelo de seleção amostral de Heckman (1979) separa:

1. **Equação de seleção**: $s_i = \mathbf{1}[\gamma_0 + \gamma_1 z_i + v_i > 0]$
2. **Equação de resultado**: $y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$, observada apenas quando $s_i = 1$.

Se $\text{Cov}(\varepsilon_i, v_i) \neq 0$, o OLS sobre a sub-amostra selecionada é viesado. A correção de Heckman (dois passos) adiciona a *Razão Inversa de Mills* (IMR) $\lambda_i = \phi(\hat{\gamma}' z_i)/\Phi(\hat{\gamma}' z_i)$ como regressor para controlar o viés de seleção.

34.2 Verificação por DGP

```
seed(42)
generate df x      = rnormal()
generate df z      = rnormal()
generate df e      = (rnormal() - 0.5) * 1.7321   // ~ N(0,1)
generate df v      = (rnormal() - 0.5) * 1.7321

// Tobit: y* = 1 + 2*x + e; y = max(0, y*)
generate df ystar   = 1.0 + 2.0 * x + e
generate df y_tobit = ystar * (ystar > 0)

// Heckman: selecionado se 0.5 + 1.5*z + v > 0
generate df s      = (0.5 + 1.5 * z + v) > 0
generate df y_out = (1.0 + 2.0 * x + e) * s
```

Listing 34.1: DGP Tobit e Heckman

Tobit

```
let m_tobit = tobit(y_tobit ~ x, df)
display m_tobit
```

```
Tobit -   MLE  (censura inferior em 0)

Obs: 500  Censuradas: 0  Não-cens.: 500  Log-L: -359.2116   : 0.4963

Variável          coef          SE           z      P>|z|
const           0.9050      0.0442      20.470     0.0000  ***
x               2.1956      0.0787      27.913     0.0000  ***
sigma           0.4963
```

O Tobit recupera $\hat{\beta}_0 = 0{,}905 \approx 1{,}0$ e $\hat{\beta}_1 = 2{,}196 \approx 2{,}0$, parâmetros da variável latente y^*. Com zero censura neste DGP (todos $y^* > 0$), equivale

ao OLS - o teste do modelo é verificado pela capacidade do MLE de convergir e recuperar $\sigma \approx 0{,}5$.

Heckman

```
let m_heck = heckman(y_out ~ x, s ~ z, df)
display m_heck
```

```
Heckman Two-Step -   Heckman (1979)

Obs (total): 500   Selecionadas: 484ˆ
: 0.3608 ˆ   _: 0.4971 ˆ    = ˜ _: 0.1793

Equação de resultado  (y | z=1)
Variável           coef          SE           z     P>|z|
const            0.9092      0.0455      20.001    0.0000   ***
x                2.1821      0.0804      27.142    0.0000   ***
lambda (IMR)     0.1793      0.1917       0.935    0.3495

Equação de seleção:  const  0.1752  z  8.6806**
```

O coeficiente da IMR não é significativo ($p = 0{,}35$) - esperado, pois no DGP $\mathrm{Cov}(\varepsilon, v)$ é fraco ($\rho = 0{,}36$). Com $N = 500$, a potência para detectar correlação moderada é limitada. A equação de resultado recupera corretamente $\hat{\beta}_1 \approx 2{,}18$.

34.3 Sintaxe

```
tobit(y ~ x1 + x2, df)                          // censura em 0 (padrão)
tobit(y ~ x1 + x2, df, ll=0, ul=10)             // censura dupla [0, 10]
heckman(y ~ x1 + x2, s ~ z1 + z2, df)           // dois passos
```

Opção	Descrição
ll	limite inferior de censura (padrão: 0)
ul	limite superior de censura (padrão: nenhum)

Atenção

Para que o Heckman identifique o modelo, a equação de seleção deve conter pelo menos uma variável **exclusa** do resultado (z mas não x). Sem exclusão, o modelo é identificado só por não-linearidade funcional - estimativas instáveis.

34.4 Validação empírica

O dataset mroz é usado no caso de validação heckman$_m rozem$validation/cases/.$Ocasoestim$

Capítulo 35

Modelos de Escolha Múltipla

35.1 Resposta ordinal e nominal

Quando o resultado assume mais de duas categorias, duas estruturas são possíveis:

- **Resposta ordinal**: categorias com ordem natural (muito insatisfeito, neutro, muito satisfeito; nível educacional). Modelos ordenados (logit/probi preservam essa estrutura.
- **Resposta nominal**: categorias sem ordem (escolha de modal de transporte; tipo de emprego). Logit multinomial sem restrição de ordem.

Logit e Probit Ordenados

O modelo supõe uma variável latente $y^* = x'\beta + \varepsilon$ e cortes $c_1 < c_2 < \cdots < c_{J-1}$:

$$y = j \quad \Longleftrightarrow \quad c_{j-1} < y^* \leq c_j$$

O logit ordenado usa $\varepsilon \sim \text{Logistic}(0,1)$; o probit ordena usa $\varepsilon \sim N(0,1)$. Ambos estimam β e os cortes por MLE.

Logit Multinomial

Para J alternativas, com alternativa $j = J$ como base:

$$P(y_i = j \mid x_i) = \frac{\exp(\alpha_j + \beta_j' \, x_i)}{1 + \sum_{k=1}^{J-1} \exp(\alpha_k + \beta_k' \, x_i)} \quad j = 1, \ldots, J-1$$

Os coeficientes (α_j, β_j) medem o efeito de x sobre as chances relativas de j versus a alternativa base.

35.2 Verificação por DGP

```
seed(42)
generate df x     = rnormal()
generate df e     = (rnormal() - 0.5) * 1.7321
generate df ystar = 1.5 * x + e
// Categorias 1, 2, 3 com cortes em 0.0 e 1.0
generate df y_ord = 1 + (ystar >= 0.0) + (ystar >= 1.0)

// Multinomial via inverse-CDF
generate df denom  = 1.0 + exp(-1.0 + 2.0*x) + exp(0.5)
generate df p1     = 1.0 / denom
generate df p12    = (1.0 + exp(-1.0 + 2.0*x)) / denom
generate df um     = rnormal()
generate df y_mlog = 1 + (um >= p1) + (um >= p12)
```

Listing 35.1: DGP para modelos ordenados e multinomial

Ordered Logit

```
let m_olog = ologit(y_ord ~ x, df)
display m_olog
```

```
============================ Ordered Logit Results ============================
N=500  Log-L=-396.4473  Pseudo R²=0.1878
------------------------------- Coefficients -------------------------------
x         4.8137  (0.4633)  z=10.39  ***
Thresholds:  cut1=-0.1472  cut2=3.2191
```

Ordered Probit

```
let m_opro = oprobit(y_ord ~ x, df)
display m_opro
```

```
============================ Ordered Probit Results ===========================
N=500  Log-L=-392.8369  Pseudo R²=0.1952
```

```
------------------------------ Coefficients ------------------------------
x         2.8683   (0.2603)   z=11.02   ***
Thresholds:  cut1=-0.0768   cut2=1.9076
```

A razão entre coeficientes logit e probit é $4{,}81/2{,}87 \approx 1{,}68$, próximo de $\pi/\sqrt{3} \approx 1{,}81$ - a relação teórica entre as duas escalas.

Logit Multinomial

```
let m_mlog = mlogit(y_mlog ~ x, df)
display m_mlog
```

```
==================== Multinomial Logit Regression Results ==================
N=500   Log-L=-513.7171   Pseudo R²=0.0448   AIC=1035.43
------------------------------ y=1 vs base y=3 ------------------------------
const    -0.0809   (0.2024)
x        -0.9006   (0.4069)   *
------------------------------ y=2 vs base y=3 ------------------------------
const    -1.5279   (0.2542)   ***
x         2.0825   (0.4119)   ***
```

Para $y=2$ versus base $y=3$: $\hat{\beta}_x = 2{,}08$ (DGP verdadeiro: $\beta_{x,2} - \beta_{x,3} = 2{,}0$). O sinal negativo de x em $y=1$ vs $y=3$ reflete que x alto aumenta as chances de $y=2$ (que tem $e^{0{,}5}$ de probabilidade base) relativo a $y=1$.

35.3 Sintaxe

```
ologit(y ~ x1 + x2, df)
oprobit(y ~ x1 + x2, df)
mlogit(y ~ x1 + x2, df)
mlogit(y ~ x1 + x2, df, base=1)    // alternativa base explí
    cita
```

Nota

O logit multinomial impõe a propriedade de **independência de alternativas irrelevantes** (IIA): a razão de probabilidades entre duas alternativas não depende das demais. Para dados de painel de escolha com alternativas correlacionadas, use clogit (cap. 47).

35.4 Validação empírica

O dataset beauty do Wooldridge é usado nos casos de validação ologit$_b eautye$

Capítulo 36

Análise de Sobrevivência

36.1 Tempos de falha e censura

A análise de sobrevivência modela o *tempo até um evento* T - falência de empresa, morte, abandono escolar. O desafio central é a **censura à direita**: quando a observação termina antes do evento, sabemos apenas $T > c$ (tempo de observação c), não o tempo exato.

Dois conceitos fundamentais:

$$S(t) = P(T > t) \quad \text{(função de sobrevivência)} \qquad h(t) = \lim_{\delta \to 0} \frac{P(t \le T < t + \delta \mid T \ge t)}{\delta} \quad \text{(funç}$$

$S(t)$ decresce de 1 para 0; $h(t)$ mede a taxa instantânea de falha dado que o indivíduo sobreviveu até t.

Kaplan-Meier

O estimador não-paramétrico de Kaplan-Meier (1958) estima $S(t)$ sem premissas distribucionais:

$$\hat{S}(t) = \prod_{t_j \le t} \left(1 - \frac{d_j}{n_j}\right)$$

onde d_j é o número de eventos no tempo t_j e n_j o número em risco imediatamente antes.

Cox — Hazard Proporcional

O modelo de Cox (1972) especifica:

$$h(t \mid x_i) = h_0(t)\,\exp(\beta' x_i)$$

A função de hazard baseline $h_0(t)$ é deixada não-paramétrica. Os coeficientes β são estimados pela verossimilhança parcial, sem especificar h_0. O exponencial e^{β_j} é a ***razão de hazard***: um aumento unitário em x_j multiplica o hazard por e^{β_j}.

36.2 Verificação por DGP

O DGP usa tempos de falha Weibull com parâmetro de forma 1,5 e hazard proporcional em $x \in \{0, 1\}$. O verdadeiro log-hazard ratio é $\beta_x = 0{,}8$.

```
seed(42)
generate df x      = rbernoulli(0.5)
generate df u      = runiform()
generate df t_true = exp(-ln(u) / exp(0.8 * x))   // Weibull, beta=0.8
generate df c      = 3.0
generate df time   = t_true * (t_true <= c) + c * (t_true > c)
generate df event  = (t_true <= c)

let m_km  = km(time, event, df)
display m_km

let m_cox = cox(time ~ x, df, event=event)
display m_cox
```

Listing 36.1: DGP de sobrevivência — Weibull com hazard proporcional

Kaplan-Meier

```
============== Kaplan-Meier Survival Estimate ==============
Observations:                300
Events:                      246
Median survival:          1.6319
```

A mediana de sobrevivência é 1,63: metade dos indivíduos experimenta o evento antes desse tempo.

Cox

```
======================== Cox Proportional Hazards Model ========================
Observations: 300   Events: 246   Log-Likelihood: -1235.9213
Concordance:  0.6725
------------------------------------------------------------------------------
Variable   |     coef |  std err |        z |  P>|z| | exp(coef)
x          |   0.7018 |   0.1306 |    5.372 |  0.000 |    2.0174
```

$\hat{\beta}_x = 0{,}702$ (verdadeiro: 0,8) - dentro do intervalo de confiança. O hazard ratio estimado $e^{0{,}702} = 2{,}02$: indivíduos com $x = 1$ têm hazard instantâneo duas vezes maior do que $x = 0$. O índice de concordância $C = 0{,}67$ indica discriminação moderada (0,5 = aleatório, 1,0 = perfeito).

36.3 Sintaxe

```
km(time, event, df)                              // Kaplan-Meier
cox(time ~ x1 + x2, df, event=col_evento)    // Cox PH
```

A coluna event deve ser 1 para evento e 0 para censura.

Nota

O modelo de Cox assume *hazards proporcionais*: $h(t \mid x = 1)/h(t \mid x = 0)$ é constante no tempo. Violações podem ser detectadas via resíduos de Schoenfeld ou interações com $\ln(t)$.

Parte VI

Métodos Avançados

Capítulo 37

Painel Avançado I — GMM, PCSE e GEE

37.1 Limitações dos estimadores estáticos

O cap. 20 apresentou FE e RE para painéis com efeito individual constante. Três extensões importantes surgem em aplicações empíricas:

1. **Dinâmica**: quando $y_{i,t-1}$ é regressor, FE é viesado (*viés de Nickell*): $\hat{\rho}_{\text{FE}} \xrightarrow{p} \rho - O(1/T)$. O estimador Arellano-Bond usa GMM em primeiras diferenças para contornar.

2. **Dependência cross-seccional**: quando erros de diferentes unidades são correlacionados (dados macroeconômicos de países), os SEs OLS e FE são inválidos. PCSE (Beck e Katz 1995) corrige.

3. **Dados não-gaussianos em painel**: GEE (Liang e Zeger 1986) estende modelos de margem para dados de painel com famílias exponenciais.

Arellano-Bond (AB)

O estimador diff-GMM de Arellano e Bond (1991) toma primeiras diferenças para eliminar o efeito individual:

$$\Delta y_{it} = \rho \, \Delta y_{i,t-1} + \beta \, \Delta x_{it} + \Delta \varepsilon_{it}$$

Instrumentos: $y_{i,t-2}, y_{i,t-3}, \ldots$ são válidos para $\Delta y_{i,t-1}$ pois são não correlacionado com $\Delta \varepsilon_{it}$, contanto que os erros em nível não sejam serialmente correlacionados de ordem 2 (teste m_2).

37.2 Verificação por DGP

```
seed(42)
generate df unit  = (_n - 1) % 50 + 1
generate df t     = (_n - (_n-1) % 50 - 1) / 50 + 1
generate df alpha = unit * 0.05        // efeito individual correlacionado com x
generate df x     = alpha + (rnormal() - 0.5) * 1.7321
generate df e     = (rnormal() - 0.5) * 1.7321
generate df y     = 1.0 + 1.0 * x + alpha + e  // verdadeiro: b=1.0

// OLS viesado (mistura alpha_i com x)
let m_ols  = ols(y ~ x, df)

// FE consistente (elimina alpha_i)
let m_fe   = fe(y ~ x, df, id=unit)

// AB-GMM (dinâmico com lags como instrumentos)
let m_ab   = ab(y ~ x, df, id=unit, time=t, lags=2)

// PCSE
let m_pcse = pcse(y ~ x, df, id=unit, time=t)

// GEE
let m_gee  = gee(y ~ x, df, id=unit)
```

Listing 37.1: DGP de painel com endogeneidade via efeito individual

OLS vs. FE

```
OLS:  const=1.330  x=1.736***  (viés: +0.74 por mistura de alpha_i)
FE:              x=1.089***  (próximo de 1.0 - efeito real)
```

Arellano-Bond

```
====================== Arellano-Bond Diff-GMM (One-Step) ======================
Observations: 900  Entities: 50  Instruments: 3  Sargan df: 1
```

```
Variable       |    Coef | Std Err |       z |  P>|z|
LD.y           | -0.0242 |  0.0472 |  -0.512 |  0.608Δ
x              |  1.0604 |  0.0464 |  22.869 |  0.000  ***

Sargan: ²(1)=0.7549  p=0.3849   (instrumentos válidos)
AB m1: z=-6.22  p=0.000 ***      (AR(1) esperado em FD)
AB m2: z=0.001  p=0.999          (sem AR(2) - instrumentos OK)
```

AB estima $\hat{\beta}_{\Delta x} = 1{,}06 \approx 1{,}0$. O lag $\hat{\rho}_{\mathrm{LD.y}} \approx 0$ confirma que o DGP não tem dinâmica real. O teste de Sargan não rejeita a validade dos instrumentos; m_2 não rejeita a ausência de autocorrelação de ordem 2.

PCSE

```
PCSE:  const=1.330  SE=0.0637   x=1.736  SE=0.0498
(SEs maiores que OLS - corrige correlação cross-seccional)
```

GEE

```
GEE:  const=1.330 (SE robust=0.0612)  x=1.736 (SE robust=0.0330)
Scale: 0.3966   QIC: 399.8152
```

GEE com estrutura de correlação exchangeable fornece estimativas de margem (populacionais), com SEs robustos para dependência intra-grupo.

37.3 Sintaxe

```
ab(y ~ x, df, id=unit, time=t, lags=2)
pcse(y ~ x, df, id=unit, time=t)
gee(y ~ x, df, id=unit)
gee(y ~ x, df, id=unit, family=binomial, corstr=exchangeable)
```

Estimador	Pressuposto	Quando usar
ab	erros i.i.d. em nível	painel dinâmico com pequeno T
pcse	corr. cross-seccional	macro painel (N pequeno, T médio)
gee	margem populacional	painel com outcome não-gaussiano

Capítulo 38

Regressões Aparentemente Não-Relacionadas

38.1 O estimador de Zellner

Quando se estima um sistema de M equações com os mesmos (ou diferentes) regressores e os erros são correlacionados entre equações, o OLS equação a equação é ineficiente. O estimador **SUR** (*Seemingly Unrelated Regressions*) de Zellner (1962) explora essa correlação via GLS conjunto.

Sistema de M equações:

$$y_m = X_m \beta_m + \varepsilon_m, \quad m = 1, \ldots, M$$

Empilhando: $Y = X\beta + \varepsilon$, com $E[\varepsilon\varepsilon'] = \Sigma \otimes I_n$, onde Σ é a matriz de covariância entre-equações ($M \times M$).

O estimador GLS conjunto:

$$\hat{\beta}_{\text{SUR}} = (X'(\Sigma^{-1} \otimes I)X)^{-1} X'(\Sigma^{-1} \otimes I)\, Y$$

Ganho de eficiência: SUR $\geq$ OLS quando Σ tem elementos fora da diagonal não nulos - i.e., quando os erros das equações são correlacionados. Se Σ for diagonal ou se os regressores forem idênticos, SUR = OLS.

38.2 Verificação por DGP

DGP com dois erros fortemente correlacionados ($\rho = 0{,}7$):

$$y_1 = 1 + 2\,x_1 + \varepsilon_1 \qquad y_2 = 0{,}5 + 1{,}5\,x_2 + \varepsilon_2$$

$$\mathrm{Cov}(\varepsilon_1, \varepsilon_2) = 0{,}7\,\sigma_1\,\sigma_2$$

```
seed(42)
generate df x1 = rnormal()
generate df x2 = rnormal()
generate df e1 = (rnormal() - 0.5) * 1.7321
generate df e2 = 0.7 * e1 + 0.7141 * (rnormal() - 0.5) *
   1.7321
generate df y1 = 1.0 + 2.0 * x1 + e1
generate df y2 = 0.5 + 1.5 * x2 + e2

let m_ols1 = ols(y1 ~ x1, df)
let m_ols2 = ols(y2 ~ x2, df)
esttab(m_ols1, m_ols2)

let m_sur = sur(df, y1 ~ x1, y2 ~ x2)
display m_sur
```

Listing 38.1: DGP SUR e comparação com OLS equação a equação

OLS equação a equação

	(1)	(2)
	OLS	OLS
x1	2.1956***	
	(0.0788)	
x2		1.5097***
		(0.0779)
Constant	0.9050***	0.4879***
N	500	500
R^2	0.6091	0.4300

SUR

```
              Seemingly Unrelated Regressions (SUR)
        Cross-Equation Error Correlation Σ():
 [  0.2463   0.1689 ]
 [  0.1689   0.2419 ]

Equation: y1 -   const=0.9513  x1=2.1003***   R²=0.6080
Equation: y2 -   const=0.4591  x2=1.5653***   R²=0.4294
```

A correlação estimada entre erros é $\hat{\Sigma}_{12}/\sqrt{\hat{\Sigma}_{11}\hat{\Sigma}_{22}} = 0{,}169/\sqrt{0{,}246 \times 0{,}242} \approx 0{,}69$, próximo do verdadeiro $0{,}7$. O SUR ganha eficiência via corte nos SEs: $\hat{\sigma}_{\text{SUR}}(x_1) < \hat{\sigma}_{\text{OLS}}(x_1)$.

38.3 Sintaxe

```
sur(df, y1 ~ x1 + x2, y2 ~ x3 + x4)
sur(df, y1 ~ x1, y2 ~ x2, y3 ~ x3)        // 3 equações
```

O primeiro argumento é o DataFrame; os demais são fórmulas (uma por equação).

Nota

Se todas as equações tiverem os mesmos regressores, SUR = OLS e não há ganho de eficiência. O ganho é máximo quando os regressores são ortogonais entre equações e a correlação entre erros é alta.

38.4 Validação empírica

O dataset de investimento grunfeld do Wooldridge é usado no caso de validação sur_grunfeldem validation/cases/. O caso estima um sistema SUR de duas equações no HAYASHI e comp

Capítulo 39

Regularização e Seleção de Variáveis

39.1 O problema de dimensionalidade

Com k regressores e N pequeno, o OLS sofre de *overfitting*: ajusta o ruído da amostra além do sinal, elevando o erro fora da amostra. Quando $k > N$, o OLS nem sequer é identificado. A regularização penaliza a magnitude dos coeficientes, trocando viés por variância reduzida.

LASSO

O LASSO (Tibshirani 1996) minimiza:

$$\hat{\beta}^{\text{LASSO}} = \arg\min_{\beta} \left\{ \sum (y_i - x_i'\beta)^2 + \lambda \sum_{j=1}^{k} |\beta_j| \right\}$$

A penalidade ℓ_1 força coeficientes pequenos a zero exatamente, fazendo *seleção de variáveis* automática. O caminho de regularização - LASSO para diferentes λ - revela quais variáveis entram conforme λ decresce de ∞ (modelo nulo) a 0 (OLS).

Ridge

A penalidade ℓ_2 do Ridge (Hoerl e Kennard 1970) encolhe coeficientes em direção a zero sem zerá-los:

$$\hat{\beta}^{\text{Ridge}} = (X'X + \lambda I)^{-1} X'y$$

Útil quando muitas variáveis têm pequeno efeito (contínuo de preditores fracos). Não realiza seleção de variáveis.

Elastic Net

O Elastic Net (Zou e Hastie 2005) combina ℓ_1 e ℓ_2:

$$\min_{\beta} \left\{ \sum (y_i - x_i'\beta)^2 + \lambda \left[\rho \sum |\beta_j| + (1-\rho) \sum \beta_j^2 \right] \right\}$$

Herda seleção de variáveis do LASSO e estabilidade em grupos correlacionados do Ridge.

39.2 Verificação por DGP

DGP com 10 regressores, apenas x_1 e x_2 com efeito não-nulo. $N = 200$ - regime onde regularização importa.

```
seed(42)
// 10 regressores; y = 1 + 2*x1 + 3*x2 + 0*(x3..x10) + e
generate df x1  = rnormal()
...
generate df y   = 1.0 + 2.0*x1 + 3.0*x2 + e

let m_ols   = ols(y ~ x1+x2+x3+x4+x5+x6+x7+x8+x9+x10, df)
let m_lasso = lasso(y ~ x1+x2+x3+x4+x5+x6+x7+x8+x9+x10, df,
   alpha=0.1)
let m_ridge = ridge_reg(y ~ x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,
   df, alpha=0.5)
let m_enet  = enet(y ~ x1+x2+x3+x4+x5+x6+x7+x8+x9+x10, df,
   alpha=0.5)
```

Listing 39.1: DGP esparso — 2 de 10 variáveis ativas

OLS

```
OLS: R²=0.801  (bom ajuste na amostra, mas x3..x10 poluem com coef não-nulos)
const=0.68  x1=1.93***  x2=2.88***  x3=0.17  x4=0.07  ...  x10=-0.11
```

LASSO

```
LASSO  =0.1: R²=0.772  vars ativas: 2
  x1=1.552  x2=2.506  x3..x10 = 0.000  (zerando corretamente)
```

Com $\alpha = 0{,}1$ o LASSO identifica os dois preditores verdadeiros e zera os oito irrelevantes.

Ridge

```
Ridge  =0.5: R²=0.801
  x1=1.878  x2=2.793  x3=0.172  x4=0.081  ...
  (encolhe mas não zera nenhum)
```

Elastic Net

```
ElasticNet  =0.5 l1_ratio=0.5: R²=0.589  vars ativas: 2
  x1=0.789  x2=1.550  x3..x10 = 0.000
```

Elastic Net também seleciona x_1 e x_2, mas com maior penalização total (combinando ℓ_1 e ℓ_2) os coeficientes ficam mais encolhidos.

39.3 Sintaxe

```
lasso(y ~ x1 + x2 + x3, df, alpha=0.1)      // alpha = lambda de penalização
ridge_reg(y ~ x1 + x2 + x3, df, alpha=0.5)
enet(y ~ x1 + x2 + x3, df, alpha=0.5)       // alpha = mistura L1/(L1+L2)
```

Parâmetro	LASSO	Ridge / Enet
alpha	penalização λ	penalização λ
alpha=0	OLS	Ridge puro (Enet)
alpha=1	L1 máximo	L1 puro (Enet)

Nota

O valor ótimo de α é tipicamente escolhido por validação cruzada (k-fold). Em amostras pequenas, LASSO com α via *BIC-LASSO* tende a ser mais conservador do que via CV.

39.4 Validação empírica

O dataset de preços de imóveis hprice1 do Wooldridge é usado nos casos de validação ridge$_h price1$,

Capítulo 40

Bootstrap

40.1 Inferência por reamostragem

Os erros-padrão analíticos derivados da teoria assintótica assumem, em geral, que os resíduos seguem uma distribuição específica (normal, homocedástica, i.i.d.). O **bootstrap** de Efron (1979) substitui premissas distribucionais por reamostragem: replica o processo de estimação B vezes em amostras com reposição do próprio conjunto de dados.

Bootstrap de pares

A abordagem mais simples reamostrea pares (y_i, x_i):

1. Para $b = 1, \ldots, B$: amostrar n observações com reposição.
2. Estimar $\hat{\theta}^{(b)}$ na amostra reamostrada.
3. $\widehat{\mathrm{SE}}_{\mathrm{boot}} = \mathrm{sd}(\hat{\theta}^{(1)}, \ldots, \hat{\theta}^{(B)})$.

O IC percentil bootstrap de 95% é o intervalo $[Q_{0{,}025},\, Q_{0{,}975}]$ da distribuição empírica de $\hat{\theta}^{(b)}$.

Vantagem: válido sem saber a distribuição de $\hat{\theta}$ - robusto a erros não-gaussianos, heterocedasticidade e amostras pequenas onde a aproximação normal é ruim.

40.2 Verificação por DGP

DGP com erro uniforme em $[-2, 2]$ - não-gaussiano. Com $N = 200$, o SE analítico do OLS pode subestimar a variância real.

```
seed(42)
generate df x = rnormal()
generate df e = (rnormal() - 0.5) * 4.0     // U(-2, 2) - não
    gaussiano
generate df y = 1.0 + 2.0 * x + e

// OLS padrão
let m_ols  = ols(y ~ x, df)
display m_ols

// Bootstrap
let m_boot = bootstrap("ols", y ~ x, df, reps=500)
display m_boot
```

Listing 40.1: Comparação SE analítico vs. bootstrap

OLS analítico

```
OLS: R²=0.178
  const=1.046   SE=0.157    x=1.828   SE=0.279
```

Bootstrap ($B = 500$)

```
                Bootstrap SE - ols (n=500/500)

Variável ^          SE orig.     SE boot     IC inf      IC sup
const     1.0462     0.1566      0.1545      0.7235      1.3710
x         1.8276     0.2787      0.2679      1.3313      2.3910
```

SE bootstrap (0,155 e 0,268) e analítico (0,157 e 0,279) são similares aqui, indicando que os SEs analíticos do OLS são adequados mesmo com erros uniformes quando $N = 200$. A concordância valida a implementação: ambos convergem para o mesmo desvio-padrão assintótico.

40.3 Sintaxe

```
// bootstrap genérico - qualquer estimador implementado
bootstrap("ols",    y ~ x, df, reps=500)
bootstrap("logit",  y ~ x, df, reps=500)
bootstrap("qreg",   y ~ x, df, reps=500, q=0.5)

// alias bootse
bootse(y ~ x, df, reps=500)                    // ols por padrão (legado)
```

Opção	Padrão	Descrição
reps / n	1000	número de réplicas bootstrap
alpha	0.05	nível para ICs (bilateral)

Nota

O estimador de bootstrap também pode estimar SEs do PSM (cap. 31), onde a distribuição assintótica do ATT é não-padrão. Nesse contexto, o bootstrap de pares é consistente (Abadie e Imbens 2008).

Capítulo 41

Análise Multivariada

41.1 Redução de dimensionalidade e estrutura latente

Quando se observa um vetor de variáveis $\mathbf{v} = (v_1, \ldots, v_p)'$ que compartilham variação comum, é útil resumir essa variação em poucas dimensões. Duas abordagens se complementam:

- **PCA**: maximiza variância explicada por componentes ortogonais (solução álgebrica - decomposição espectral da matriz de covariância).
- **Análise Fatorial**: modela variáveis como combinação linear de fatores latentes não-observados + unicidade (parte idiossincrática).

Análise de Componentes Principais (PCA)

$$\mathbf{v} = W\,\mathbf{f} + \mathbf{e} \qquad W = \text{eigenvectors de } \Sigma_v$$

O j-ésimo componente principal $\text{PC}_j = w_j'\mathbf{v}$ captura a j-ésima maior fração da variância total. Os eigenvalues medem a variância de cada PC; a soma de todos os eigenvalues é $\text{tr}(\Sigma_v) = \sum \text{Var}(v_j)$.

Análise Fatorial

$$\mathbf{v} = \Lambda\,\mathbf{f} + \boldsymbol{\delta} \qquad \mathbf{f} \perp \boldsymbol{\delta}$$

Λ são as *cargas fatoriais* (loadings): λ_{ij} mede o quanto o fator j contribui para v_i. A *comunalidade* $h_i^2 = \sum_j \lambda_{ij}^2$ é a fração da variância de v_i explicada pelos fatores comuns.

Correlação Canônica

Dadas duas baterias de variáveis $X = (x_1, \ldots, x_p)$ e $Y = (y_1, \ldots, y_q)$, a análise canônica encontra vetores a e b tais que $\mathrm{Corr}(a'X, b'Y)$ é maximizada. Fornece $\min(p, q)$ pares canônicos.

41.2 Verificação por DGP

DGP com 2 fatores latentes e 5 variáveis observadas.

```
seed(42)
generate df f1 = rnormal()
generate df f2 = rnormal()
// v1, v2: cargas altas em f1; v3, v4: cargas altas em f2; v5
   : ambos
generate df v1 = 0.9*f1 + 0.1*f2 + e1
generate df v2 = 0.8*f1 + 0.2*f2 + e2
generate df v3 = 0.1*f1 + 0.9*f2 + e3
generate df v4 = 0.2*f1 + 0.8*f2 + e4
generate df v5 = 0.5*f1 + 0.5*f2 + e5

let m_pca = pca(df, v1, v2, v3, v4, v5, n=2)
display m_pca

let m_fac = factor(df, v1, v2, v3, v4, v5, n=2, rotation=
   varimax)
display m_fac

let m_can = cancorr(df, xvars=["v1","v2"], yvars=["v3","v4","
   v5"])
display m_can
```

Listing 41.1: DGP fatorial e análise multivariada

PCA

```
Principal Component Analysis
```

```
 Componente    Var Expl.    % Acum.   Eigenvalue
 PC1              0.5939     0.5939       2.9695
 PC2              0.2437     0.8377       1.2187

Loadings:
          PC1      PC2
 v1     0.677   -0.641
 v2     0.808   -0.450
 v3     0.693    0.623
 v4     0.790    0.466
 v5     0.868   -0.002
```

PC1 (59,4% da variância) tem cargas positivas em todas as variáveis - captura o componente comum. PC2 (24,4%) contrasta o grupo f_1 (v1, v2) com o grupo f_2 (v3, v4). Os dois PCs somam 83,8% da variância total.

Análise Fatorial

```
        Factor Analysis - Varimax

Variável         F1        F2  Comunalit.
 v1            0.027    -0.932       0.869
 v2            0.255    -0.889       0.856
 v3            0.931    -0.048       0.869
 v4            0.888    -0.228       0.841
 v5            0.613    -0.614       0.753
```

Após rotação varimax, F1 carrega em v3 e v4 (associados ao fator latente f_2); F2 carrega em v1 e v2 (fator f_1). Comunalidades acima de 0,75 confirmam que os fatores comuns explicam grande parte da variância individual.

Correlação Canônica

```
============== Canonical Correlation Analysis ==============
Observations: 300  Wilks' Lambda: 0.5396  F=35.54  p=0.000
  CC1: 0.6706    CC2: 0.1396
X vars: v1, v2  |  Y vars: v3, v4, v5
============================================================
```

A primeira correlação canônica $\hat{\rho}_1 = 0{,}671$ captura a associação entre o bloco {v1, v2} e o bloco {v3, v4, v5} através do fator latente compartilhado.

41.3 Sintaxe

```
pca(df, v1, v2, v3, ..., n=2)                    // n componentes a reter
factor(df, v1, v2, v3, ..., n=2, rotation=none)  // rotation: none | varimax
cancorr(df, xvars=["x1","x2"], yvars=["y1","y2"])
```

Capítulo 42

Modelos Lineares Generalizados

42.1 A família exponencial

Os Modelos Lineares Generalizados (Nelder e Wedderburn 1972) unificam OLS, Logit, Poisson e muitos outros estimadores sob um arcabouço comum. Três componentes definem um GLM:

1. **Componente aleatório**: $Y_i \sim$ família exponencial com média μ_i.
2. **Preditor linear**: $\eta_i = x_i'\beta$.
3. **Função de ligação**: $g(\mu_i) = \eta_i$, que conecta média à escala linear.

Família	Link padrão	Variância	Aplicação
Gaussiana	identidade	σ^2	OLS
Binomial	logit	$\mu(1-\mu)$	dados binários
Poisson	log	μ	contagens
Gamma	log	μ^2/ϕ	dados positivos com caudas

A estimação por IRLS (Iteratively Reweighted Least Squares) é eficiente e converge rapidamente.

GLM Gamma — dados positivos com caudas pesadas

Gastos, tamanho de empréstimos e durações seguem distribuições assimétricas à direita com variância proporcional ao quadrado da média. O GLM Gamma com link log modela $E[Y \mid x] = \exp(x'\beta)$ com $\mathrm{Var}[Y \mid x] \propto \mu^2$.

PPML — comércio internacional

O estimador PPML (Poisson Pseudo MLE) de Santos Silva e Tenreyro (2006) usa a equação de estimação Poisson mesmo quando Y não é inteira (ex: valores de comércio). Ele é robusto a zeros e a heterocedasticidade multiplicativa - problemas comuns em equações gravitacionais de comércio.

42.2 Verificação por DGP

```
seed(42)
generate df x  = rnormal()
generate df mu = exp(0.5 + 1.0 * x)
// Gamma: média=mu, variância ~ mu² (simulado via lognormal)
generate df y_gamma = mu * exp(rnormal() * 0.6)

let m_gamma = glm(y_gamma ~ x, df, family=gamma, link=log)
display m_gamma

// PPML com dados de "comércio" (mistura zeros e positivos)
generate df x2    = rnormal()
generate df trade = round(exp(0.3 + 1.2*x + 0.8*x2) * (
   runiform() > 0.2))
let m_ppml = glm(trade ~ x + x2, df, family=poisson, link=log
   )
display m_ppml
```

Listing 42.1: GLM Gamma e PPML

GLM Gamma

O modelo Gamma deve recuperar estimativas positivas de intercepto e inclinação, próximas ao DGP de média logarítmica, com a dispersão resumindo a variância relativa remanescente.

PPML

PPML deve recuperar efeitos positivos para as duas covariáveis enquanto lida com os zeros da variável trade.

42.3 Sintaxe

```
glm(y ~ x, df, family=gaussian, link=identity)  // OLS via GLM
glm(y ~ x, df, family=binomial, link=logit)     // Logit via GLM
glm(y ~ x, df, family=gamma,    link=log)       // Gamma com link log
glm(y ~ x, df, family=poisson,  link=log)       // Poisson / PPML
```

Família	Links disponíveis
gaussian	identity, log, inverse
binomial	logit, probit, cloglog
poisson	log, identity, sqrt
gamma	log, identity, inverse

Capítulo 43

Modelos de Mudança de Regime

43.1 Heterogeneidade estrutural ao longo do tempo

Muitas séries econômicas exibem mudanças estruturais: diferentes médias, volatilidades ou dinâmicas em distintos períodos. Dois enfoques capturam essa heterogeneidade:

- **Markov Switching** (Hamilton 1989): regime latente segue uma cadeia de Markov. A probabilidade de transição entre regimes é estimada pelo algoritmo EM junto com os parâmetros de cada regime.
- **Threshold Regression** (Hansen 1997): mudança de regime é determinada por uma variável observável (variável limiar q) cruzar um limiar γ estimado.

Markov-Switching AR(p)

$$y_t = \mu_{S_t} + \sum_{k=1}^{p} \phi_k^{(S_t)} \left(y_{t-k} - \mu_{S_t}\right) + \sigma_{S_t}\, \varepsilon_t$$

onde $S_t \in \{1, \ldots, K\}$ é o regime latente com probabilidades de transição $P(S_t = j \mid S_{t-1} = i) = p_{ij}$. O parâmetro μ_{S_t} é o intercepto do regime, $\phi_k^{(S_t)}$ são os coeficientes AR e σ_{S_t} é o desvio-padrão do regime.

43.2 Verificação por DGP

DGP com dois regimes alternantes:

- Regime 1 (expansão): $\mu_1 = 1{,}0$, $\phi_1 = 0{,}30$, $\sigma_1 = 0{,}5$

- Regime 2 (recessão): $\mu_2 = -1{,}0$, $\phi_2 = 0{,}70$, $\sigma_2 = 1{,}5$

```
seed(42)
generate df t      = _n
generate df regime = (t % 60 < 40)   // ~2/3 no regime 1
// y recursivo com parâmetros condicionais ao regime
...
let m_ms = markov(df, y, k=2, p=1)
display m_ms
```

Listing 43.1: DGP Markov Switching AR(1) e estimação

Markov Switching

```
====================== Markov Switching AR(1) - 2 regimes ======================
Observations: 299   Log-L: -155.76   AIC: 327.51   BIC: 357.12
---------------------------- Transition Matrix --------------------------------
[ 0.9493   0.0507 ]
[ 0.0251   0.9749 ]
---------------------------- Regime 0 (Recessão) ------------------------------
Intercept: -1.2864   AR.L1: 0.6169   Variance: 0.5141
---------------------------- Regime 1 (Expansão) ------------------------------
Intercept:  0.9946   AR.L1: 0.2956   Variance: 0.0625
---------------------------- Expected Durations --------------------------------
Regime 0: 19.7 períodos    Regime 1: 39.9 períodos
```

O modelo recupera os dois regimes com boa precisão:

- Regime 0 (recessão): $\hat{\mu} = -1{,}29$ (verdadeiro: $-1{,}0$), $\hat{\phi} = 0{,}62$ (verdadeiro: 0,7). Duração esperada 19,7 períodos.
- Regime 1 (expansão): $\hat{\mu} = 0{,}99 \approx 1{,}0$, $\hat{\phi} = 0{,}30$ (verdadeiro: 0,3). Duração esperada 39,9 períodos.

A matriz de transição indica alta persistência de ambos os regimes ($p_{00} = 0{,}95$, $p_{11} = 0{,}97$) - a série raramente troca de regime, condizente com o DGP (60% do tempo no regime 1).

43.3 Sintaxe

```
markov(df, y, k=2, p=1)     // k regimes, AR de ordem p
```

Opção	Padrão	Descrição
k	2	número de regimes
p	1	ordem AR dentro de cada regime

Nota

O número de regimes k é fixo e deve ser especificado a priori. Critérios de informação (AIC, BIC) comparados entre modelos com $k = 2$ e $k = 3$ guiam essa escolha. Para dados com $N < 200$, $k > 3$ raramente é identificado de forma confiável.

Capítulo 44

Filtros e Decomposição de Séries Temporais

44.1 Separação de componentes

Uma série temporal pode ser decomposta em:

$$y_t = \tau_t + c_t + s_t + \varepsilon_t$$

onde τ_t é a tendência, c_t o componente cíclico, s_t a sazonalidade e ε_t o ruído idiossincrático. Filtros lineares extraem um ou mais desses componentes a partir da série observada.

Filtro Hodrick-Prescott (HP)

O filtro HP (Hodrick e Prescott 1997) separa tendência de ciclo minimizando:

$$\min_{\{\tau_t\}} \left\{ \sum_{t=1}^{T} (y_t - \tau_t)^2 + \lambda \sum_{t=2}^{T-1} [(\tau_{t+1} - \tau_t) - (\tau_t - \tau_{t-1})]^2 \right\}$$

O parâmetro λ penaliza variações na segunda diferença da tendência. Valores canônicos: $\lambda = 1600$ (trimestral), 100 (anual), 14400 (mensal).

Filtro Baxter-King (BK)

O filtro BK (Baxter e King 1999) extrai um componente de *banda de frequência*: retém variações periódicas entre $[p_L, p_H]$ períodos e elimina tendência e frequências altas. Para ciclos de negócios: $p_L = 6$, $p_H = 32$ trimestres (1,5 a 8 anos).

Suavização Exponencial Holt-Winters (ETS)

O modelo ETS (*Error-Trend-Seasonal*) de Holt e Winters combina: equações de nível, tendência e sazonalidade atualizadas recursivamente com pesos exponencialmente decrescentes. Parâmetros (α, β, γ) controlam a velocidade de adaptação de cada componente.

44.2 Verificação por DGP

DGP com tendência linear, ciclo anual e sazonalidade semestral, $T = 120$ meses.

```
seed(42)
generate df t       = _n
generate df trend   = 0.05 * t
generate df cycle   = 0.5 * sin(t * 3.14159 / 12)
generate df season  = 0.3 * sin(t * 3.14159 / 6)
generate df noise   = (rnormal() - 0.5) * 0.3
generate df y       = 10.0 + trend + cycle + season + noise

hprescott(df, y, lambda=1600)
display df

baxter_king(df, y, low=6, high=32, k=12)
display df

let m_hw = holtwinters(df, y, trend=add, seasonal=add, period
   =12)
display m_hw
```

Listing 44.1: Séries com tendência, ciclo e sazonalidade

Hodrick-Prescott

```
hpfilter:  =1600 →   y_trend e y_cycle adicionadas a df
  t=1:  y=10.34  y_trend=10.56  y_cycle=-0.22
  t=2:  y=10.62  y_trend=10.56  y_cycle= 0.06
  ...
```

O HP adiciona colunas y_trend (tendência suavizada) e y_cycle (ciclo = resíduo).

Baxter-King

O BK adiciona y_cycle com o componente de banda de frequência. Os primeiros e últimos $k = 12$ períodos ficam como NaN - custo dos filtros de duas pontas.

Holt-Winters ETS

```
======================== ETS(A,A,A) ========================
Observations:    120
Alpha:         0.8000     (nível)
Beta:          0.0500     (tendência)
Gamma:         0.6000     (sazonalidade)
SSE:             5.3570
AIC:           -339.09
BIC:           -291.70
============================================================
```

O modelo ETS(A,A,A) - erros aditivos, tendência aditiva, sazonalidade aditiva - ajusta os parâmetros de suavização (α, β, γ) minimizando SSE. α alto (0,80) indica resposta rápida do nível.

44.3 Sintaxe

```
hprescott(df, var, lambda=1600)              // adiciona
    var_trend e var_cycle
baxter_king(df, var, low=6, high=32, k=12)  // adiciona
    var_cycle
holtwinters(df, var, trend=add, seasonal=add, period=12)
ets(df, var, trend=add, seasonal=add, period=12)  // alias
```

Filtro	Output	Uso típico
HP	tendência + ciclo	macro trimestral
BK	ciclo de banda	análise de negócios
Holt-Winters	modelo ETS + previsão	previsão de curto prazo

Nota

O filtro HP em dados de ponta (*end-of-sample*) sofre do *end-point problem*: tendência e ciclo são revisados conforme novos dados chegam. O BK evita esse problema ao usar uma janela simétrica (custo: perde k observações em cada extremo).

Capítulo 45

VAR Estrutural

45.1 Da forma reduzida à estrutura

O VAR da forma reduzida (cap. 28) fornece $K^2 \cdot p + K$ parâmetros livres. A impulso-resposta reduzida mede o efeito de *inovações* u_t sem conteúdo econômico. O SVAR recupera *choques estruturais* ε_t (oferta, demanda, política monetária) impondo restrições que refletem teoria econômica:

$$A\, y_t = A_1^*\, y_{t-1} + \cdots + B\, \varepsilon_t \qquad \varepsilon_t \sim (0, I_K)$$

Identificação de Cholesky

A decomposição triangular inferior de $\Sigma_u = P\, P'$ impõe uma ordenação causal recursiva: a variável na posição 1 é exógena contemporaneamente; a na posição 2 responde contemporaneamente apenas à primeira; etc.

$$B = P \quad \text{(triangular inferior)} \implies y_{1,t} \perp \varepsilon_{2,t} \text{ no mesmo período}$$

O IRF estrutural $\Theta_h = \Phi_h B$ mede a resposta de $y_{k,t+h}$ a um choque unitário em $\varepsilon_{j,t}$.

45.2 Verificação por DGP

DGP: VAR(1) bivariado com restrição Cholesky verdadeira (y2 responde contemporane
a y1, mas não vice-versa).

```
seed(42)
```

```
generate df t  = _n
// VAR(1): y1 e y2 com coeficientes cruzados
...
replace df y1 = 0.5*ly1 + 0.3*ly2 + e1t          if t == it
replace df y2 = 0.2*ly1 + 0.4*ly2 + 0.6*e1t + e2t if t == it

let m_var  = var(df, y1, y2, lags=1)
display m_var

let m_svar = svar(df, y1, y2, lags=1, id=cholesky)
display m_svar

let m_sirf = sirf(m_svar, steps=10)
display m_sirf
```

Listing 45.1: DGP estrutural e SVAR Cholesky

VAR reduzido

```
Vector Autoregression (VAR(1)):  Observations=299  AIC=-2.83  BIC=-2.76
Residual Covariance Σ(_u):
[ 0.2452   0.1617 ]
[ 0.1617   0.3375 ]
```

SVAR — Matriz B

```
============================== SVAR(1) - Cholesky ==============================
B Matrix (impacto estrutural):
[ 0.4952   0.0000 ]     <- choque 1 afeta y1, não afeta y2 contemporaneamente
[ 0.3265   0.4805 ]     <- choque 2 afeta y2; y2 responde ao choque 1
```

A coluna nula em B_{12} reflete a ordenação Cholesky: o choque 2 não afeta y_1 contemporaneamente. $B_{21} = 0{,}33$ estima o efeito contemporâneo do choque 1 sobre y_2 (DGP verdadeiro: $0{,}6 \times \sigma_{\varepsilon_1}$).

IRF estrutural

```
Impulso: Choque 1 (Var0)
  h=0:  y1=+0.495  y2=+0.327   (impacto contemporâneo)
  h=1:  y1=+0.324  y2=+0.224
```

```
  h=5:   y1=+0.063   y2=+0.044   (amortecimento progressivo)
  h=9:   y1=+0.012   y2=+0.009

Impulso: Choque 2 (Var1)
  h=0:   y1= 0.000   y2=+0.481   (restrição Cholesky: y1 não responde)
  h=1:   y1=+0.115   y2=+0.170
```

A resposta de y_1 ao choque 2 em $h = 0$ é exatamente zero - restrição de identificação Cholesky aplicada. No horizonte $h = 1$, y_1 começa a responder via o canal VAR defasado.

45.3 Sintaxe

```
svar(df, y1, y2, lags=1, id=cholesky)      // SVAR com identificação Cholesky
svar(df, y1, y2, lags=1, id=longrun)       // BQ: restrições de longo prazo
sirf(m_svar, steps=10)                     // IRF estrutural
```

Identificação	Restrições
cholesky	triangular inferior - restrições de curto prazo recursivas
longrun	Blanchard-Quah - restrições de longo prazo

Nota

A ordenação das variáveis no SVAR Cholesky é crucial: a variável listada primeiro é tratada como contemporaneamente exógena a todas as demais. A escolha da ordenação deve ser guiada por teoria econômica ou por resultados robustos a permutações da ordem.

Parte VII

Métodos Complementares

Capítulo 46

Inferência Clássica

46.1 Testes de hipóteses paramétricos e não-paramétricos

Os testes apresentados neste capítulo respondem à pergunta: *os dados são consistentes com uma hipótese sobre a distribuição populacional?*

Teste t de Student

Para uma amostra de tamanho n com variância desconhecida, a estatística:

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}} \sim t_{n-1}$$

testa $H_0 : \mu = \mu_0$. Para duas amostras independentes, o teste de Welch usa graus de liberdade aproximados pelo método de Satterthwaite, sem assumir homoscedasticidade.

ANOVA de um fator

Generaliza o t-test para $K > 2$ grupos. Decompõe a variação total:

$$\mathrm{SS}_{\mathrm{total}} = \mathrm{SS}_{\mathrm{entre}} + \mathrm{SS}_{\mathrm{dentro}}$$

$$F = \frac{\mathrm{SS}_{\mathrm{entre}}/(K-1)}{\mathrm{SS}_{\mathrm{dentro}}/(N-K)} \sim F_{K-1,\,N-K}$$

H_0: todas as médias grupais são iguais.

Kruskal-Wallis

Alternativa não-paramétrica à ANOVA de um fator. Substitui observações por *ranks* e compara as somas de ranks entre grupos. Estatística:

$$H = \frac{12}{N(N+1)} \sum_{k=1}^{K} \frac{R_k^2}{n_k} - 3(N+1) \sim \chi^2_{K-1}$$

válida mesmo sem normalidade, desde que as distribuições tenham forma similar.

46.2 Verificação por DGP

DGP com 3 grupos de tamanhos distintos; o valor esperado cresce linearmente com o grupo. $N = 300$, $\sigma = 1$.

```
seed(42)
...
generate df g   = (_n <= 150) * 1 + (_n > 150) * 2
generate df yg  = 2.0 + e + (g - 1) * 0.5  // g=1: mu=2.0, g
    =2: mu=2.5

// Teste t de uma amostra: H0: mu = 2.0
ttest(df, y, mu=2.0)

// Teste t de dois grupos (Welch): H0: mu(g1) = mu(g2)
ttest(df, yg, by=g)

// Pareado: antes vs depois
ttest(df, y_antes, y_depois, paired=true)

// ANOVA: H0: mu1 = mu2 = mu3
anova(df, y3, by=k)

// Kruskal-Wallis (não-paramétrico)
kruskal_wallis(df, y3, by=k)
```

Listing 46.1: Testes de hipóteses

Teste t — uma amostra

```
One-sample t-test: y   H0: mean = 2.0
```

```
t = -0.6946   df = 299   p = 0.4879
```

Não rejeita H_0 - média amostral 1,98 é compatível com 2,0.

Teste t — dois grupos (Welch)

```
Two-sample t-test (Welch): yg by g
  grupo 1: n=150  mean=1.971  SE=0.040
  grupo 2: n=150  mean=2.489  SE=0.041
Welch's t = -9.0552   df = 297.38   p = 0.0000
```

Rejeita H_0: diferença de 0,52 diferença verdadeira de 0,50.

ANOVA

```
============================ One-Way ANOVA
   ============================
Source            SS      df       MS          F        P>F
------------------------------------------------------------------------------

Between        51.195       2   25.597   104.037   0.0000
Within         73.074     297    0.246
Total         124.269     299
```

$F = 104{,}0$, $p < 0{,}001$: rejeita H_0 com segurança.

Kruskal-Wallis

```
H = 120.38   df = 2   p = 0.0000
```

Confirma a ANOVA sem assumir normalidade.

46.3 Sintaxe

```
ttest(df, var, mu=0.0)                    // uma amostra
ttest(df, var, by=grupo)                    // dois grupos (
   Welch)
ttest(df, var, by=grupo, unequal=false)  // dois grupos (
   Student)
ttest(df, var1, var2, paired=true)        // pareado
anova(df, var, by=grupo)                  // ANOVA de um fator
kruskal_wallis(df, var, by=grupo)         // Kruskal-Wallis
```

Nota

O teste de Welch não assume homocedasticidade e é preferível ao t-test clássico de Student em quase todos os casos práticos. A perda de potência é negligenciável quando as variâncias são iguais; o ganho quando diferem é substancial. Caso deseje executar o teste de Student clássico, use a opção unequal=false.

Capítulo 47

Painel Avançado II

47.1 Além do FE e RE básicos

Os estimadores do cap. 37 (AB-GMM, PCSE, GEE) lidam com dinâmica e heterocedastic
Aqui cobrimos quatro abordagens complementares:

- **FE** - referência; efeitos fixos padrão (within estimator)
- **Mundlak (CRE)** - Correlated Random Effects; testa RE vs FE via γ
- **xtgls** - GLS para painel; flexível quanto à estrutura de Σ
- **mixedlm** - modelo misto com intercepts aleatórios
- **clogit** - logit condicional para variáveis binárias em painel

Estimador de Mundlak (CRE)

Mundlak (1978) mostra que o estimador RE é equivalente ao FE se:

$$\alpha_i = \gamma' \bar{x}_i + u_i \quad u_i \perp x_{it}$$

Regride y_{it} em x_{it} *e* $\bar{x}_i$ (média por entidade). Se $\hat{\gamma} \neq 0$, o efeito fixo é necessário.

GLS para painel (xtgls)

O GLS de painel especifica uma estrutura para a covariância dos erros entre entidades. Com panels=hetero, cada painel tem variância própria σ_i^2, permitindo heterocedasticidade entre painéis.

Modelo misto (mixedlm)

O modelo linear misto especifica:

$$y_{it} = \beta x_{it} + u_i + \varepsilon_{it} \quad u_i \sim N(0, \sigma_u^2)$$

onde u_i é um efeito aleatório de intercept por entidade. Difere do RE padrão por usar REML e integrar sobre u_i via máxima verossimilhança.

47.2 Verificação por DGP

DGP: painel estático endógeno. $y = 1{+}2x{+}\alpha_i{+}\varepsilon$; $\alpha_i = 0{,}05{\cdot}i$ correlacionado com x (via z). $N = 50$ entidades, $T = 20$ períodos.

```
seed(42)
generate df unit  = (_n - 1) % 50 + 1
generate df t     = (_n - (_n - 1) % 50 - 1) / 50 + 1
generate df alpha = unit * 0.05
generate df z     = (rnormal() - 0.5) * 1.7321
generate df x     = 0.5 * z + alpha + (rnormal() - 0.5)
generate df y     = 1.0 + 2.0 * x + alpha + e

let m_fe   = fe(y ~ x, df, id=unit)
let m_mund = mundlak(df, y ~ x, id="unit")
let m_gls  = xtgls(y ~ x, df, id=unit, time=t)
let m_mix  = mixedlm(y ~ x, df, id="unit")
generate df yb = (y > 2.5)
let m_cl   = clogit(yb ~ x, df, group="unit")
```

Listing 47.1: Painel II: FE, Mundlak, xtgls, mixedlm, clogit

```
FE (within):        x = 2.041 ***    (verdadeiro: 2.0)
Mundlak:            x = 2.040,   (x) = 0.939 *** → rejeita RE
xtgls (hetero):     x = 2.768 ***    (tendencioso sem
   demeaning)
mixedlm:            x = 2.134 ***    (intercept aleatório:
   ²_u = 0.388)
clogit (yb ~ x):    x = 1.397 ***    (grupos sem variação
   descartados)
```

Teste de Mundlak

O teste $H_0 : \gamma = 0$ (médias não correlacionadas com x) rejeita com $F = 419{,}96$, $p < 0{,}001$. Isso confirma que x é endógeno - o estimador RE seria inconsistente e deve-se usar FE.

O xtgls sem remoção de FE superestima β pois captura parte do efeito individual. O mixedlm fica próximo do FE porque trata α_i como efeito aleatório normal.

47.3 Sintaxe

```
fe(y ~ x, df, id=unit)                          // Fixed Effects (within)
mundlak(df, y ~ x, id="unit")                   // CRE: inclui médias de grupo
xtgls(y ~ x, df, id=unit, time=t)               // GLS para painel
xtgls(y ~ x, df, id=unit, time=t, panels="corr")  // Σ correlacionada
mixedlm(y ~ x, df, id="unit")                   // modelo misto (REML)
clogit(yb ~ x, df, group="unit")                // logit condicional
```

Nota

No clogit, grupos onde y não varia (todos 0 ou todos 1) são automaticamente descartados porque não contribuem para a verossimilhança condicional. Isso reduz a amostra efetiva, mas é teoricamente correto.

Capítulo 48

Raízes Unitárias II

48.1 Além do ADF: PP e Zivot-Andrews

O capítulo anterior (cap. 27) cobriu o teste ADF, que corrige para autocorrelação residual por meio de defasagens. Dois problemas permanecem:

1. O ADF é sensível à heterocedasticidade condicional dos erros (GARCH etc.)
2. O ADF tem baixo poder na presença de *quebra estrutural*: uma série estacionária com quebra de nível se parece com um passeio aleatório.

Teste Phillips-Perron (PP)

Phillips e Perron (1988) adotam uma abordagem não-paramétrica: corrigem a estatística ADF para heterocedasticidade e autocorrelação de *qualquer forma* via estimador de Newey-West:

$$Z_\alpha = T(\hat{\rho} - 1) - \frac{T^2 \hat{\sigma}^2}{2\hat{s}_T^2}(\hat{\lambda}^2 - \hat{\gamma}_0)$$

onde $\hat{s}_T^2$ é a variância de longo prazo estimada com kernel de Bartlett. A distribuição assintótica é idêntica à do ADF (Dickey-Fuller).

Teste Zivot-Andrews (ZA)

Zivot e Andrews (1992) relaxam a hipótese de ausência de quebra estrutural. O teste estima a data de quebra *endogenamente* como o valor que minimiza a estatística t:

$$\hat{t}_B = \arg\min_{\lambda} \; t_\alpha(\lambda), \quad \lambda = t_B/T \in (0.15, 0.85)$$

A equação testada inclui um nível de degrau ou mudança de tendência:

$$\Delta y_t = c + \beta t + \gamma DU_t(\lambda) + \alpha y_{t-1} + \sum_{j=1}^{k} d_j \Delta y_{t-j} + \varepsilon_t$$

H_0: raiz unitária sem quebra estrutural.

48.2 Verificação por DGP

```
seed(42)
// Passeio aleatório: rw_t = rw_{t-1} + e_t
// Série estacionária com tendência: stat_t = 0.5*t + e
// Série com quebra em t=150: AR(1) + degrau de 3.0

phillips_perron(df, rw)
phillips_perron(df, stat)
zivot(df, break_t)
```

Listing 48.1: PP e Zivot-Andrews

```
PP - rw:      Zt = 77.365   VC 5%=-2.860   NÃO rejeita H  (
   raiz unitária)
PP - stat:    Zt = 2058     VC 5%=-2.860   NÃO rejeita H  (
   trend-stationary)
ZA - break_t: t  = -11.597  VC 5%=-4.800   REJEITA H, quebra
   em obs=147
```

O PP para stat (tendência linear + ruído) não rejeita H_0 porque o teste padrão (sem tendência determinística na equação auxiliar) não distingue tendência estocástica de determinística. O ZA identifica corretamente a quebra estrutural em $t = 147$ (próximo ao verdadeiro $t = 150$) e rejeita H_0.

48.3 Sintaxe

```
phillips_perron(df, var)          // PP sem tendência
phillips_perron(df, var, trend=true)  // PP com tendência
   determinística
```

```
zivot(df, var)                    // Zivot-Andrews (quebra endógena de nível)
zivot(df, var, type="trend")      // quebra na tendência
```

Teste	Correção	Quebra?	VC 5%
ADF	Defasagens paramétricas	Não	$-2{,}86$
PP	Newey-West não-param.	Não	$-2{,}86$
ZA	PP + quebra endógena	Sim	$-4{,}80$

Nota

Séries trend-stationary e séries com quebra estrutural são dois casos onde o ADF/PP pode levar a conclusões erradas. O ZA resolve o segundo; para o primeiro, incluir uma tendência determinística na equação auxiliar (trend=true) é suficiente.

Capítulo 49

Decomposição de Séries II

49.1 STL e Christiano-Fitzgerald

O cap. 44 apresentou HP, BK e Holt-Winters. Este capítulo cobre dois métodos adicionais:

STL *Seasonal-Trend decomposition via Loess* (Cleveland et al. 1990): decompõe $y_t = T_t + S_t + R_t$ iterativamente via regressão local (loess). Robusto a outliers; permite sazonalidade variante no tempo.

Christiano-Fitzgerald (CF) Filtro passa-banda assimétrico (2003): extrai frequências em $[p_L, p_H]$ usando pesos de uma janela não-simétrica que se adapta à posição na série. Ao contrário do BK, não perde pontos de borda - o preço é que os pesos mudam a cada observação.

STL: decomposição iterativa

O algoritmo STL alterna entre:

1. *Seasonal loop*: suaviza cada posição sazonal com loess
2. *Trend loop*: suaviza a série dessazonalizada com loess

A convergência produz T_t (tendência), S_t (sazonalidade) e R_t (resíduo) com propriedades de robustez a outliers quando robust=true.

Filtro Christiano-Fitzgerald

O CF aplica pesos a_j que satisfazem $\sum_j a_j e^{i\omega j} \approx 1$ para $\omega \in [\omega_L, \omega_H]$ e ≈ 0 fora da banda. Para t nas bordas, os pesos se ajustam para não usar observações futuras.

49.2 Verificação por DGP

DGP: série mensal com tendência linear, ciclo anual e sazonalidade semestral. $T = 120$ meses.

```
seed(42)
generate df trend  = 0.05 * t
generate df cycle  = 0.5 * sin(t * 3.14159 / 12)
generate df season = 0.3 * sin(t * 3.14159 / 6)
generate df y      = 10.0 + trend + cycle + season + noise

stl(df, y, period=12)
display df

christiano_fitzgerald(df, y, low=6, high=32)
display df
```

Listing 49.1: STL e Christiano-Fitzgerald

STL

```
Seasonal Decomposition (STL)
  Observations: 120
  Trend mean:   13.0243
  Seasonal std:  0.1943
  Residual std:  0.1868
```

O STL adiciona colunas y_trend, y_seasonal e y_resid ao DataFrame. Residual std $< 0{,}2$ - o ruído verdadeiro ($\sigma = 0{,}3$) é bem separado dos demais componentes.

Christiano-Fitzgerald

```
cffilter: períodos [6,32] →   y_cycle adicionada a df
  t=1:  y_cycle = 1.04
  t=6:  y_cycle = 1.19    (ciclo anual dominante)
  t=12: y_cycle = 0.87
```

O CF retém variações periódicas entre 6 e 32 meses (1,5 a 8 anos - ciclos de negócios). A vantagem sobre o BK é que todas as 120 observações são mantidas (sem trimagem de pontas).

49.3 Sintaxe

```
stl(df, var, period=12)                          // adiciona trend, seasonal, resid
christiano_fitzgerald(df, var, low=6, high=32) // adiciona var_cycle
```

Filtro	Tipo	Perda de borda	Sazonalidade
HP	passa-baixo	Sim (ponta)	Não
BK	passa-banda	Sim (k cada lado)	Não
CF	passa-banda	Não	Não
STL	decomposição	Não	Sim

Nota

A função hamilton no Hayashi refere-se ao modelo de Markov Switching de Hamilton (1989), *não* ao filtro de Hamilton (2018) (cap. 43). O filtro Hamilton 2018 - que propõe substituir o HP por uma regressão OLS com 4 defasagens e horizonte $h = 8$ - não está implementado separadamente.

Capítulo 50

SVAR com Restrições de Longo Prazo

50.1 Identificação de Blanchard-Quah

O cap. 45 (SVAR Cholesky) impõe restrições de *curto prazo*: a variável na posição 1 não é afetada contemporaneamente pela segunda. Blanchard e Quah (1989) propõem uma alternativa via restrições de *longo prazo*: choques permanentes e transitórios.

A ideia central: na macroeconomia, um choque de oferta tem efeito permanente sobre o produto (y), enquanto um choque de demanda tem efeito apenas transitório. A restrição:

$$\sum_{h=0}^{\infty} \Theta_h^{(1,1)} = \text{livre}, \quad \sum_{h=0}^{\infty} \Theta_h^{(1,2)} = 0$$

impõe que o choque 2 não afeta y no longo prazo. Essa restrição identifica a decomposição B sem nenhuma restrição de impacto contemporâneo.

Decomposição BQ

Dado o VAR na forma reduzida $y_t = \Phi_1 y_{t-1} + u_t$:

1. Compute a soma $C = (I - \Phi_1)^{-1}$ - multiplicador de longo prazo

2. Fatore $C\,\Sigma_u\,C' = F\,F'$ (Cholesky de F)

3. Extraia $B = C^{-1}F$ - os choques estruturais de longo prazo

50.2 Verificação por DGP

DGP: VAR(1) bivariado onde y_1 é um passeio aleatório (choque 1 permanente), y_2 é estacionária e responde transitoriamente ao choque 1.

```
seed(42)
// y1_t = y1_{t-1} + e1_t                (passeio aleatório)
// y2_t = 0.5*y2_{t-1} + 0.4*e1_t + 0.6*e2_t  (estacionária)

let m_bq = svar(df, y1, y2, lags=1, id=longrun)
display m_bq
let m_irf_bq = sirf(m_bq, steps=20)
display m_irf_bq
```

Listing 50.1: SVAR Blanchard-Quah

Matriz B (choques estruturais)

```
SVAR(1) - Long-run restrictions
B Matrix:
[ 0.4907   0.0358 ]
[ 0.1935   0.3031 ]
```

A matriz B não é triangular - ao contrário do Cholesky, as restrições de longo prazo não implicam restrições de impacto contemporâneo.

IRF estrutural

```
Impulso: Choque 1 (permanente)
  h= 0:  y1=+0.491  y2=+0.194
  h= 1:  y1=+0.464  y2=+0.087
  h= 5:  y1=+0.405  y2=-0.003   (efeito em y2 decai)
  h=19:  y1=+0.271  y2=-0.005   (y1 retém efeito permanente)

Impulso: Choque 2 (transitório)
  h= 0:  y1=+0.036  y2=+0.303
  h= 1:  y1=+0.016  y2=+0.144
  h= 5:  y1=-0.001  y2=+0.007   (efeito em y1 decai a ~0)
  h=19:  y1=-0.001  y2= 0.000   (transitório confirmado)
```

O choque 2 afeta y_1 apenas transitoriamente (soma → 0), confirmando a restrição de longo prazo. O choque 1 tem efeito acumulado crescente sobre y_1 - coerente com a raiz unitária.

50.3 Sintaxe

```
svar(df, y1, y2, lags=1, id=longrun)     // BQ: restrições de longo prazo
svar(df, y1, y2, lags=1, id=cholesky)    // Cholesky: restrições contemporâneas
sirf(m_svar, steps=20)                   // IRF estrutural
```

Nota

A identificação BQ requer que as variáveis sejam integradas de ordem 1 ($I(1)$). Para séries estacionárias, $(I - \hat{\Phi}_1)^{-1}$ é finito e a decomposição permanente/transitória não tem interpretação natural. Verificar raiz unitária (cap. 27, 48) antes de aplicar BQ.

Capítulo 51

Estimação Local e Ponderada

51.1 Pesos, janelas e suavização

A estimação clássica (OLS, MLE) trata cada observação simetricamente. Três alternativas permitem que o modelo se adapte localmente:

WLS Mínimos quadrados ponderados: observações com menor variância (ou maior relevância) recebem maior peso.

Rolling OLS Estima o modelo em janelas temporais deslizantes, revelando como os coeficientes evoluem ao longo do tempo.

LOWESS Suavização não-paramétrica: para cada ponto x_0, ajusta uma regressão ponderada localmente usando vizinhos próximos.

WLS (Weighted Least Squares)

O estimador WLS minimiza:

$$\hat{\beta}_{WLS} = \arg\min_{\beta} \sum_{i=1}^{n} w_i (y_i - x_i'\beta)^2 = (X'WX)^{-1}X'Wy$$

onde $W = \text{diag}(w_1, \ldots, w_n)$. Se os pesos são proporcionais a $1/\sigma_i^2$ (variâncias dos erros), o WLS é o estimador eficiente sob heterocedasticidade (= GLS para erros independentes).

Rolling OLS

Para séries temporais não-estacionárias nos parâmetros (coeficientes variantes no tempo), o rolling OLS com janela h estima:

$$\hat{\beta}_t = (X'_{t-h:t} X_{t-h:t})^{-1} X'_{t-h:t} Y_{t-h:t}$$

LOWESS

O LOWESS (Cleveland 1979) estima $m(x_0) = E[y|x = x_0]$ localmente:

1. Seleciona os $f \cdot n$ vizinhos mais próximos de x_0
2. Aplica pesos tricúbicos: $w_i = (1 - |d_i|^3)^3$
3. Ajusta uma regressão ponderada (linear ou constante)
4. Itera com re-ponderação robusta (opcional)

51.2 Verificação por DGP

DGP: erro heterocedástico $\varepsilon_i \propto (1 + x_i)$; coeficiente de t pequeno (0,01). $N = 300$ observações.

```
seed(42)
generate df w = exp(-0.002 * t)        // peso: observações recentes mais relevantes
generate df e = (rnormal()-0.5)*2*(1+x)  // heterocedasticidade em x
generate df y = 1.0 + 2.0*x + 0.01*t + e

let m_ols = ols(y ~ x, df)
let m_wls = wls(y ~ x, df, weights="w")
let m_rol = rols(y ~ x + t, df, window=50)
lowess(df, y, x, frac=0.3)
```

Listing 51.1: WLS, Rolling OLS, LOWESS

```
OLS:  const=2.362, x=2.263***  (ignora heterocedasticidade)
WLS:  const=2.212, x=2.257***  (pesos corrigem variância crescente)
Rolling OLS (última janela t=251..300):
  const=1.898, x=2.411, t=0.006
```

```
LOWESS: frac=0.30, 300 obs →   y_hat_lowess adicionado a df
```

O WLS altera pouco os coeficientes aqui – os pesos temporais compensam o crescimento do erro em x de forma parcial. O rolling OLS revela que o coeficiente de x na última janela (2,41) é ligeiramente maior, sugerindo não-estacionariedade leve.

51.3 Sintaxe

```
wls(y ~ x, df, weights="coluna_pesos")        // WLS
rols(y ~ x + t, df, window=50)                // Rolling OLS
rolling(y ~ x + t, df, window=50)             // alias
lowess(df, y_var, x_var, frac=0.67, it=3)    // LOWESS
```

Nota

O LOWESS recebe como argumento a variável dependente **antes** da independente: lowess(df, y, x). Isso difere de regressões convencionais. O parâmetro frac (fração de vizinhos usados em cada ponto) controla a suavização: valores menores → curva mais nervosa; maiores → mais suave.

Capítulo 52

AutoReg e ARDL

52.1 Modelos com defasagens em y e em x

O modelo ARIMA (cap. 27) trata apenas da dinâmica de y. Em economia, porém, o nível atual de y depende não só do próprio passado, mas também de valores correntes e passados de regressores x. O modelo *Autoregressive Distributed Lag* (ARDL) captura exatamente isso.

AutoReg(p): AR puro via OLS

O modelo AR(p) escreve y_t como função linear das próprias defasagens:

$$y_t = c + \sum_{i=1}^{p} \phi_i \, y_{t-i} + \varepsilon_t$$

Ao contrário do ARIMA (estimado por MLE), o autoreg() estima via OLS. Isso é computacionalmente mais simples e assintoticamente equivalente para processos estacionários. A opção trend permite incluir constante e/ou tendência determiníst

trend	Componente determinístico
"c"	Constante (padrão)
"ct"	Constante + tendência linear
"t"	Apenas tendência
"n"	Nenhum

ARDL(p, q): defasagens distribuídas

O modelo ARDL(p, q) combina p defasagens de y com o valor contemporâneo e q defasagens de cada regressor x:

$$y_t = c + \sum_{i=1}^{p} \phi_i \, y_{t-i} + \sum_{j=0}^{q} \beta_j \, x_{t-j} + \varepsilon_t$$

O **multiplicador de longo prazo** de x sobre y - o efeito total acumulado após todos os ajustes dinâmicos - é:

$$\theta = \frac{\sum_{j=0}^{q} \beta_j}{1 - \sum_{i=1}^{p} \phi_i}$$

Para o modelo ser estacionário, exige-se $\left|\sum_{i=1}^{p} \phi_i\right| < 1$.

Relação com o VECM e o teste de Pesaran

Quando y e x são integrados de ordem 1 ($I(1)$), o ARDL pode ser reescrito numa forma de correção de erros. Pesaran, Shin e Smith (2001) propõem o *bounds test*: testa H_0 de ausência de relação de longo prazo via estatística F aplicada ao ARDL irrestrito, comparando com valores críticos tabelados. A vantagem sobre o Johansen (cap. 29) é que o bounds test funciona independentemente de x ser $I(0)$ ou $I(1)$.

52.2 Verificação por DGP

DGP: $y_t = 1 + 2x_t + \varepsilon_t$, sem autocorrelação em y. $N = 300$, $x \sim U(-\sqrt{3}, \sqrt{3})$, $\varepsilon \sim U(-0{,}5, 0{,}5)$.

```
seed(42)
// ... (300 obs, geração de x e e)
generate df y = 1.0 + 2.0 * x + 0.5 * (rnormal() - 0.5)

// AR(2) puro: nenhuma autocorrelação real em y → coefs ~0
let m_ar   = autoreg(df, y, lags=2)
display m_ar

// AR(1) com tendência determinística
let m_ar_ct = autoreg(df, y, lags=1, trend="ct")
display m_ar_ct

```

```
// ARDL(1,1)
let m_ardl  = ardl(y ~ x, df, lags=1, xlags=1)
display m_ardl

// ARDL(2,2): verifica se defasagens extras são
   significativas
let m_ardl2 = ardl(y ~ x, df, lags=2, xlags=2)
display m_ardl2
```

Listing 52.1: AutoReg e ARDL

AR(2) puro

```
================================== AutoReg(2)
   ==================================
Observations:                298
R-squared:                0.0004
AIC:                    905.2221

--------------------------------- Coefficients
   ---------------------------------
                          coef      std err            t
                                    P>|t|
------------------------------------------------------------------------------

const                  0.9904       0.1022       9.6914
   0.0000
y.L1                   0.0007       0.0580       0.0118
   0.9906
y.L2                  -0.0209       0.0581      -0.3597
   0.7193
==============================================================================
```

$R^2 = 0{,}0004$ - as defasagens de y não explicam nada, conforme o DGP (sem autocorrelação verdadeira). Os coeficientes y.L1 e y.L2 não diferem de zero.

AR(1) com tendência

```
================================= AutoReg(1)
   ================================
Trend:                         ct

                         coef      std err            t
                                   P>|t|
------------------------------------------------------------------------------

const                  0.9275       0.1394       6.6551
   0.0000
trend                  0.0003       0.0007       0.4172
   0.6768
y.L1                   0.0018       0.0581       0.0310
   0.9753
==============================================================================
```

A tendência determinística ($p = 0{,}68$) e o lag ($p = 0{,}98$) não são significativos - coerente com um DGP sem tendência nem persistência.

ARDL(1,1)

```
================================= ARDL(1, 1)
   ================================
Observations:               299
R-squared:                0.9836
AIC:                   -318.6638

                         coef      std err            t
                                   P>|t|
------------------------------------------------------------------------------

const                  1.0018       0.0594      16.8580
   0.0000
y.L1                   0.0062       0.0583       0.1061
   0.9155
x1                     1.9648       0.0148     133.1872
   0.0000
x1.L1                 -0.0078       0.1156      -0.0675
   0.9462
```

```
==============================================================================
```

O modelo recupera o DGP com precisão: const $\approx 1{,}0$, x1 $\approx 2{,}0$. O coeficiente de x1.L1 não é significativo ($p = 0{,}95$), confirmando que não há efeito defasado de x no DGP. O multiplicador de longo prazo é:

$$\theta = \frac{1{,}9648 + (-0{,}0078)}{1 - 0{,}0062} \approx 1{,}97$$

próximo ao valor verdadeiro de 2,0.

ARDL(2,2)

```
================================== ARDL(2, 2)
   ==================================
Observations:                298
R-squared:                0.9838
AIC:                   -317.3857

                          coef      std err            t
                                    P>|t|
------------------------------------------------------------------------------

const                   1.0727       0.0833      12.8843
   0.0000
y.L1                    0.0013       0.0582       0.0220
   0.9824
y.L2                   -0.0662       0.0582      -1.1385
   0.2558
x1                      1.9628       0.0147     133.1320
   0.0000
x1.L1                   0.0018       0.1154       0.0154
   0.9877
x1.L2                   0.1264       0.1153       1.0968
   0.2736
==============================================================================
```

O AIC do ARDL(2,2) ($-317{,}4$) é ligeiramente *maior* (pior) do que o do ARDL(1,1) ($-318{,}7$) - as defasagens extras não compensam a perda de graus de liberdade. A seleção por AIC/BIC favorece o modelo mais parcimonioso.

52.3 Sintaxe

```
// AR(p) via OLS
autoreg(df, var, lags=1)                          // AR(1) com constante
autoreg(df, var, lags=p, trend="ct")              // AR(p) com tendência

// ARDL(p,q)
ardl(y ~ x, df, lags=p, xlags=q)                  // um regressor
ardl(gdp ~ inv + exp, df, lags=2, xlags=2)        // múltiplos regressores
```

Modelo	Comando	Caso de uso
AR(p)	autoreg(df, y, lags=p)	Dinâmica pura de y
ARIMA(p,d,q)	arima(df, y, p=, d=, q=)	$I(1)$ com MA
ARDL(p,q)	ardl(y~x, df, lags=p, xlags=q)	y e x com defasagens
VECM	vecm(...)	Cointegração multivariada

Nota

Para séries $I(1)$, o ARDL pode ser reescrito numa forma de correção de erros. O bounds test de Pesaran et al. (2001) testa a existência de relação de longo prazo sem exigir que os regressores sejam todos $I(0)$ ou todos $I(1)$ - vantagem sobre o teste de Johansen. A estatística F do teste é acessível via m_ardl.bounds_test(y, x) na API Rust do Greeners, mas ainda não está exposta como comando do DSL.

Capítulo 53

Filtro de Kalman

53.1 Estado oculto e observação ruidosa

O filtro de Kalman resolve um problema diferente de todos os modelos anteriores: estimar uma variável *latente* (o ``estado'') que não é diretamente observada, com base em medições ruidosas ao longo do tempo.

O modelo de espaço de estados em forma geral é:

$$y_t = H\, s_t + \varepsilon_t, \quad \varepsilon_t \sim N(0, R) \tag{53.1}$$

$$s_t = F\, s_{t-1} + R_s\, \eta_t, \quad \eta_t \sim N(0, Q) \tag{53.2}$$

A equação (53.1) é a **equação de observação**: y_t é o que se mede, s_t é o estado latente e ε_t é o ruído de medição. A equação (53.2) é a **equação de transição**: o estado evolui de acordo com uma dinâmica linear perturbada por η_t.

Modelos pré-definidos

O Hayashi oferece dois modelos prontos para séries escalares:

Local Level (LL):

$$y_t = \mu_t + \varepsilon_t, \quad \mu_t = \mu_{t-1} + \eta_t$$

O estado μ_t é um passeio aleatório - o nível deriva ao longo do tempo. σ_ε controla o ruído de observação; σ_η controla a velocidade com que o nível muda.

Local Linear Trend (LLT):

$$y_t = \mu_t + \varepsilon_t, \quad \mu_t = \mu_{t-1} + \nu_{t-1} + \eta_t, \quad \nu_t = \nu_{t-1} + \zeta_t$$

O estado é bidimensional: nível μ_t e inclinação ν_t. A inclinação varia suavemente ao longo do tempo.

Filtro de Kalman (passo forward)

O filtro processa as observações recursivamente:

1. **Predição**: $\hat{s}_{t|t-1} = F\,\hat{s}_{t-1|t-1}$, $\quad P_{t|t-1} = FP_{t-1|t-1}F' + R_sQR_s'$
2. **Inovação**: $v_t = y_t - H\,\hat{s}_{t|t-1}$, $\quad S_t = HP_{t|t-1}H' + R$
3. **Ganho de Kalman**: $K_t = P_{t|t-1}H'S_t^{-1}$
4. **Atualização**: $\hat{s}_{t|t} = \hat{s}_{t|t-1} + K_tv_t$, $\quad P_{t|t} = (I - K_tH)P_{t|t-1}$

O resultado $\hat{s}_{t|t}$ é o ***estado filtrado***: estimativa ótima do estado usando $y_1, \ldots, y_t$ (informação até t).

Suavizador de Rauch-Tung-Striebel (passo backward)

Após o filtro, o RTS percorre a série ao contrário e incorpora informação futura:

$$L_t = P_{t|t}\,F'\,P_{t+1|t}^{-1}, \quad \hat{s}_{t|T} = \hat{s}_{t|t} + L_t\left(\hat{s}_{t+1|T} - \hat{s}_{t+1|t}\right)$$

O resultado $\hat{s}_{t|T}$ é o ***estado suavizado***: usa ***toda*** a amostra $y_1, \ldots, y_T$. A variância do suavizado é sempre menor ou igual à do filtrado - a informação futura reduz incerteza.

53.2 Verificação por DGP

DGP: nível verdadeiro $\mu_t = 10{+}0{,}1\,t$ (tendência linear) observado com ruído $\varepsilon_t \sim N(0,1)$. $T = 150$.

```
seed(42)
generate df mu = 10.0 + 0.1 * t
generate df y  = mu + noise        // noise ~ N(0,1)

// Local Level (sigma especificado)
kalman(df, y, model="ll", sigma_obs=1.0, sigma_state=0.1)

// Local Linear Trend (sigmas automáticos)
kalman(df, y, model="llt")
```

Listing 53.1: Filtro de Kalman: LL e LLT

Saída — Local Level

```
Kalman (ll):  T=150  loglik=-278.1966  _obs=1.0000  _state
   =0.1000→
   y_filtered, y_smoothed adicionadas a df
```

Saída — Local Linear Trend

```
Kalman (llt):  T=150  loglik=-219.2354  _obs=0.9663  _state
   =0.0966  _slope=0.0097→
   y_filtered, y_smoothed, y_slope_filtered, y_slope_smoothed
       adicionadas a df
```

O loglik do LLT ($-219{,}2$) é substancialmente maior do que o do LL ($-278{,}2$) porque o DGP tem tendência linear - o modelo com inclinação livre captura melhor a estrutura dos dados.

Comparação com o estado verdadeiro

```
Primeiras 8 observações (LLT):

  t    mu     y       y_smoothed   y_slope_smoothed
  1   10.10  10.19      9.74           0.13
  2   10.20  10.35      9.87           0.13
  3   10.30  10.77      9.99           0.13
  4   10.40  10.07     10.10           0.13
  5   10.50   8.89     10.21           0.13
  6   10.60  10.31     10.33           0.12
  7   10.70  11.52     10.46           0.12
  8   10.80  12.01     10.56           0.12

summarize y_smoothed:  mean=17.49  sd=4.35
summarize mu:          mean=17.55  sd=4.34
```

O suavizado recupera o nível verdadeiro com precisão notável: média 17,49 vs 17,55. A inclinação suavizada (y_slope_smoothed $\approx 0{,}13$) é próxima

ao valor verdadeiro de $0{,}1$. A série bruta y oscila em ± 2 ao redor de μ; o suavizado elimina esse ruído.

Nota

Filtrado vs suavizado: y_filtered usa apenas $y_1, \ldots, y_t$ – é causal e útil para previsão em tempo real. y_smoothed usa $y_1, \ldots, y_T$ inteiro – é retrospectivo, mas ótimo para análise histórica. Nos pontos interiores, o suavizado é sempre mais preciso; no último ponto $t = T$, os dois coincidem.

53.3 Sintaxe

```
// Local Level
kalman(df, y, model="ll")
kalman(df, y, model="ll", sigma_obs=1.0, sigma_state=0.1)

// Local Linear Trend
kalman(df, y, model="llt")
kalman(df, y, model="llt", sigma_obs=0.5, sigma_state=0.05,
   sigma_slope=0.005)
```

Parâmetro	Padrão	Descrição
model	"ll"	"ll" ou "llt"
sigma_obs	sd(diff(y))/$\sqrt{2}$	Desvio-padrão do ruído de observação
sigma_state	sigma_obs × 0.1	Desvio-padrão do ruído de nível
sigma_slope	sigma_state × 0.1	Desvio-padrão do ruído de inclinação (LLT)
Colunas adicionadas ao DataFrame:		
{var}_filtered	–	Nível filtrado (forward pass)
{var}_smoothed	–	Nível suavizado (RTS)
{var}_slope_filtered	–	Inclinação filtrada (só LLT)
{var}_slope_smoothed	–	Inclinação suavizada (só LLT)

Modelo	Usar quando	Relação
LL	Nível drift sem tendência clara	UCM nível local (cap. 44)
LLT	Tendência variante no tempo	HP filter generalizado estocástico

Nota

O Hayashi usa o suavizador RTS com matrizes fixas especificadas pelo usuário. Para estimação conjunta dos parâmetros de variância (σ_ε, σ_η) via máxima verossimilhança, use o comando ucm() (cap. 44), que executa o algoritmo EM sobre a representação de espaço de estados.

Instalação Avançada

53.4 Compilação com suporte a ODBC

Para conectar HAYASHI a bancos de dados via ODBC (SQL Server, Oracle, etc.):

```
cargo install hayashi-lang --features odbc
```

Requer driver ODBC instalado no sistema (unixODBC no Linux).

53.5 Variáveis de ambiente

Variável	Efeito
HAY_HISTORY	Caminho do arquivo de histórico do REPL
HAY_NOCOLOR	Desabilita cores nas mensagens de erro
HAY_TRACE	Habilita trace de execução (depuração)

53.6 Módulos e funcionalidades opcionais

Modelos econométricos avançados

Os estimadores abaixo requerem a crate greeners como dependência do interpretador
Em builds de desenvolvimento (cargo build), ela é compilada automaticamente.
Nenhuma instalação adicional é necessária.

Grupo	Estimadores
Dados em painel	fe, re, ab, pcse, gee, mundlak, xtgls, mixedlm, clogit
Resposta qualitativa	logit, probit, mlogit, ologit, oprobit
Contagem/limitada	poisson, nbreg, zinb, tobit, heckman
GLM geral	glm (5 famílias, múltiplos links)
Regularização	lasso, ridge_reg, enet
Sobrevivência	km, cox
Multivariada	pca, factor, cancorr, manova
Séries temporais	var, svar, vecm, garch, markov, arima
Filtros	hprescott, baxter_king, holtwinters, stl, christiano_fitzgeral
Inferência causal	psm, synth, did, rd

Uso da memória

Modelos computacionalmente intensivos podem requerer ajuste de stack:

```
RUST_MIN_STACK=8388608 hay script.hay  // 8 MB de stack
```

Relevante para VAR/SVAR com muitas defasagens ou PCA em datasets amplos.

53.7 Integração com editores

Neovim

Adicione ao init.lua para reconhecimento de sintaxe básico:

```
vim.filetype.add({ extension = { hay = "hayashi" } })
```

Highlighting completo via Tree-sitter está no roadmap.

VS Code

Extensão disponível na marketplace como hayashi-lang.

53.8 Compilação de desenvolvimento

Para iterar com builds locais (sem --release):

```
git clone https://github.com/sheep-farm/hayashi
cd hayashi
cargo build            // debug build (recomendado para dev)
```

```
./target/debug/hay script.hay
```

O build debug tem verificações extras de bounds e mensagens de erro detalhadas. Não usar --release em desenvolvimento - o ganho de velocidade não compensa a perda de diagnóstico.

Execução de scripts

```
./target/debug/hay script.hay        // executa script
./target/debug/hay                   // abre REPL interativo
./target/debug/hay < script.hay      // lê do stdin
```

Referência Rápida

53.9 Tipos

Tipo	Exemplo	Notas
Int	42, -7	64 bits, com sinal
Float	3.14, 1e-5	IEEE 754
Bool	true, false	
Nil	nil	Ausência de valor
String	"texto"	UTF-8
FString	f"val: {x}"	Interpolação
List	[1, 2, 3]	Índice base 0
Dict	{"a": 1}	Chave string ou int
DataFrame	load "f.csv"as df	COW, colunas Float

53.10 Operadores

Aritmético	+ - * / ^ %	
Comparação	== != > < >= <=	
Lógico	and or not	
Pertencimento	in	
Pipe	|>	
Atribuição	= += -= *= /=	
Range	a..b (exclusivo)	a..=b (inclusivo)

53.11 Operadores de séries temporais

Disponíveis dentro de `generate df col = expr`:

L.x	Lag de ordem 1
L2.x	Lag de ordem 2
F.x	Lead de ordem 1
D.x	Diferença primeira

Constante especial: _n = número de linha (base 1).

53.12 Palavras-chave

```
let const fn return if else for in while break continue
match try catch and or not true false nil quietly
```

quietly on suprime a saída automática; quietly off restaura. A flag respeita escopo.

53.13 Funções auxiliares

Função	Descrição
dataframe({"x": [1, 2]})	Constroi DataFrame a partir de dict
is_int(x)	Testa se x é int
is_str(x)	Testa se x é string
is_df(x)	Testa se x é DataFrame
is_fn(x)	Testa se x é função
contains(s, sub)	Verifica se sub está em s
type(x)	Retorna nome do tipo

53.14 Estatística descritiva

Função	Resultado	Formas
`mean(df, x)`	$E(X)$	`lista / df / if =`
`variance(df, x)`	$\mathrm{Var}(X)$	`lista / df / if =`
`sd(df, x)`	$\sigma(X)$	`lista / df / if =`
`median(df, x)`	`mediana`	`lista / df / if =`
`quantile(df, x, p)`	`percentil` p	`lista / df / if =`
`cov(df, x, y)`	$\mathrm{Cov}(X, Y)$	`df / if =`
`corr_pair(df, x, y)`	$\rho(X, Y)$	`df / if =`
`summarize(df, x)`	`dict completo`	`+ detail=true`
`correlate(df, x, y, ...)`	`matriz de correlação`	`display`
`codebook df`	`sumário por coluna`	`display`
`tabulate df col`	`frequências`	`display`
`xtsum(df, x, id=unit)`	`painel: within/between`	`display`

53.15 Manipulação de DataFrames

Função	Descrição
load "f"as df	Carrega arquivo (CSV, parquet)
save df "f"	Salva arquivo
count(df [, cond])	Número de linhas
filter(df, cond)	Filtra linhas
select(df, cols...)	Seleciona colunas
drop(df, cols...)	Remove colunas
generate df col = expr	Cria/modifica coluna
replace df col = e if c	Modifica com condição
sort(df, col [desc])	Ordena
dropna(df [, cols...])	Remove NaN
ffill(df)	Preenche NaN com último valor válido
append(df1, df2)	Empilha verticalmente
merge(df1, df2, on=col)	Junção horizontal
group_by(df, col, agg)	Agrega por grupo
collapse(df, expr, by=c)	Colapsa por grupo
pivot_longer(...)	Largo → longo
pivot_wider(...)	Longo → largo
reshape(df, ...)	Reshape geral
encode(df, col)	String → numérico
decode(df, col, labels)	Numérico → string
drop_collinear(df)	Remove colunas colineares

53.16 Auxiliares de arquivo e I/O

Função	Descrição
file_exists(path)	Retorna true se o path existe
ensure_dir(path)	Cria diretório (e pais)
write(content, path)	Escreve string em arquivo
export(df, "csv", path)	Salva DataFrame em CSV
load "path"as df	Carrega CSV/parquet em DataFrame

53.17 Declaração de estrutura

Comando	Efeito
`xtset df id time`	Declara painel (unidade e tempo)
`tsset df time`	Declara série temporal (habilita L.x, F.x, D.x)

53.18 Geradores aleatórios

Função	Distribuição
`rnormal()`	$N(0,1)$ - normal padrão
`runiform()`	$U(0,1)$ - uniforme
`rbernoulli(p)`	Bernoulli com probabilidade p fixa
`seed(n)`	Fixa semente PRNG para reprodutibilidade

53.19 Regressão linear e extensões

Estimador	Sintaxe
OLS	`ols(y ~ x1 + x2, df)`
IV/2SLS	`iv(y ~ x, ~ z, df)`
GMM	`gmm(y ~ x, ~ z1 + z2, df)`
WLS	`wls(y ~ x, df, weights="w")`
Rolling OLS	`rolling(y ~ x, df, window=50 [, date=col])`
LASSO	`lasso(y ~ x1+...+xk, df, alpha=0.1)`
Ridge	`ridge_reg(y ~ x1+...+xk, df, alpha=0.5)`
Elastic Net	`enet(y ~ x1+...+xk, df, alpha=0.5)`
Bootstrap	`bootstrap("ols", y ~ x, df, reps=500)`
Bootstrap SE	`bootse("ols", y ~ x, df, n=500)`
Regressão quantílica	`qreg(y ~ x, df, tau=0.5)`
Regressão robusta	`rlm(y ~ x, df)`
LOWESS	`lowess(df, y, x, frac=0.3)`
SUR (Zellner)	`sur(df, y1 ~ x1, y2 ~ x2)`

Regularização: em lasso, alpha é a força de penalização λ (maior = mais zeros). Em enet, alpha é o mixing L1/(L1+L2) (0 = Ridge, 1 = LASSO).

53.20 Dados em painel

Estimador	Sintaxe
FE (within)	`fe(y ~ x, df, id=unit)`
RE (GLS)	`re(y ~ x, df, id=unit)`
FE com IV	`feiv(y ~ x, z ~ z+w, df, id="unit")`
Hausman	`hausman(m_fe, m_re)`
Mundlak (CRE)	`mundlak(df, y ~ x, id="unit")`
AB-GMM	`ab(y ~ x, df, id=unit, time=t, lags=2)`
PCSE	`pcse(y ~ x, df, id=unit, time=t)`
GEE	`gee(y ~ x, df, id=unit)`
xtgls	`xtgls(y ~ x, df, id=unit, time=t)`
Modelo misto	`mixedlm(y ~ x, df, id="unit")`
Logit cond. (FE)	`clogit(y ~ x, df, group="unit")`

53.21 Inferência causal

Método	Sintaxe
DiD	`did(y ~ treated + post, df)`
RDD	`rd(y ~ x, 0.0, df)`
Fuzzy RDD	`fuzzy_rd(y ~ x, "treat", 0.0, df)`
PSM	`psm(y ~ d + x1, df, k=1, caliper=0.05, boot=200)`
Controle sintético	`synth("y", treated_id, t0, df, id="col", time="col")`

PSM: sem caliper, ATT pode ser tendencioso mesmo com PSM. **synth**: t0 = primeiro período pós-tratamento (inclusive).

53.22 Variáveis dependentes especiais

Modelo	Sintaxe	
Logit / Probit	`logit(y ~ x, df)`	`probit(y ~ x, df)`
Efeitos marginais	`margins(m, df)`	
Logit multinomial	`mlogit(y ~ x, df)`	
Logit ordenado	`ologit(y ~ x, df)`	
Probit ordenado	`oprobit(y ~ x, df)`	
Poisson	`poisson(y ~ x, df)`	
Binomial negativa	`nbreg(y ~ x, df)`	
Zero-inflacionada	`zinb(y ~ x, df)`	
Tobit	`tobit(y ~ x, df)`	
Heckman	`heckman(y_out ~ x, s ~ z, df)`	
GLM geral	`glm(y ~ x, df, family=gamma, link=log)`	

Família GLM	Links disponíveis
gaussian	identity, log, inverse
binomial	logit, probit, cloglog
poisson	log, identity
gamma	log, inverse, identity
inverseGaussian	inverse_squared

53.23 Análise de sobrevivência

Método	Sintaxe
Kaplan-Meier	`km(time, event, df)`
Cox PH	`cox(time ~ x1 + x2, df, event=event)`

Cox: exp(coef) = hazard ratio. event é coluna indicadora de falha (1) / censura (0).

53.24 Séries temporais

Modelo	Sintaxe
ARMA/ARIMA	`arima(df, y, p=1, d=0, q=1)`
AR(p) via OLS	`autoreg(df, y, lags=2, trend="c")`
ARDL(p,q)	`ardl(y ~ x1 + x2, df, lags=p, xlags=q)`
GARCH	`garch(df, y, p=1, q=1)`
EGARCH / GJR	`egarch(df, y, p=1, q=1)` `gjrgarch(df, y, p=1, q=1)`
ADF	`adf(df, var)`
KPSS	`kpss(df, var)`
Phillips-Perron	`phillips_perron(df, var)`
Zivot-Andrews	`zivot(df, var)`
VAR	`var(df, y1, y2, lags=2)`
VECM	`vecm(df, y1, y2, lags=1, r=1)`
Cointegração	`johansen(df, y1, y2, lags=1)`
Granger	`granger(df, y1, y2, lags=2)`
IRF (VAR reduz.)	`irf(m_var, steps=12)`
SVAR Cholesky	`svar(df, y1, y2, lags=1, id=cholesky)`
SVAR longo prazo	`svar(df, y1, y2, lags=1, id=longrun)`
IRF estrutural	`sirf(m_svar, steps=12)`
Markov Switching	`markov(df, y, k=2, p=1)`

53.25 Filtros e decomposição

Filtro	Sintaxe	Output
Hodrick-Prescott	`hprescott(df, y, lambda=1600)`	y_trend,
Baxter-King	`baxter_king(df, y, low=6, high=32, k=12)`	y_cycle
Holt-Winters / ETS	`holtwinters(df, y, trend=add, seasonal=add, period=12)`	modelo E
STL	`stl(df, y, period=12)`	y_trend,
Christiano-Fitz.	`christiano_fitzgerald(df, y, low=6, high=32)`	y_cycle

Perda de borda: HP perde pontas; BK perde k observações em cada extremo; CF e STL mantêm todos os pontos. **Nota:** hamilton() é alias do Markov Switching (Hamilton 1989), *não* do filtro Hamilton 2018. **autoreg trend:** "c" = constante (padrão); "ct" = const + tendência; "t" = só tendência; "n" = nenhum. **ARDL:** fórmula padrão y ~ x; lags = defasagens de y; xlags = defasagens de cada regressor; inclui x contemporâneo automaticamente.

53.26 Análise multivariada

Método	Sintaxe
PCA	`pca(df, v1, v2, ..., n=2)`
Análise fatorial	`factor(df, v1, v2, ..., n=2, rotation=varimax)`
Correlação canôn.	`cancorr(df, xvars=["v1","v2"], yvars=["v3","v4"])`
MANOVA	`manova(df, y1, y2, by="grupo")`
KDE	`kde(df, var, bwidth=0.5)`

53.27 Testes de hipóteses

Teste	Sintaxe
t - uma amostra	`ttest(df, var, mu=0.0)`
t - dois grupos	`ttest(df, var, by=grupo)`
t - pareado	`ttest(df, var1, var2, paired=true)`
ANOVA	`anova(df, var, by=grupo)`
Kruskal-Wallis	`kruskal_wallis(df, var, by=grupo)`
Normalidade (SW)	`swilk(df, var)`
Normalidade (SF)	`sfrancia(df, var)`
Homocedasticidade	`vif(m_ols)`
Hausman	`hausman(m_fe, m_re)`
Teste linear	`test(m, "x1 = x2")`
Teste conjunto	`testparm(m, ["x1", "x2"])`
Combinação linear	`lincom(m, "x1 + x2")`

53.28 Apresentação de resultados

Função	Descrição
display m	Exibe resultado do modelo
esttab(m1, m2, ...)	Tabela comparativa
eststo(m)	Armazena modelo para esttab
estclear()	Limpa modelos armazenados
predict(m, df)	Adiciona coluna yhat ao df
margins(m, df)	Efeitos marginais médios (AME)
tidy(m)	DataFrame organizado de coeficientes
glance(m)	DataFrame de resumo do modelo (1 linha)

53.29 Construção de painel (fórmulas)

Para construir painel a partir de DataFrame empilhado (N unidades, T períodos):

```
generate df unit = (_n - 1) % N + 1
generate df t    = (_n - (_n - 1) % N - 1) / N + 1
```

Onde N é o número de unidades. Substitua N pelo valor numérico.

53.30 Atalhos do REPL

Atalho	Ação
exit	Sair do REPL
Ctrl-D	Sair do REPL (EOF)
Ctrl-C	Cancelar linha atual
Ctrl-R	Buscar no histórico

Código Equivalente por Plataforma

Este apêndice reproduz DGPs representativos em Stata, R e Python, lado a lado com o código Hayashi. O objetivo é duplo: permitir validação cruzada de resultados e evidenciar diferenças de sintaxe.

Nota

Os geradores de números aleatórios diferem entre plataformas - os valores numéricos serão distintos, mas as propriedades estatísticas (recuperação de parâmetros, direção do viés, decisão do Hausman) são idênticas para qualquer semente.

53.31 OLS — DGP sem ruído e comparação com variável omitida

Hayashi

```
seed(42)
input df
id
1
end
for i in 1..=10 { df = append(df, df) }
df |> filter(_n <= 1000)
generate df x1 = rnormal()
generate df x2 = rnormal()
generate df y  = 2.5 + 1.8*x1 - 0.7*x2
let m1 = ols(y ~ x1 + x2, df)
let m2 = ols(y ~ x1, df)
esttab(m1, m2)
```

Stata

```
clear
set obs 1000
set seed 42
gen x1 = rnormal()
gen x2 = rnormal()
gen y  = 2.5 + 1.8*x1 - 0.7*x2
reg y x1 x2
estimates store m1
reg y x1
estimates store m2
esttab m1 m2          // ssc install estout
```

R

```
set.seed(42)
n  <- 1000
x1 <- rnorm(n);  x2 <- rnorm(n)
y  <- 2.5 + 1.8*x1 - 0.7*x2
m1 <- lm(y ~ x1 + x2)
m2 <- lm(y ~ x1)
library(stargazer)
stargazer(m1, m2, type = "text")
```

Python

```
import numpy as np, pandas as pd
import statsmodels.formula.api as smf
from statsmodels.iolib.summary2 import summary_col
np.random.seed(42)
n = 1000
x1 = np.random.randn(n);  x2 = np.random.randn(n)
y  = 2.5 + 1.8*x1 - 0.7*x2
df = pd.DataFrame({'y': y, 'x1': x1, 'x2': x2})
m1 = smf.ols('y ~ x1 + x2', data=df).fit()
m2 = smf.ols('y ~ x1',      data=df).fit()
print(summary_col([m1, m2], stars=True))
```

53.32 IV — Endogeneidade com instrumento válido

Hayashi

```
seed(42)
...
generate df z   = rnormal()
generate df v   = rnormal()
let d1          = cov(df, z, v) / variance(df, z)
let d0          = mean(df, v) - d1 * mean(df, z)
generate df eps = v - d0 - d1*z
generate df x   = 0.8*z + eps
generate df y   = 1.5 + 2.0*x + 0.7*eps
let m_ols = ols(y ~ x, df)
let m_iv  = iv(y ~ x, ~ z, df)
esttab(m_ols, m_iv)
```

Stata

```
gen z = rnormal()
gen v = rnormal()
reg v z
predict eps, residuals
gen x = 0.8*z + eps
gen y = 1.5 + 2.0*x + 0.7*eps
reg y x
ivregress 2sls y (x = z)
```

R

```
eps <- resid(lm(v ~ z))
x   <- 0.8*z + eps
y   <- 1.5 + 2.0*x + 0.7*eps
library(ivreg)
ivreg(y ~ x | z, data = data.frame(y,x,z))
```

Python

```
from linearmodels.iv import IV2SLS
import statsmodels.api as sm
IV2SLS(y, sm.add_constant(x),
       None, sm.add_constant(z)).fit()
```

53.33 Painel — FE, RE e Hausman

Hayashi

```
seed(42) / ... / df |> filter(_n <= 1000)
generate df id    = (_n - 1) % 100 + 1
generate df alpha = id * 0.05
generate df x     = alpha + rnormal()
generate df y     = 1.0 + 2.0*x + alpha + rnormal()
let m_fe  = fe(y ~ x, df, id=id)
let m_re  = re(y ~ x, df, id=id)
hausman(m_fe, m_re)
```

Stata

```
xtset id
xtreg y x, fe
estimates store m_fe
xtreg y x, re
estimates store m_re
hausman m_fe m_re
```

R

```
library(plm)
pd   <- pdata.frame(df, index = "id")
m_fe <- plm(y ~ x, data = pd, model = "within")
m_re <- plm(y ~ x, data = pd, model = "random")
phtest(m_fe, m_re)
```

Python

```
from linearmodels import PanelOLS, RandomEffects
m_fe = PanelOLS(df['y'], df[['x']], entity_effects=True).fit
    ()
m_re = RandomEffects(df['y'], df[['x']]).fit()
```

53.34 Logit e Probit

Hayashi

```
generate df p = 1 / (1 + exp(-(-1.0 + 2.0*x)))
generate df y = runiform() < p
let m_l = logit(y ~ x, df)
let m_p = probit(y ~ x, df)
margins(m_l, df)
```

Stata

```
logit  y x
probit y x
margins, dydx(x)
```

R

```
m_l <- glm(y ~ x, family = binomial("logit"))
m_p <- glm(y ~ x, family = binomial("probit"))
library(margins)
summary(margins(m_l))
```

Python

```
import statsmodels.formula.api as smf
m_l = smf.logit('y ~ x', data=df).fit()
print(m_l.get_margeff().summary())
```

53.35 Modelos de contagem

Hayashi

```
generate df lam = exp(0.5 + 1.0*x)
generate df y   = floor(lam + runiform())
let m_p  = poisson(y ~ x, df)
let m_nb = nbreg(y ~ x, df)
```

Stata

```
poisson y x
nbreg   y x
```

R

```
glm(y ~ x, family = poisson("log"))
library(MASS)
glm.nb(y ~ x)
```

Python

```
smf.poisson('y ~ x', data=df).fit()
smf.negativebinomial('y ~ x', data=df).fit()
```

53.36 Tobit e Heckman

Hayashi

```
let m_t = tobit(y_tobit ~ x, df)
let m_h = heckman(y_out ~ x, s ~ z, df)
```

Stata

```
tobit y_tobit x, ll(0)
heckman y_out x, select(s = x z)
```

R

```
library(AER)
tobit(y_tobit ~ x, left = 0)
library(sampleSelection)
heckit(s ~ x + z, y_out ~ x)
```

Python

```
# Tobit: sem implementação nativa no statsmodels
# Heckman: via heckit do pyheckit ou statsmodels
from statsmodels.regression.linear_model import OLS
# implementação manual via two-step
```

53.37 Modelos multinomiais ordenados

Hayashi

```
let m_ol = ologit(y ~ x, df)
let m_op = oprobit(y ~ x, df)
let m_ml = mlogit(y ~ x, df)
```

Stata

```
ologit  y x
oprobit y x
mlogit  y x
```

R

```
library(MASS)
polr(factor(y) ~ x, method = "logistic")  # ologit
polr(factor(y) ~ x, method = "probit")    # oprobit
library(nnet)
multinom(y ~ x)                           # mlogit
```

Python

```
from statsmodels.miscmodels.ordinal_model import OrderedModel
OrderedModel(y, X, distr='logit').fit()
OrderedModel(y, X, distr='probit').fit()
smf.mnlogit('y ~ x', data=df).fit()
```

53.38 Regularização (LASSO, Ridge, Elastic Net)

Hayashi

```
let m_las = lasso(y ~ x1+x2+...+x10, df, alpha=0.1)
let m_rid = ridge_reg(y ~ x1+...+x10, df, alpha=0.5)
let m_ent = enet(y ~ x1+...+x10, df, alpha=0.5)
display m_las
```

Stata

```
// requer Stata 16+
lasso linear y x1-x10
cvlasso y x1-x10
```

R

```
library(glmnet)
X <- model.matrix(y ~ ., df)[,-1]
# LASSO (alpha=1), Ridge (alpha=0), Enet (alpha=0.5)
cv.glmnet(X, df$y, alpha = 1)   # LASSO com CV
cv.glmnet(X, df$y, alpha = 0)   # Ridge com CV
cv.glmnet(X, df$y, alpha = 0.5) # Enet com CV
```

Python

```
from sklearn.linear_model import Lasso, Ridge, ElasticNet
Lasso(alpha=0.1).fit(X, y)
Ridge(alpha=0.5).fit(X, y)
ElasticNet(alpha=0.1, l1_ratio=0.5).fit(X, y)
```

Nota de escala: alpha em Hayashi e statsmodels é λ (força de penalização). Em glmnet (R), os dados são padronizados automaticamente e lambda é escolhido por validação cruzada. Em scikit-learn, alpha é análogo a λ.

53.39 Análise de sobrevivência

Hayashi

```
let m_km  = km(time, event, df)
let m_cox = cox(time ~ x, df, event=event)
display m_cox
```

Stata

```
stset time, failure(event)
sts graph              // Kaplan-Meier
stcox x                // Cox PH
stcox x, nohr          // coeficientes log-hazard
```

R

```
library(survival)
km_fit  <- survfit(Surv(time, event) ~ 1)
cox_fit <- coxph(Surv(time, event) ~ x)
summary(cox_fit)
```

Python

```
from lifelines import KaplanMeierFitter, CoxPHFitter
kmf = KaplanMeierFitter().fit(df['time'], df['event'])
cph = CoxPHFitter().fit(df, 'time', 'event')
```

53.40 SVAR — Cholesky e Blanchard-Quah

Hayashi

```
let m_ch = svar(df, y1, y2, lags=1, id=cholesky)
let m_bq = svar(df, y1, y2, lags=1, id=longrun)
let m_ir = sirf(m_ch, steps=12)
display m_ir
```

Stata

```
var y1 y2, lags(1)
// Cholesky (padrao do IRF pos-VAR):
irf create myirf, step(12) replace
irf table sirf
// Blanchard-Quah requer restrições manuais via svar
matrix A = (1, 0 \ ., 1)
matrix B = (., 0 \ ., .)
svar y1 y2, lags(1) aeq(A) beq(B)
```

R

```
library(vars)
m <- VAR(cbind(y1, y2), p = 1)
# Cholesky:
irf(m, n.ahead = 12, ortho = TRUE)
# Blanchard-Quah:
BQ(m)
```

Python

```
from statsmodels.tsa.api import VAR
m  = VAR(df[['y1', 'y2']]).fit(1)
# Cholesky:
m.irf(12).orth_ma            # orthogonalized IRF
# BQ: requer implementação manual em Python
```

53.41 Filtros de decomposição

Hayashi

```
hprescott(df, y, lambda=1600)
baxter_king(df, y, low=6, high=32, k=12)
stl(df, y, period=12)
christiano_fitzgerald(df, y, low=6, high=32)
```

Stata

```
// ssc install hprescott
hprescott y, smooth(1600)      // HP
// ssc install bking
bking y, minperiod(6) maxperiod(32) stub(bk)
// ssc install cfitzroy
// stl: via Python/R (nao disponivel nativamente no Stata)
```

R

```
library(mFilter)
hp <- hpfilter(y, freq = 1600)
bk <- bkfilter(y, pl = 6, pu = 32, nfix = 12)
cf <- cffilter(y, pl = 6, pu = 32)
library(stats)
stl(ts(y, frequency=12), s.window="periodic")
```

Python

```
from statsmodels.tsa.filters.hp_filter import hpfilter
from statsmodels.tsa.filters.bk_filter import bkfilter
from statsmodels.tsa.seasonal import STL
_, trend = hpfilter(y, lamb=1600)
cycle_bk = bkfilter(y, low=6, high=32, K=12)
stl = STL(y, period=12).fit()
```

53.42 PSM e Controle Sintético

Hayashi

```
let m_psm = psm(y ~ d + x1, df, k=1, caliper=0.05, boot=200)
display m_psm

// Controle Sintético (painel)
let m_sc = synth("y", 1, 11, df, id="unit", time="year")
display m_sc
```

Stata

```
// ssc install psmatch2
psmatch2 d x1, outcome(y) neighbor(1) caliper(0.05)
// ssc install synth
synth y x1, trunit(1) trperiod(11) xperiod(1/10)
```

R

```
library(MatchIt)
m <- matchit(d ~ x1, data=df, method="nearest",
             caliper=0.05)
library(Synth)
dataprep.out <- dataprep(...)     # setup
synth(dataprep.out)
```

Python

```
# PSM:
from sklearn.linear_model import LogisticRegression
# calcular propensity scores e fazer matching manual
# Controle Sintético:
# pip install pysynth
from pysynth import Synth
```

53.43 Resumo comparativo

Tarefa	Hayashi	Stata	R	Python
Regressão				
OLS	ols()	reg	lm()	smf.ols()
IV/2SLS	iv()	ivregress	ivreg()	IV2SLS()
GMM	gmm()	gmm	gmm()	LinearIVGMM()
WLS	wls()	vwls	lm(weights=)	WLS()
Rolling OLS	rols()	rolling()	rollregres()	RollingOLS()
LASSO	lasso()	lasso linear	glmnet(a=1)	Lasso()
Ridge	ridge_reg()	-	glmnet(a=0)	Ridge()
Elastic Net	enet()	-	glmnet(a=.5)	ElasticNet()
Bootstrap	bootstrap()	bootstrap:	boot()	resample()
Bootstrap SE	bootse()	-	-	-
LOWESS	lowess()	lowess	lowess()	lowess()
Quant. reg.	qreg()	qreg	rq()	QuantReg()
SUR	sur()	sureg	systemfit()	SUR()
Dados em painel				
FE (within)	fe()	xtreg, fe	plm(within)	PanelOLS()
RE (GLS)	re()	xtreg, re	plm(random)	RandomEffects()
Hausman	hausman()	hausman	phtest()	manual
AB-GMM	ab()	xtabond2	pgmm()	FirstDifferenceGMM()
PCSE	pcse()	xtpcse	pcse()	-
GEE	gee()	xtgee	geeglm()	GEE()
Mundlak (CRE)	mundlak()	manual	mundlak()	manual
xtgls	xtgls()	xtgls	pggls()	-
Modelo misto	mixedlm()	mixed	lme4::lmer()	MixedLM()
Logit FE	clogit()	clogit	clogit()	ConditionalLogit()
Resumo painel	xtsum()	xtsum	-	-
Variáveis dependentes especiais				
Logit	logit()	logit	glm(logit)	smf.logit()
Probit	probit()	probit	glm(probit)	smf.probit()
Logit ordenado	ologit()	ologit	polr()	OrderedModel
Probit ordenado	oprobit()	oprobit	polr()	OrderedModel
Logit multinom.	mlogit()	mlogit	multinom()	MNLogit()
Poisson	poisson()	poisson	glm(poisson)	smf.poisson()
Neg. Binomial	nbreg()	nbreg	glm.nb()	smf.negativebinomial(
Zero-inflated	zinb()	zinb	zeroinfl()	ZeroInflatedNB
Tobit	tobit()	tobit	AER::tobit()	manual
Heckman	heckman()	heckman	heckit()	manual
GLM geral	glm()	glm	glm()	GLM()
Ef. marginais	margins()	margins	margins()	get_margeff()
Inferência causal				
DiD	did()	diff	lm()	smf.ols()

(continua na próxima página

(continuação)

Tarefa	Hayashi	Stata	R	Python
RDD	rd()	rdrobust	rdrobust()	rdd()
Fuzzy RDD	fuzzy_rd()	rdrobust	rdrobust()	rdd()
PSM	psm()	psmatch2	matchit()	manual
Controle sint.	synth()	synth	Synth	pysynth
Sobrevivência				
Kaplan-Meier	km()	sts	survfit()	KaplanMeierFitter
Cox PH	cox()	stcox	coxph()	CoxPHFitter
Séries temporais				
AR/MA/ARMA	arima()	arima	arima()	ARIMA()
AR(p) OLS	autoreg()	arima	arima()	AutoReg()
ARDL(p,q)	ardl()	ardl	dynlm()	ardl()
GARCH	garch()	arch	garch()	arch()
Raiz unit. (ADF)	adf()	dfuller	adf.test()	adfuller()
Phillips-Perron	phillips_perron()	pperron	PP.test()	PhillipsPerron()
Zivot-Andrews	zivot()	zandrews	ur.za()	ZivotAndrews()
Cointegração	johansen()	vecrank	ca.jo()	coint_johansen
VAR	var()	var	VAR()	VAR()
IRF (reduzida)	irf()	irf create	irf()	irf().plot()
SVAR Cholesky	svar(chol.)	svar	id.chol()	manual
SVAR BQ	svar(long.)	svar	BQ()	manual
IRF estrutural	sirf()	irf create	irf()	manual
Markov Switch.	markov()	mswitch	msmFit()	MarkovRegression
Filtros e decomposição				
HP	hprescott()	hprescott*	hpfilter()	hpfilter()
Baxter-King	baxter_king()	bking*	bkfilter()	bkfilter()
Holt-Winters	holtwinters()	tssmooth	HoltWinters()	ExponentialSmoothing()
STL	stl()	-	stl()	STL()
CF filter	christiano_fitzgerald()	cfitzroy	cffilter()	-
Análise multivariada				
PCA	pca()	pca	prcomp()	PCA()
Análise fatorial	factor()	factor	factanal()	FactorAnalysis()
Cor. canônica	cancorr()	canon	cancor()	CCA()
MANOVA	manova()	manova	manova()	MANOVA()
Inferência e testes				
Teste t	ttest()	ttest	t.test()	ttest_ind()
ANOVA	anova()	anova	aov()	f_oneway()
Kruskal-Wallis	kruskal_wallis()	kwallis	kruskal.test()	kruskal()
Normalidade	swilk()	swilk	shapiro.test()	shapiro()
Teste conjunto	testparm()	testparm	linearHypothesis()	wald_test()

(continua na próxima página)

(continuação)

Tarefa	Hayashi	Stata	R	Python
Tabela	esttab()	esttab†	stargazer()	summary_col()
Armazenar modelo	eststo()	eststo	-	-
Limpar modelos	estclear()	-	-	-

* HP e BK no Stata requerem ssc install hprescott e ssc install bking.

† esttab no Stata requer ssc install estout.

Nota sobre geradores aleatórios: rnormal() corresponde ao gerador normal padrão usado nos exemplos de Stata, R e Python. Use runiform() quando o DGP exigir um sorteio $U(0,1)$.

53.44 Matriz de validação

Além dos exemplos de código deste apêndice, todo HAYASHI é validado sistematicame contra implementações de referência. O programa de validação está no diretório validation/ e é executado com:

```
hay validate
```

A Tabela 53.2 lista os casos de validação atuais, com a família de estimadores a fonte dos dados e as referências (R, Python e/ou Stata) usadas na comparação. Os detalhes completos --- tolerâncias, notas e status --- estão em validation/MAT

ID do Caso	Família	Dataset/Fonte	Referências (R/Python/Stata)
ab_grunfeld	ab	wooldridge::grunfeld	R
ar_book	arima	simulated_ar1	R, Python
ardl_gdp	ardl	statsmodels::macrodata	R, Python
arima_book	arima	simulated_rw	R, Python
arima_gdp	arima	statsmodels::macrodata	R, Python
arma_book	arima	simulated_arma11	R, Python
autoreg_gdp	autoreg	statsmodels::macrodata	R, Python
coint_book	vecm	simulated_cointegrated	R, Python
cox_heart	cox	statsmodels::heart	R, Python
did_kielmc	did	wooldridge::kielmc	R, Python
ets_gdp	ets	statsmodels::macrodata	R, Python
garch_book	garch	simulated_garch11	Python
garch_nyse	garch	wooldridge::nyse	R, Python
glsar_hprice1	glsar	wooldridge::hprice1	R, Python
gmm_card	gmm	wooldridge::card	R, Python

(continua na próxima página)

(continuação)

ID do Caso	Família	Dataset/Fonte	Referências (R/Python/Stata)
heckman_mroz	heckman	wooldridge::mroz	R, Python
iv_card	iv	wooldridge::card	R, Python
lasso_hprice1	lasso	wooldridge::hprice1	R, Python
logit_mroz	logit	wooldridge::mroz	R, Python
ma_book	arima	simulated_ma1	R, Python
ologit_beauty	logit	wooldridge::beauty	R, Python
ols_wooldridge_wage1	ols	wooldridge::wage1	R, Python
panel_fe_grunfeld	panel_fe	wooldridge::grunfeld	R, Python
poisson_fertil2	poisson	wooldridge::fertil2	R, Python
probit_mroz	probit	wooldridge::mroz	R, Python
psm_jtrain3	psm	wooldridge::jtrain3	R, Python
quantile_wage1	qreg	wooldridge::wage1	R, Python
re_grunfeld	re	grunfeld	R, Python
rdd_book	rdd	rdd_book	R, Python
synth_smoking	synth	synth_smoking	R, Python
tobit_mroz	tobit	wooldridge::mroz	R, Python
var_book	var	simulated_var1	R, Python
var_macro	var	statsmodels::macrodata	R, Python
wls_hprice1	wls	wooldridge::hprice1	R, Python

Índice Remissivo