Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Estatística

Detecção de pontos de mudança em séries temporais com base no algoritmo *Pruned Exact Linear Time (PELT)*

Nicole Maria Onofre

Trabalho de Conclusão de Curso

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Estatística

Detecção de pontos de mudança em séries temporais com base no algoritmo *Pruned Exact Linear Time (PELT)*

Nicole Maria Onofre
Orientadora: Maria Sílvia de Assis Moura

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do título de Bacharel em Estatística.

São Carlos
Dezembro de 2025

Resumo

A análise de séries temporais é uma técnica estatística voltada ao estudo de observações coletadas ao longo do tempo, que naturalmente apresentam uma dependência entre si. Esse tipo de análise desempenha um papel importante na compreensão do comportamento dos dados ao longo do tempo, especialmente quando há indícios de que este comportamento sofreu alguma modificação. Essas alterações, conhecidas como pontos de mudança (*change points*), representam modificações significativas em características da série como média, variância ou tendência, e estão geralmente associadas a eventos de grande impacto no contexto dos dados, como crises econômicas, choques políticos ou pandemias.

Com o avanço computacional e, consequentemente, a disponibilidade de séries temporais mais extensas, surge a necessidade de desenvolver e estudar métodos que consigam fazer a detecção destes pontos de mudança em dados extensos sem ter uma complexidade computacional muito alta, ou seja, métodos que sejam precisos e eficientes.

Diante deste contexto, este trabalho pretende identificar pontos de mudança em uma série financeira real, utilizando o algoritmo *Pruned Exact Linear Time* (PELT), o qual consegue ter uma boa precisão na identificação dos pontos e também uma complexidade computacional baixa, uma vez que alia a minimização de uma função de custo com um método de poda.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

A análise de séries temporais é uma técnica estatística voltada ao estudo de observações coletadas ao longo do tempo que naturalmente apresentam uma dependência entre si (Morettin e Toloi, 2018). Uma das motivações para investigar esses dados é a possibilidade de identificar mudanças no comportamento da série, o que pode indicar alterações no processo gerador dos dados (Kuo, 2022).

Essas alterações são chamadas pontos de mudança, ou *change points*, e elas representam uma alteração significativa na estrutura dos dados, podendo ocorrer em parâmetros como a média, a variância ou a tendência. Essas quebras estruturais podem ser provocadas por eventos externos relevantes, como crises econômicas, intervenções políticas ou fatores ambientais, dependendo do contexto dos dados. Detectar esses pontos permite uma compreensão mais precisa da dinâmica dos dados e possibilita a segmentação da série em trechos com comportamentos estatísticos mais homogêneos (Kuo, 2022).

Nos últimos anos, com o avanço computacional, a detecção de múltiplos pontos de mudança tornou-se um assunto de grande interesse na Estatística, principalmente diante do aumento da disponibilidade de séries temporais mais extensas. Um dos principais desafios, nestes casos, está em determinar ao mesmo tempo o número ótimo e as localizações dos pontos onde há alguma alteração, uma tarefa cuja complexidade computacional aumenta rapidamente com o aumento do tamanho da série (Wambui *et al.*, 2015). Diante desse desafio, diversos algoritmos têm sido desenvolvidos e estudados, numa tentativa de buscar métodos que sejam ao mesmo tempo precisos e computacionalmente eficientes. (Wambui *et al.*, 2015). Dentre as abordagens existentes, o algoritmo *Pruned Exact Linear Time* (PELT), baseado na minimização de uma função de custo para encontrar os pontos ótimos de mudança e em uma constante de penalização para que não seja encontrados pontos de

maneira excessiva, se destaca por conseguir combinar precisão na segmentação da série com alta eficiência computacional, uma vez que, sob algumas condições, apresenta um custo computacional linear em relação ao número n de observações (Killick *et al.*, 2012).

Esse tipo de abordagem é particularmente relevante no contexto de séries financeiras, em que há o interesse em detectar pontos de mudança em índices de bolsas de valores, seja para entender dinâmicas de mercado ou para analisar o impacto da ocorrência de algum evento político. Um exemplo deste contexto é o índice Ibovespa, principal indicador do desempenho médio das ações mais negociadas na B3. Por estar associado ao comportamento da economia brasileira, o Ibovespa é sensível a acontecimentos macroeconômicos e políticos relevantes, como a crise financeira global de 2008, a recessão nacional entre 2014 e 2016 e o impacto da pandemia de COVID-19 em 2020. Esses eventos podem causar mudanças abruptas no comportamento do índice, evidenciando pontos de transição que podem ser detectados e analisados estatisticamente. Identificar esses momentos de transição é de grande interesse para economistas e investidores, pois permite compreender melhor os impactos desses choques e antecipar possíveis tendências futuras.

Diante desse contexto, este trabalho tem como objetivo estudar o funcionamento do algoritmo *Pruned Exact Linear Time* e analisar uma série financeira real, a ser obtida por meio do site Yahoo Finance, com o intuito de identificar pontos de mudança ao longo da série, utilizando o algoritmo. Para isso, serão consideradas alguns tipos de penalização, de modo a avaliar os impactos que essas escolhas têm sobre os resultados do método.

A estrutura deste trabalho está organizada da seguinte forma: no Capítulo 2, apresentamos a definição matemática de pontos de mudança, assim como uma metodologia para sua detecção, com foco no algoritmo PELT; No Capítulo 3, analisamos o desempenho do PELT em diferentes cenários controlados, variando a quantidade e a localização dos pontos de mudança para avaliar a influência das penalidades nos resultados. No Capítulo 4, utilizamos o algoritmo para detectar possíveis mudanças estruturais na série histórica do índice Ibovespa, permitindo comparar o comportamento do método em dados reais e simulados. Por fim, no Capítulo 5, apresentamos as considerações gerais do trabalho, incluindo os principais achados e as principais recomendações para estudos futuros.

Capítulo 2

Materiais e métodos

2.1 Definições gerais

Conforme definido por Killick *et al.* (2012), seja $y_{1:n} = (y_1, y_2, ..., y_n)$ uma sequência de dados ordenados ao longo do tempo. Admitimos a existência de $m > 0$ pontos de mudança, localizados nas posições $\tau_{1:m} = (\tau_1, \tau_2, ..., \tau_m)$, e fixamos $\tau_0 = 0$ e $\tau_{m+1} = n$. A partir desses pontos, a segmentação dos dados ocorre de forma que o intervalo entre dois pontos de mudança consecutivos forme um segmento de dados, isto é, o i-ésimo segmento corresponde ao conjunto $y_{(\tau_{i-1}+1):\tau_i}$.

Uma abordagem comumente utilizada na literatura para identificar múltiplos pontos de mudança é a baseada na minimização de uma função de custo, a qual quantifica a dissimilaridade entre os segmentos formados. Formalmente, a lógica por trás dessa abordagem é minimizar a expressão:

$$\sum_{i=1}^{m+1} C(y_{(\tau_{i-1}+1):\tau_i}) + \beta f(m), \tag{2.1}$$

em que $C(\cdot)$ é a função de custo associada a cada segmento, e $\beta f(m)$ é uma penalidade para evitar o sobreajuste - isto é, a criação de um número desnecessário de segmentações -, dependente do número total de pontos de mudança identificados (Wambui *et al.*, 2015).

Como colocado por Wambui *et al.* (2015), a escolha da função de custo a ser utilizada depende das características dos dados que estão sendo analisados e do tipo de mudança que se deseja detectar. Segundo Truong *et al.* (2019), em contextos paramétricos, em que assumimos que os dados seguem uma distribuição conhecida e o vetor de parâmetros dessa distribuição tem dimensão finita, as principais funções de custo baseiam-se na estimação

de máxima verossimilhança associada aos dados. Nesses casos, os pontos observados são modelados como uma sequência de variáveis aleatórias e identicamente distribuídas (*i.i.d.*) segundo uma família de distribuições (ao longo do trabalho, será discutido modelos sem esta suposição). Os pontos de mudança são representados por instantes τ_i nos quais os parâmetros da distribuição (como a média ou a variância) sofrem alguma alteração. O objetivo é encontrar esses instantes maximizando a verossimilhança dos possíveis segmentos, o que equivale a minimizar o custo definido como o logaritmo negativo da verossimilhança de cada segmento, ou seja, uma soma de log-verossimilhanças negativas. Matematicamente, essa função de custo pode ser definida como:

$$C(y_{a:b}) := -\sup_{\theta} \sum_{h=a+1}^{b} \ln f(y_h \mid \theta), \tag{2.2}$$

em que $f(y_h \mid \theta)$ é a densidade da distribuição parametrizada por θ, sendo θ o vetor de parâmetros da distribuição; $a+1$ representa o ínicio do trecho da série, enquanto b representa o último ponto desse segmento; e $y_{a:b}$ é o segmento correspondente ao conjunto de posições.

Essa função de custo é a base para diversas outras funções utilizadas em casos mais específicos, como:

- **Mudanças na média:** assume-se que os dados seguem uma distribuição Normal com variância fixa, e o interesse é identificar pontos de mudança na média. A função de custo nesse caso é a soma dos quadrados dos desvios em relação à media amostral:

$$C_1(y_{a:b}) := \sum_{h=a+1}^{b} |y_h - \bar{y}_{a:b}|^2, \tag{2.3}$$

  em que $a+1$ representa o ínicio do trecho da série, b representa o último ponto desse segmento, $y_{a:b}$ corresponde ao intervalo $(a, b] = \{a+1, ..., b\}$, ou seja, representa o segmento completo que está sendo testado pelo algoritmo e $\bar{y}_{a:b}$ é a média empírica do segmento.

- **Mudança na média e na variância:** uma extensão do caso anterior consiste em buscar mudanças tanto na média quanto na variância. Nesse caso, também assume-se que os dados seguem uma distribuição Normal e a função de custo é dada

por:

$$C_2(y_{a:b}) \;:=\; (b-a)\,\ln\widehat{\sigma}^2_{a:b} \;+\; \sum_{h=a+1}^{b} \frac{(y_h-\bar{y}_{a:b})^2}{\widehat{\sigma}^2_{a:b}}, \tag{2.4}$$

em que $a+1$ representa o ínicio do trecho da série, b representa o último ponto desse segmento,$y_{a:b}$ corresponde ao intervalo $(a,b] = \{a+1,...,b\}$, ou seja, representa o segmento completo que está sendo testado pelo algoritmo, $\bar{y}_{a:b}$ é a média empírica escalar do segmento e $\widehat{\sigma}^2_{a:b}$ é a variância empírica do segmento.

Além das funções de custo paramétricas, existem também funções de custo desenvolvidas para cenários não paramétricos, nos quais não conhecemos a distribuição a que os dados pertencem. Nesses casos, em vez de depender de suposições sobre a forma da distribuição, a ideia é avaliar como as observações se comportam dentro de cada segmento. Neste contexto, funções de custo baseadas em kernels são comumente utilizadas, uma vez que utilizam medidas de similaridade entre os pontos para quantificar quão homogêneo é um trecho da série, sem supor uma estrutura paramétrica pré-definida (Garreau e Arlot, 2017). Dessa forma, essas funções conseguem captar mudanças mais amplas na distribuição, não apenas em média ou variância. A ideia geral é transformar os dados para um espaço em que as diferenças na estrutura da série fiquem mais evidentes.

Uma função de custo amplamente utilizada dentro dessa classe é a baseada no kernel de função base radial (RBF) (Truong *et al.*, 2019), a qual pode ser escrita como:

$$C_3(y_{a:b}) = (b-a) - \frac{1}{b-a}\sum_{s,h=a+1}^{b} \exp\!\left(-\gamma\|y_s-y_h\|^2\right), \tag{2.5}$$

em que y_s e y_h representam duas observações quaisquer pertencetes ao intervalo, isto é, dois valores pertencentes ao intervalo $(a,b]$. O parâmetro γ atua como uma largura de banda do kernel, determinando a escala em que diferenças entre as observações passam a ser relevantes e, comumente, é definido como o inverso da mediana das distâncias entre todos os possíveis pares de observações.

De forma interpretativa, esse custo mede o quão dispersos os pontos transformados estão em torno de sua média no espaço de alta dimensão induzido pelo kernel. Segmentos gerados por uma mesma distribuição tendem a apresentar baixa dispersão e, portanto, menor custo, enquanto mudanças no comportamento dos dados, como alterações em média, variância ou na forma geral da distribuição, aumentam o espalhamento em $\mathcal{H}$, elevando o valor da função de custo e permitindo que pontos de mudança sejam identificados.

Em relação ao termo associado à penalização na Equação (2.1), conforme apontado por Wambui *et al.* (2015), a penalidade utilizada pelo algoritmo é dada pelo produto $\beta f(m)$, cujo objetivo é evitar o sobreajuste do modelo, isto é, evitar que o modelo encontre um número excessivo de pontos de mudança e, consequentemente, considere como mudança na estrutura da série alguma flutuação natural do comportamento dos dados. Para o termo $f(m)$, que controla como o número de pontos de mudança entra na penalização, a forma mais comum adotada na literatura é a função identidade, dada por

$$f(m) = m, \tag{2.6}$$

o que resulta em uma penalidade linear no formato βm.

Já o coeficiente β, costuma assumir principalmente a fórmula de critérios de informação, como:

- Critério de Informação de Akaike (AIC):

$$AIC = -2ln(\hat{L}) + 2k. \tag{2.7}$$

- Critério de Informação de Schwarz (BIC):

$$BIC = -2ln(\hat{L}) + kln(n). \tag{2.8}$$

- Critério de Informação Hannah-Quinn (HQ):

$$HQ = -2\ln(\hat{L}) + k\ln(ln(n)), \tag{2.9}$$

em que, no contexto de detecção de pontos de mudança,

- L: é a máxima verossimilhança do modelo.
- k: número de parâmetros em que buscamos detectar a mudança.
- n: número de observações da série temporal.

2.2 Métodos de detecção de pontos de mudança

Killick *et al.* (2012) apresenta um método de busca para a detecção de $m > 0$pontos de

mudança, podendo ser m uma quantidade de pontos conhecida ou desconhecida. Proposto por Yao (1984), o Particionamento Dinâmico, do inglês *Optimal Partitioning* (opt), tenta minimizar a função (2.1) baseando-se nos conceitos de programação dinâmica, que resolve problemas custosos computacionalmente decompondo-os em subproblemas menores, cujas soluções são armazenadas e reutilizadas, evitando recálculos desnecessários. Segundo Killick *et al.* (2012), o opt começa primeiro considerando cada possível posição para o último ponto de mudança antes do instante atual. Ele relaciona o valor ótimo da função de custo ao custo da segmentação ótima dos dados até o ponto imediatamente anterior ao último ponto de mudança, somado ao custo do segmento que vai desse ponto até o passo atual. Em outras palavras, a lógica por trás desse algoritmo é construir soluções com m mudanças a partir das soluções ótimas com $m-1$ mudanças, avançando da esquerda (início da série) para a direita ao longo dos dados: primeiro encontra-se a melhor forma de segmentar os dados até um ponto anterior com $m-1$ mudanças, e depois decide-se qual é o melhor local para formar o próximo segmento da série.

Matematicamente, podemos escrever esse problema da seguinte forma: seja $F(s)$ a minimização da Equação (2.1) para os dados $y_{1:s}$ e seja

$$T_s = \{\tau : 0 = \tau_0 < \tau_1 < \cdots < \tau_m < \tau_{m+1} = s\}$$

o conjunto de possíveis vetores de pontos de mudança para tais dados. Aqui, o índice s indica até qual observação da série estamos conduzindo a análise naquele passo. Assim, em T_s, o valor $\tau_{m+1} = s$ marca a posição em que análise atual é interrompida e não necessariamente o final da série completa $y_{1:n}$. Apenas quando tomamos $s = n$ é que esse limite coincide com o final real dos dados. Definimos também $F(0) = -\beta$. Dessa forma, segue-se que

$$F(s) = \min_{\tau \in T_s} \left(\sum_{i=1}^{m+1} \left[C(y_{(\tau_{i-1}+1):\tau_i}) + \beta m \right] \right) \tag{2.10}$$

$$= \min_{t} \left(\min_{\tau \in T_t} \sum_{i=1}^{m} \left[C(y_{(\tau_{i-1}+1):\tau_i}) + \beta m \right] + C(y_{(t+1):n}) + \beta m \right) \tag{2.11}$$

$$= \min_{t} (F(t) + C(y_{(t+1):n}) + \beta m). \tag{2.12}$$

Isso fornece uma recursividade que retorna o custo mínimo para os dados $y_{1:s}$, quando $t < s$. A Equação (2.12) pode ser resolvida de forma iterativa para $s = 1, 2, ..., n$, sendo

que em cada iteração s, o algoritmo avalia todos os possíveis pontos de segmentação $t < s$, o que resulta em um custo computacional linear em s, fazendo com que o custo computacional geral de encontrar $F(n)$ seja quadrático em n, isto é, $O(n^2)$, o que significa que o custo computacional para o algoritmo encontrar a localização ótima dos pontos de mudança em um conjunto de dados de tamanho n cresce proporcionalmente ao quadrado do tamanho do conjunto.

Esse crescimento quadrático torna-se um problema quando a quantidade de dados a serem analisados aumenta, como por exemplo quando acompanha-se uma série temporal por períodos de tempo mais longos, uma vez que, para alguns cenários, esse aumento de observações traz consigo um aumento no número de pontos de mudança, fazendo com que seja necessário encontrar uma forma mais eficiente, isto é, menos computacionalmente custosa, para o problema. É nesse sentido que Killick *et al.* (2012) propõe um método de busca que combina a lógica recursiva vista no opt com uma etapa de poda, com o intuito de reduzir o custo computacional do método.

2.2.1 *Pruned Exact Linear Time*

O algoritmo PELT, do inglês *Pruned Exact Linear Time*, desenvolvido por Killick *et al.* (2012) de forma a minimizar a função de custo (2.1) e ainda assim manter, sob certas condições, uma complexidade computacional linear em relação ao número de observações do conjunto de dados. Para isso, ele junta a minimização da função de custo (2.1) com um procedimento de poda, que consiste em remover das iterações os valores de τ que nunca resultarão no menor custo, para que então a complexidade seja reduzida, uma vez que menos recálculos são feitos.

Como formulado por Killick *et al.* (2012), o Teorema 2.13 fornece uma condição para que tal poda possa ser feita:

Teorema 2.13 *Suponha que ao introduzir um ponto de mudança em uma sequência de observações, o custo C dessa sequência é reduzido. Matematicamente, assume-se que existe uma constante K tal que para todos $t < s < T$,*

$$C(y_{(t+1):s}) + C(y_{(s+1):T}) + K \leq C(y_{(t+1):T}). \tag{2.14}$$

Dessa forma, se

$$F(t) + C(y_{(t+1):s}) + K \geq F(s), \tag{2.15}$$

para algum tempo futuro $T > s$, t nunca poderá ser o último ponto de mudança ótimo antes de T.

A lógica por trás desse teorema é a de que, uma vez que sabe-se que uma segmentação terminando em s é melhor do que qualquer segmentação que termine em t, dado o custo extra de continuar a partir de t, então não vale a pena considerar t novamente em cálculos futuros, o que permite que o algoritmo possa descartar esses valores.

Matematicamente, é possível demonstrar o Teorema 2.13 reescrevendo os termos das equações (2.14) e (2.15) de forma que se compare diretamente os custos acumulados de segmentações terminando em t e em s. Assumindo que a Desigualdade (2.15) vale:

$$F(t) + C(y_{(t+1):s}) + K \geq F(s) + . \tag{2.16}$$

Somando $C(y_{(s+1):T})$ dos dois lados:

$$F(t) + C(y_{(t+1):s}) + K + C(y_{(s+1):T}) \geq F(s) + C(y_{(s+1):T}). \tag{2.17}$$

Agora, aplicando (2.14) no termo $C(y_{(t+1):s}) + C(y_{(s+1):T}) + K \leq C(y_{(t+1):T})$:

$$F(t) + C(y_{(t+1):T}) \geq F(s) + C(y_{(s+1):T}), \tag{2.18}$$

ou seja, o custo total para uma segmentação que termina em t é maior do que o custo de uma segmentação terminando em s. Logo, t não pode ser o melhor ponto de mudança anterior a T, podendo ser podado. Em outras palavras, a ideia desta etapa de poda é verificar se o custo não penalizado de utilizar t como último ponto de mudança antes de s é maior do que o custo ótimo (o qual é penalizado) de s. Se isso acontece, então t nunca será a melhor no futuro e pode ser podado.

2.3 Detecção de pontos de mudança em séries temporais

Como apresentado na Seção 2.1, em contextos paramétricos, as principais funções de custo baseiam-se na estimação de máxima verossimilhança associada aos dados e, nesses casos, os pontos observados são considerados como sendo independentes e advindos de

uma família de distribuições conhecida. No entanto, para o contexto em que os dados estudados são provenientes de uma série temporal, essa suposição dificilmente acontece na prática, uma vez que essas observações frequentemente possuem algum grau de dependência em relação às suas observações passadas, isto é, as observações são autocorrelacionadas. Para superar essa incompatibilidade entre a suposição teórica e a prática, muitas vezes é necessário um fazer tratamento prévio da série antes de detectarmos as mudanças.

Uma das técnicas utilizadas nesse contexto é conhecida como pré-branqueamento, do inglês, *pre-whitening* (Serinaldi e Kilsby, 2016). Seu objetivo é transformar a série original em uma nova série que se assemelhe a um ruído branco, isto é, uma sequência de observações não correlacionadas. O procedimento mais comum para isso consiste em ajustar um modelo autorregressivo integrado de médias móveis (ARIMA) aos dados. Se houver pontos de mudança na estrutura da série, como por exemplo alguma alteração no processo gerador das observações após certo instante, essa mudança se refletirá nos resíduos, pois ao ajustar um único modelo a trechos com comportamentos distintos, os resíduos tendem a evidenciar essa discrepância. Uma vez obtida a série de resíduos, que até certo ponto satisfaz a suposição de ausência de autocorrelação, podemos aplicar funções de custo paramétricas nos algoritmos de detecção. Dessa forma, detectar mudanças nos resíduos pode indicar a presença de quebras na série temporal original.

Outra estratégia para reduzir a autocorrelação é diferenciar a série, considerando $y'_t = y_t - y_{t-1}$, em que y_t representa o valor da série no instante t e y_{t-1} é a observação imediatamente anterior no tempo. Em muitos casos, as observações consecutivas tendem a ser mais correlacionadas entre si do que observações distantes, e a diferenciação diminui até certo ponto essa dependência, aproximando a série de um processo não correlacionado.

Por fim, como mencionado na Seção 2.1, quando as suposições paramétricas não são atendidas ou quando a distribuição dos dados é desconhecida, também é possível recorrer a funções de custo não paramétricas, que não exigem suposições sobre a distribuição dos dados e, em geral, possuem suposições mais flexíveis em relação à independência das observações.

Capítulo 3

Simulações

Com o objetivo de avaliar o desempenho do algoritmo na detecção de pontos de mudança, conduzimos um estudo de simulação de séries temporais com pontos de mudança sob três cenários diferentes, a fim de verificar se o algoritmo seria capaz de recuperar os pontos de mudança previamente definidos por nós. Em cada cenário, simulamos 1000 séries temporais provenientes de um processo *ARIMA(1,1,0)*, variando tanto o parâmetro autorregressivo quanto o número de pontos de mudança. Nesse processo gerador de dados, o termo $AR(1)$ indica que os dados possuem correlação de primeira ordem, isto é, cada valor está autocorrelacionado com o valor imediatamente anterior. Já o segundo termo $I(1)$ significa que a série passa por uma diferenciação, o que está associado à presença de tendência ao longo do tempo, podendo essa tendência ser crescente ou decrescente. Optamos por gerar dados autocorrelacionados e com tendência por se tratarem de características comuns em séries financeiras, que são justamente o tipo de dado real que pretendemos analisar na aplicação.

Tendo as observações geradas, o algoritmo foi aplicado para detectar alterações na média e na variância da série, isto é, para identificar o momento em que o processo gerador dos dados se modificou. Para isso, utilizamos a função de custo RBF, que é uma medida baseada em kernel capaz de capturar mudanças na distribuição dos dados e foi explorada na Seção 2.1, e três critérios de penalização: AIC (*Akaike Information Criterion*), BIC (*Bayesian Information Criterion*) e HQ (*Hannan Quinn*).

A fórmula geral desses critérios, para um modelo com k parâmetros e n observações, é dada por:

- $\text{AIC} = -2\ln(\hat{L}) + 2k$;

- BIC $= -2\ln(\hat{L}) + k\ln(n)$;
- HQ $= -2\ln(\hat{L}) + k\ln(ln(n))$.

Como no contexto deste trabalho não trabalhamos diretamente com a log-verossimilhança, pois utilizamos a função de custo RBF, utilizamos apenas os termos de penalização associados a cada critério. Nesse caso, o valor de k representa o número de parâmetros da série em que estamos buscando modificações, e foi fixado em $k = 2$, pois buscamos identificar mudanças na média e na variância da série. Além disso, para a função $f(m)$ seguimos com a função identidade $f(m) = m$. vamos considerar m como sendo desconhecido, isto é, não vamos atribuir previamente um número definido de pontos de mudança, para que possamos verificar também quantos pontos o algoritmo irá encontrar.

As penalidades utilizadas foram, portanto:

- $pen_{BIC} = 2 \cdot \ln(n) \cdot m$.
- $pen_{HQ} = 2 \cdot 2 \cdot \ln(\ln(n)) \cdot m$.
- $pen_{AICmodificado} = 2 \cdot 2 \cdot \ln(n) \cdot m$.

No caso do AIC, cuja penalidade original seria $2k$, optamos por fazer um ajuste na fórmula, incluindo o termo $\ln(n)$. Essa modificação foi feita para evitar que o AIC se tornasse muito brando na penalização, o que poderia levar à detecção de um número excessivo de pontos, e também para manter a escala da penalidade mais alinhada às demais (BIC e HQ), permitindo comparações mais equilibradas entre os critérios. Desta forma, vamos chamar a penalidade utilizada de AIC modificado.

Para cada cenário e penalidade, comparamos os pontos de mudança detectados pelo algoritmo com os pontos reais previamente conhecidos e analisamos o desempenho do algoritmo de três formas:

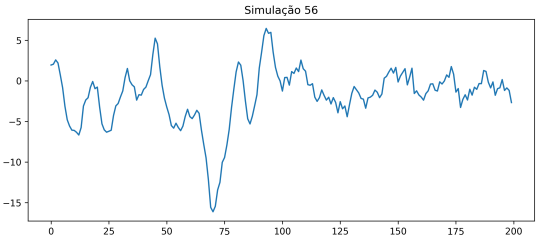
- Acerto exato: quando o algoritmo identificou apenas os pontos de mudança verdadeiros;
- Acerto parcial: quando todos os pontos de mudança reais foram encontrados, mas outros pontos adicionais também foram considerados pontos de mudança ('falsos positivos'). aqui, não contabiliza-se as vezes em que o algoritmo encontrou apenas os pontos de mudança reais, ou seja, os acertos exatos não são contabilizados.

- Acerto com banda de tolerância: quando o algoritmo indicou ao menos um ponto de mudança dentro de uma faixa de tolerância de 5 índices antes e 5 índices depois do ponto verdadeiro, sem identificar nenhum outro ponto fora dessa faixa e sem ter identificado nenhum ponto de mudança real.

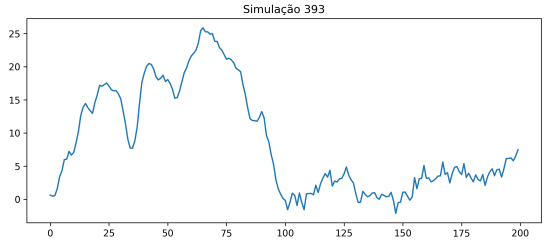
3.0.1 Cenário 1 – Um ponto de mudança em $t = 100$

No primeiro cenário, foram simuladas duas séries temporais autoregressivas de ordem 1 ($AR(1)$) de 100 observações cada, sendo a primeira série gerada com parâmetro $AR = 0,7$ e a segunda com parâmetro $AR = 0,3$. Em seguida, as duas séries foram concatenadas, resultando em uma única série de comprimento 200. Esse procedimento foi repetido 1000 vezes, resultando em 1000 séries temporais de 200 observações, cada uma contendo um ponto de mudança no índice $t = 100$, correspondente à transição entre os dois processos autorregressivos.

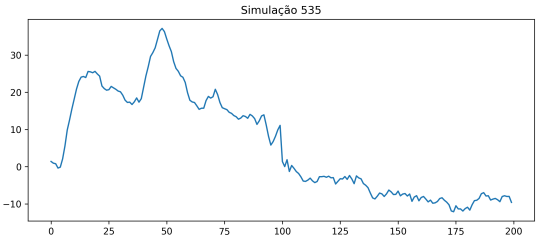
A Figura 3.1 ilustra quatro exemplos de séries temporais simuladas neste cenário, cada uma com comportamentos distintos ao longo do tempo e variações de média e variância. Podemos notar que, antes e depois do ponto de mudança previamente conhecido, as séries apresentam padrões diferentes: trechos iniciais mais voláteis ou estáveis podem ser seguidos por períodos com nível médio mais elevado ou reduzido, o que indica uma alteração na distribuição das séries, ilustrando justamente o tipo de mudança que buscamos detectar com o algoritmo.



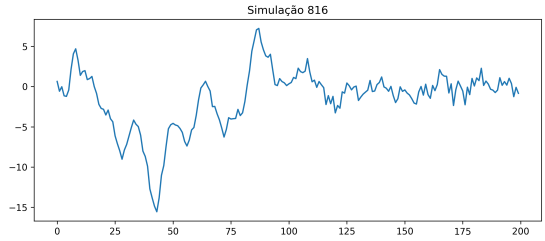
(a) Cenário 1 - Simulação 56



(b) Cenário 1 - Simulação 393



(c) Cenário 1 - Simulação 535



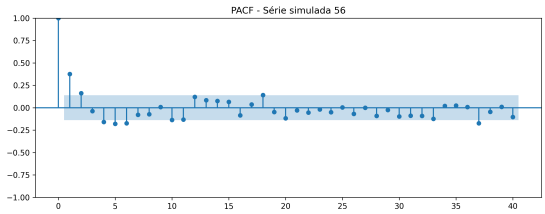
(d) Cenário 1 - Simulação 816

Figura 3.1: Exemplo de quatro séries simuladas no cenário 1.

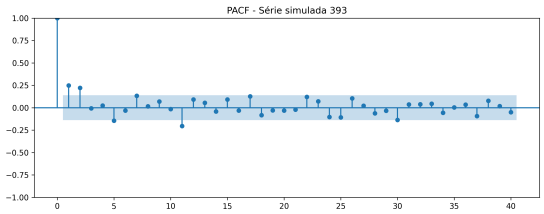
Na Figura 3.2, apresentamos os gráficos da função de autocorrelação parcial (PACF) das séries diferenciadas para quatro simulações do Cenário 1. Podemos observar que as quatro séries possuem um padrão muito parecido: há um pico significativo apenas no primeiro lag, enquanto os demais permanecem baixos. Esse comportamento está de acordo com o que se espera de um processo autorregressivo de primeira ordem, e o fato

de todas as séries apresentarem o mesmo padrão sugere que os dados simulados vieram do mesmo processo gerador.

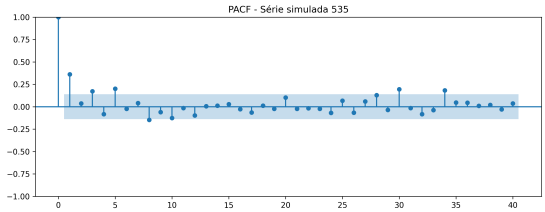
Utilizamos o gráfico de PACF porque ele mostra, em cada passo no tempo, quanto o passado ainda ajuda a explicar a série, depois de descontar o efeito dos lags anteriores. Quando diferentes simulações exibem um PACF com o mesmo padrão, isso sugere que todas seguem o mesmo tipo de relação com o próprio passado, ou seja, indica que todas têm o mesmo comportamento em relação ao processo que gerou os dados.



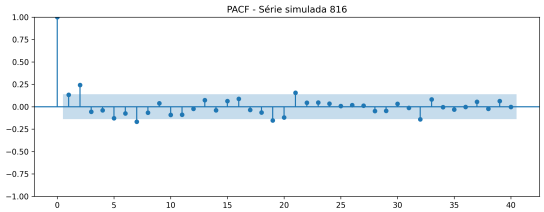
(a) Cenário 1 - Função de autocorrelação parcial da série diferenciada na simulação 56



(b) Cenário 1 - Função de autocorrelação parcial da série diferenciada na simulação 393



(c) Cenário 1 - Função de autocorrelação parcial da série diferenciada na simulação 535



(d) Cenário 1 - Função de autocorrelação parcial da série diferenciada na simulação 816

Figura 3.2: Gráficos PACF de quatro séries simuladas no cenário 1.

Na Tabela 3.1, apresentamos os resultados obtidos para cada critério de penalização - AIC modificado, BIC e HQ. Observando os resultados, notamos que o AIC modificado foi o critério com melhor desempenho em acertos exatos, identificando corretamente o ponto de mudança em 225 das 1000 simulações (proporção de $0,225$). No que diz respeito aos acertos parciais, o critério HQ apresentou a maior proporção ($0,794$), porém isso ocorre às custas de um desempenho muito baixo em acertos exatos: apenas 2 simulações identificaram o ponto correto. Ou seja, embora o HQ encontre o ponto verdadeiro com frequência, ele tende a adicionar muitos pontos falsos-positivos. Considerando os acertos dentro da banda de tolerância, o AIC modificado novamente se destaca, encontrando pontos dentro do intervalo permitido em 24 simulações. Em comparação, o BIC apresentou apenas 5 acertos dentro da banda, e o HQ apenas 1.

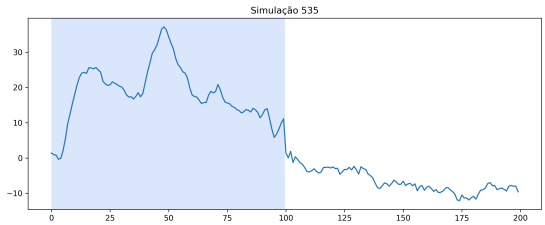
De forma geral, observamos que o AIC modificado foi o critério com melhor desem-

penho neste cenário, por reunir o maior número de acertos exatos, o maior número de acertos dentro da banda de tolerância e, ao mesmo tempo, apresentar a menor proporção de acertos parciais, estes últimos associados à presença de falsos positivos.

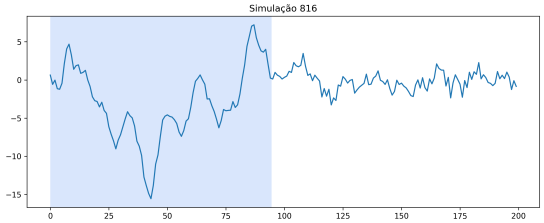
Tabela 3.1: Resultados das simulações do cenário 1

Critério	Acertos Exatos	Prop. Acertos Exatos	Acertos Parciais	Prop. Acertos Parciais	Acertos com Tolerância	Prop. Acertos com Tolerância
AIC Modificado	225	0.225	584	0.584	24	0.024
BIC	30	0.03	733	0.733	5	0.005
HQ	2	0.002	794	0.794	1	0.001

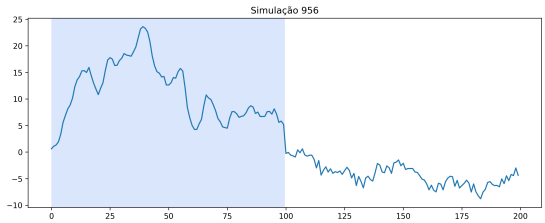
Na Figura 3.3, apresentamos alguns exemplos de pontos de mudança identificados de forma exata e dentro da banda de tolerância para cada uma das penalidades testadas. Nos casos de acerto exato (subfiguras a, c e e), podemos observar que o ponto encontrado coincide com a mudança no comportamento da série, marcada por uma alteração na média e na variância dos dados. Já nos casos de acerto dentro da banda de tolerância (subfiguras b, d e f), é possível notar que o ponto identificado está muito próximo da quebra no comportamento da série, ainda que não esteja exatamente sobre ela. Mesmo assim, o algoritmo reconhece a região de transição da série, indicando que o método é capaz de capturar a mudança estrutural, ainda que com um pequeno deslocamento.



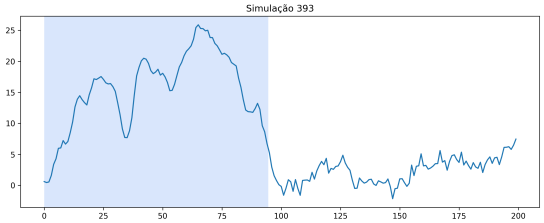
(a) Ponto de mudança exato encontrado na simulação 535 utilizando a penalidade BIC.



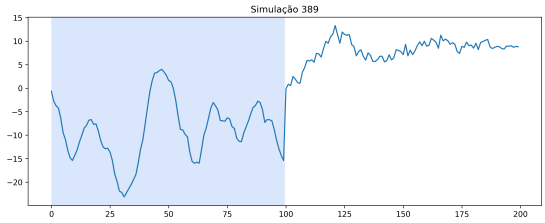
(b) Ponto de mudança encontrado na banda de tolerância na simulação 816 utilizando a penalidade BIC.



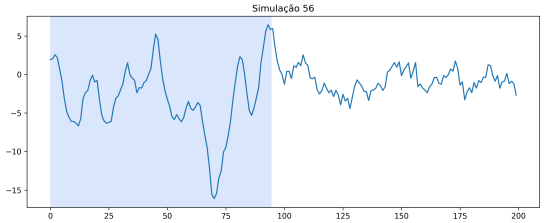
(c) Ponto de mudança exato encontrado na simulação 956 utilizando a penalidade AIC modificada.



(d) Ponto de mudança encontrado na banda de tolerância na simulação 393 utilizando a penalidade AIC modificada.



(e) Ponto de mudança exato encontrado na simulação 389 utilizando a penalidade HQ.



(f) Ponto de mudança encontrado na banda de tolerância na simulação 56 utilizando a penalidade HQ.

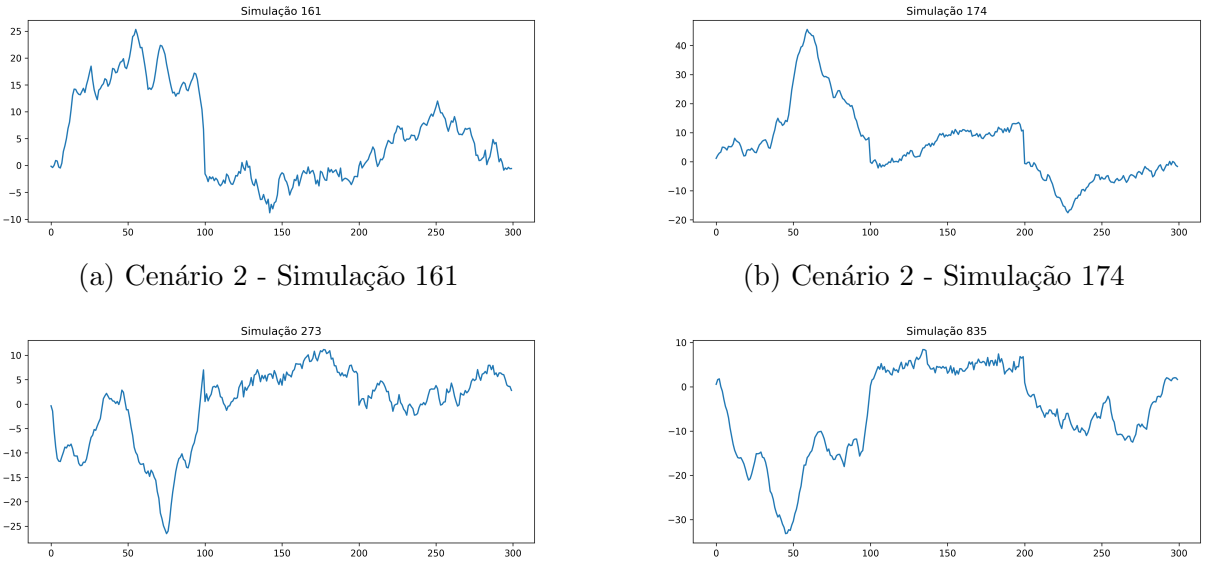
Figura 3.3: Ilustração de alguns pontos de mudança encontrados em algumas simulações do cenário 1 com as diferentes penalidades utilizadas.

3.0.2 Cenário 2 – Dois pontos de mudança em $t = 100$ e $t = 200$

Já no segundo cenário, simulamos três séries temporais autoregressivas de ordem 1 ($AR(1)$) de 100 observações cada, sendo a primeira série gerada com parâmetro $AR = 0,7$, a segunda com parâmetro $AR = -0,3$ e a terceira com parâmetro $AR = 0,2$. Essas três séries foram concatenadas, resultando em uma única série de comprimento 300. Esse procedimento foi repetido 1000 vezes, resultando em 1000 séries temporais de 300 observações, cada uma contendo um ponto de mudança no índice $t = 100$ e um outro ponto no índice $t = 200$, correspondente à transição entre os três processos autorregressivos.

Na Figura 3.4, apresentamos quatro exemplos de séries temporais simuladas no segundo cenário, cada uma exibindo comportamentos distintos ao longo do tempo, com mudanças na média e na variância dos dados. É possível observar que, antes e depois

dos pontos de mudança previamente definidos, nos índices 100 e 200, as séries passam a apresentar padrões diferentes em cada um dos três trechos. Isso evidencia alterações na distribuição dos dados ao longo do tempo e ilustra o tipo de quebra estrutural que buscamos identificar com o algoritmo, reforçando a presença de três comportamentos distintos dentro de cada série.



(a) Cenário 2 - Simulação 161

(b) Cenário 2 - Simulação 174

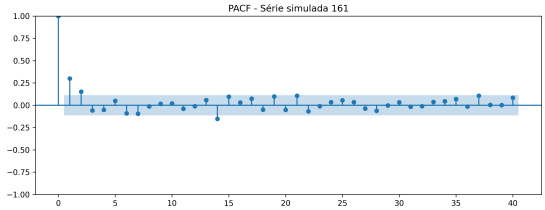
(c) Cenário 2 - Simulação 273

(d) Cenário 2 - Simulação 835

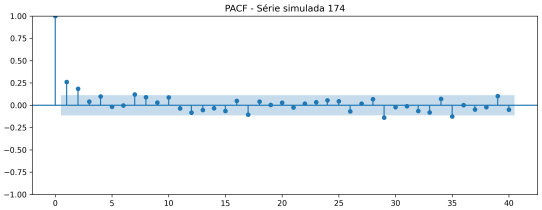
Figura 3.4: Exemplo de quatro séries simuladas no cenário 2.

Na Figura 3.5 são apresentados os gráficos da função de autocorrelação parcial das séries diferenciadas para quatro simulações do Cenário 2. Assim como no cenário anterior, as quatro séries exibem um padrão bastante semelhante: há um pico mais marcado apenas no primeiro lag, enquanto as demais autocorrelações parciais se mantêm baixas. Isso está de acordo com o que se espera de um processo autorregressivo de primeira ordem e, como o mesmo comportamento aparece em todas as simulações, reforça a ideia de que as séries do Cenário 2 também foram geradas pelo mesmo tipo de processo dentro deste cenário.

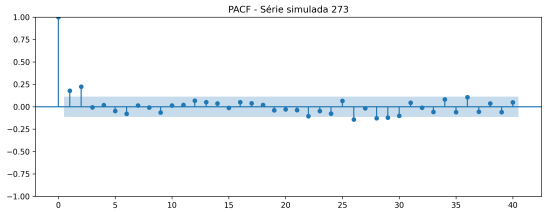
Na Tabela 3.2, apresentamos os resultados obtidos para cada critério de penalização testado no cenário 2. Observando a tabela, nota-se que o AIC modificado foi o critério com maior número de acertos exatos, identificando corretamente o ponto de mudança em 28 das 1000 simulações, enquanto o BIC apresentou apenas 2 acertos e o HQ não registrou nenhum. Em relação aos acertos parciais, o comportamento se inverte: o critério HQ foi o que mais apresentou esse tipo de resultado, com 331 ocorrências, seguido do BIC com 200 e do AIC modificado com 67. Esses acertos parciais indicam que o método encontrou o ponto verdadeiro, mas também adicionou pontos de mudança que não existiam, ou seja,



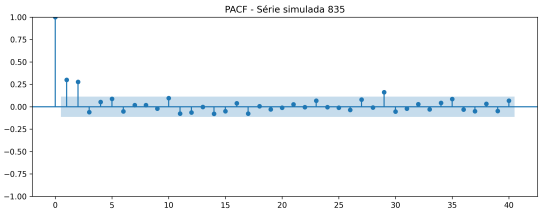
(a) Cenário 2 - Função de autocorrelação parcial da série diferenciada na simulação 161



(b) Cenário 2 - Função de autocorrelação parcial da série diferenciada na simulação 174



(c) Cenário 2 - Função de autocorrelação parcial da série diferenciada na simulação 273



(d) Cenário 2 - Função de autocorrelação parcial da série diferenciada na simulação 835

Figura 3.5: Gráficos PACF de quatro séries simuladas no cenário 2.

detectou pontos que podem ser considerados falsos positivos. Já ao considerar os acertos dentro da banda de tolerância, o AIC modificado novamente se destaca, com 49 simulações em que o ponto detectado ficou dentro do intervalo permitido. O BIC apresentou apenas 4 acertos dentro da banda e o HQ não registrou nenhum.

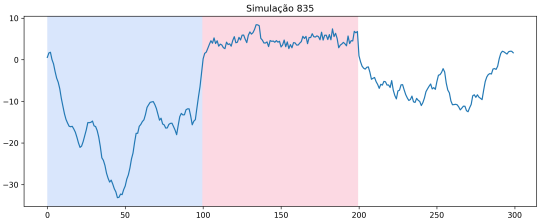
De forma geral, assim como no cenário anterior, o AIC modificado apresentou o melhor desempenho, reunindo o maior número de acertos exatos, o maior número de acertos dentro da banda de tolerância e, simultaneamente, o menor número de acertos parciais — o que indica menor ocorrência de falsos positivos.

Tabela 3.2: Resultados das simulações do cenário 2

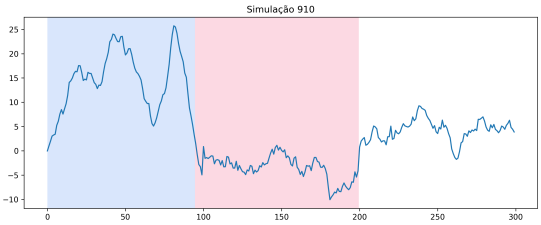
Critério	Acertos Exatos	Prop. Acertos Exatos	Acertos Parciais	Prop. Acertos Parciais	Acertos com Tolerância	Prop. Acertos com Tolerância
AIC Modificado	28	0.028	67	0.067	49	0.049
BIC	2	0.002	200	0.200	4	0.004
HQ	0	0.00	331	0.331	0	0.00

Na Figura 3.6, apresentamos exemplos de pontos de mudança identificados de forma exata e dentro da banda de tolerância ao longo das séries simuladas. Nos casos de acerto exato (subfiguras a e c), observamos que os pontos detectados coincidem com as transições de comportamento da série, marcadas por alterações na média e na variância após os instantes 100 e 200. Já nas subfiguras que ilustram acertos dentro da banda de tolerância (b, d, e e f), é possível notar que os pontos identificados se encontram muito próximos das quebras reais, ainda que não exatamente sobre elas. Além disso, embora a penali-

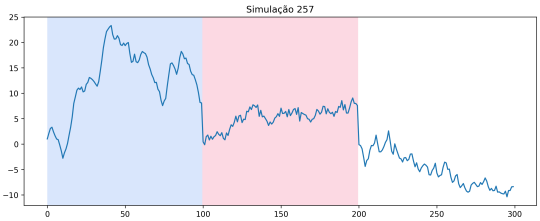
dade HQ não tenha identificado simultaneamente os dois pontos de mudança verdadeiros em nenhuma simulação, as imagens mostram que, em alguns casos, um dos pontos foi localizado exatamente e o outro foi encontrado, ao menos, dentro da banda de tolerância.



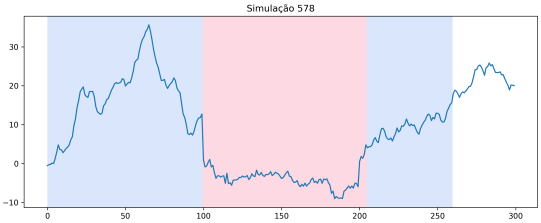
(a) Ponto de mudança exato encontrado na simulação 835 utilizando a penalidade BIC.



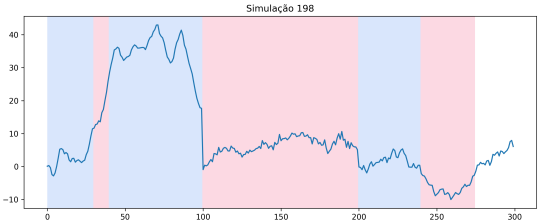
(b) Ponto de mudança encontrado na banda de tolerância na simulação 910 utilizando a penalidade BIC.



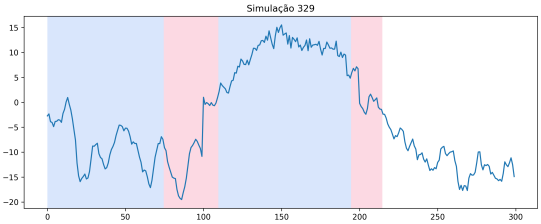
(c) Ponto de mudança exato encontrado na simulação 257 utilizando a penalidade AIC modificada.



(d) Ponto de mudança encontrado na banda de tolerância na simulação 578 utilizando a penalidade AIC modificada.



(e) Ponto de mudança encontrado na banda de tolerância na simulação 198 utilizando a penalidade HQ.



(f) Ponto de mudança encontrado na banda de tolerância na simulação 329 utilizando a penalidade HQ.

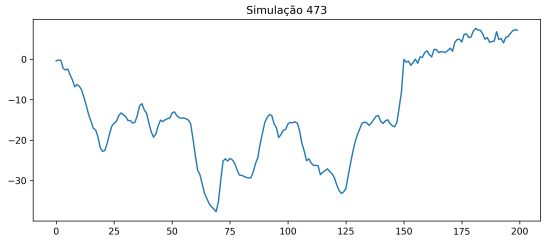
Figura 3.6: Ilustração de alguns pontos de mudança encontrados em algumas simulações do cenário 2 com as diferentes penalidades utilizadas.

3.0.3 Cenário 3 – Um ponto de mudança em $t = 150$

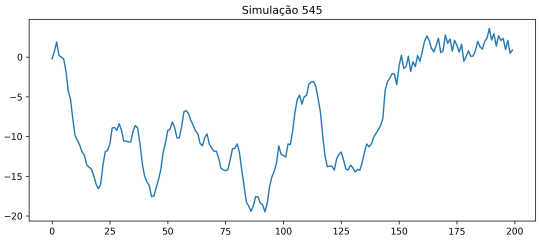
Por fim, no terceiro cenário, foram simuladas 2 séries temporais autoregressivas de ordem 1 ($AR(1)$), sendo a primeira com parâmetro $AR = 0,7$ e tamanho 150, e a segunda com parâmetro $AR = -0,3$ e tamanho 50. Essas séries foram concatenadas, resultando em uma única série de tamanho 200. Assim como nos cenários anteriores, esse procedimento foi repetido 1000 vezes, resultando em 1000 séries temporais de 200 observações, cada uma contendo um ponto de mudança no índice $t = 150$, correspondente à transição entre

os dois processos autorregressivos.

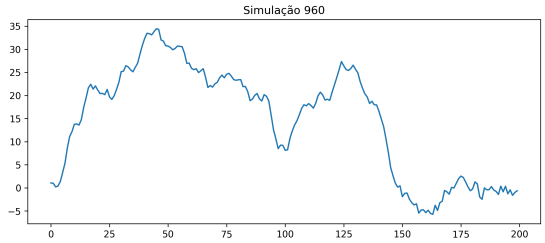
A Figura 3.7 mostra quatro exemplos de séries simuladas no cenário 3, todas com um ponto de mudança no índice 150. Em todas as séries, podemos observar que o comportamento antes do ponto é diferente do comportamento após ele: algumas séries apresentam maior oscilação no início e tornam-se mais estáveis depois, enquanto outras mudam de nível médio ou de tendência. Essas diferenças visíveis indicam uma alteração na distribuição dos dados, o que ilustra justamente o tipo de mudança que o algoritmo deve ser capaz de identificar.



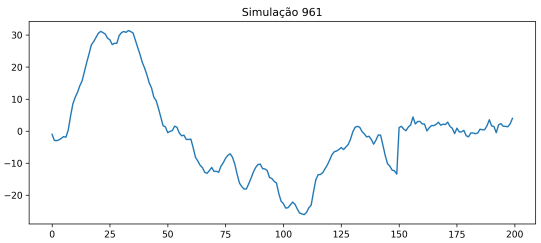
(a) Cenário 3 - Simulação 473



(b) Cenário 3 - Simulação 545



(c) Cenário 3 - Simulação 960

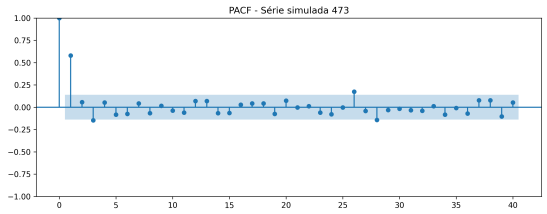


(d) Cenário 3 - Simulação 961

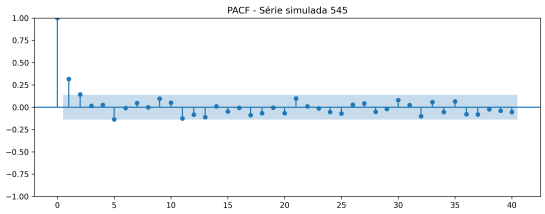
Figura 3.7: Exemplo de quatro séries simuladas no cenário 3.

Na Figura 3.8 vemos os gráficos da função de autocorrelação parcial das séries diferenciadas para quatro simulações do Cenário 3. Assim como nos cenários anteriores, as séries apresentam um desenho muito parecido: aparece um pico mais forte apenas no primeiro lag e os demais valores ficam próximos de zero. Esse padrão é compatível com um processo autorregressivo de primeira ordem e, por surgir de forma semelhante em todas as simulações, novamente reforça a ideia de que as séries do Cenário 3 também foram geradas pelo mesmo tipo de processo dentro desse cenário.

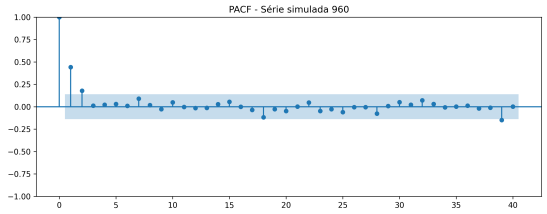
Na Tabela 3.3, apresentamos os resultados obtidos pelos três critérios de penalização no cenário 3. Observando os valores, podemos notar que o AIC modificado novamente foi o critério com melhor desempenho em acertos exatos, identificando corretamente o ponto de mudança em 87 das 1000 simulações. O BIC apresentou apenas 9 acertos exatos, enquanto o HQ não registrou nenhum. Em relação aos acertos parciais, o HQ foi o que



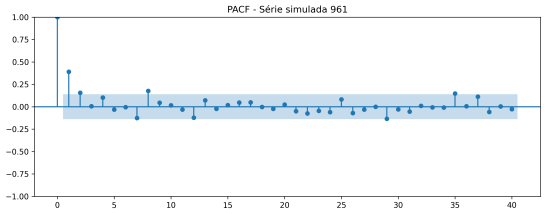
(a) Cenário 3 - Função de autocorrelação parcial da série diferenciada na simulação 473



(b) Cenário 3 - Função de autocorrelação parcial da série diferenciada na simulação 545



(c) Cenário 3 - Função de autocorrelação parcial da série diferenciada na simulação 960



(d) Cenário 3 - Função de autocorrelação parcial da série diferenciada na simulação 961

Figura 3.8: Gráficos PACF de quatro séries simuladas no cenário 3.

mais acumulou ocorrências (814), seguido pelo BIC (753) e pelo AIC modificado (610). Assim como nos cenários anteriores, esses resultados indicam que o HQ e o BIC tendem a encontrar pontos adicionais além do verdadeiro, refletindo uma maior incidência de falsos positivos. Considerando os acertos dentro da banda de tolerância, o AIC modificado mais uma vez apresentou o melhor resultado, com 8 simulações nas quais o ponto encontrado esteve próximo ao verdadeiro. O BIC apresentou 4 acertos na banda, enquanto o HQ não identificou nenhum ponto dentro do intervalo definido.

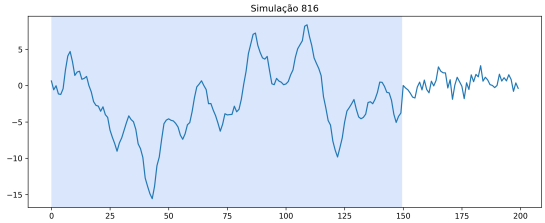
De forma geral, assim como nos cenários anteriores, o AIC modificado foi o critério que apresentou o desempenho mais consistente, combinando maior número de acertos exatos, maior número de acertos na banda de tolerância e menor quantidade de acertos com falsos positivos.

Tabela 3.3: Resultados das simulações do cenário 3

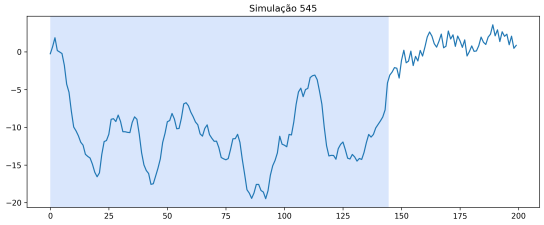
Critério	Acertos Exatos	Prop. Acertos Exatos	Acertos Parciais	Prop. Acertos Parciais	Acertos com Tolerância	Prop. Acertos com Tolerância
AIC Modificado	87	0.087	610	0.610	8	0.008
BIC	9	0.009	753	0.753	4	0.004
HQ	0	0.00	814	0.814	0	0.00

Na Figura 3.9, apresentamos exemplos de pontos de mudança identificados pelos diferentes critérios de penalização no cenário 3. Nos casos de acerto exato (subfiguras a e c), é possível observar que o ponto detectado coincide com a alteração mais evidente no comportamento da série por volta do índice 150. Já nas subfiguras que ilustram acertos

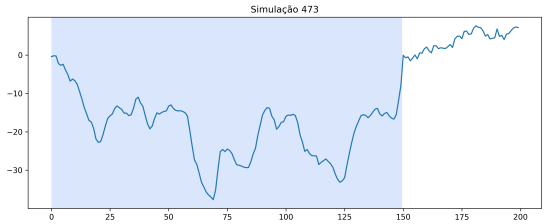
dentro da banda de tolerância (b, d, e e f), notamos que os pontos encontrados estão próximos da região em que ocorre a transição dos regimes dos dados. Por fim, é possível verificar os pontos detectados quando utilizamos a penalidade HQ, sendo possível ver que o ponto verdadeiro é detectado, mas apenas juntamente com outros pontos considerados falsos positivos.



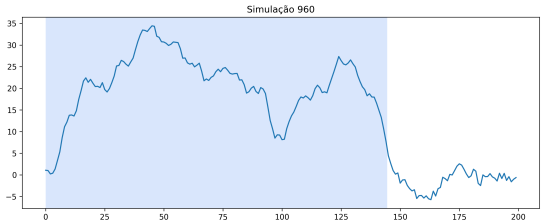
(a) Ponto de mudança exato encontrado na simulação 816 utilizando a penalidade BIC.



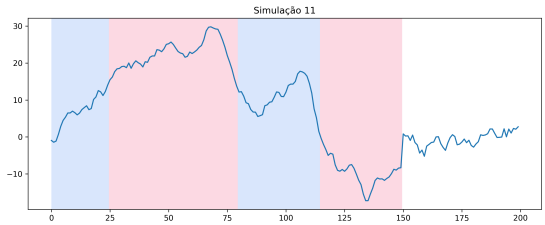
(b) Ponto de mudança encontrado na banda de tolerância na simulação 545 utilizando a penalidade BIC.



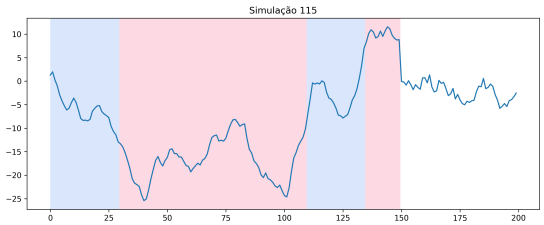
(c) Ponto de mudança exato encontrado na simulação 473 utilizando a penalidade AIC modificada.



(d) Ponto de mudança encontrado na banda de tolerância na simulação 960 utilizando a penalidade AIC modificada.



(e) Ponto de mudança encontrado na banda de tolerância na simulação 11 utilizando a penalidade HQ.



(f) Ponto de mudança encontrado na banda de tolerância na simulação 115 utilizando a penalidade HQ.

Figura 3.9: Ilustração de alguns pontos de mudança encontrados em algumas simulações do cenário 3 com as diferentes penalidades utilizadas.

3.0.4 Conclusão - Simulações

De forma geral, os resultados dos três cenários mostraram um padrão no comportamento das penalidades avaliadas. O AIC modificado destacou-se de maneira consistente, sendo o critério que mais vezes encontrou o ponto de mudança correto e que identificou melhor a região exata da mudança no comportamento das séries, além de apresentar menos falsos positivos. O BIC acertou com menos frequência, mas ainda teve um desempenho razoável quando a mudança era mais evidente. Já o HQ mostrou-se mais sensível às oscilações da série, muitas vezes detectando a área certa, mas ao custo de marcar pontos a mais, o que resultou em muitos acertos parciais, ou seja, muitos acertos com falsos positivos. Também observamos que todas as penalidades tiveram mais dificuldade nos cenários em que havia dois pontos de mudança e quando a quebra ocorria mais ao final da série (como no índice 150 do cenário 3), em comparação às situações em que a mudança estava mais centralizada. Esses resultados indicam que, nestas simulações, o AIC modificado foi o critério que apresentou o desempenho mais equilibrado diante das diferentes estruturas simuladas.

Capítulo 4

Aplicação

O Índice Bovespa (Ibovespa) é o principal indicador do desempenho médio das ações negociadas na B3 (Bolsa de Valores Brasileira) e reflete as expectativas, incertezas e movimentos do mercado financeiro brasileiro. Construído como uma carteira composta pelos ativos mais negociados da bolsa, o índice reúne informações sobre liquidez, perspectivas macroeconômicas, confiança dos investidores e crises externas que afetam a economia e o ambiente corporativo. Por reunir empresas de diversos setores, ele funciona como um termômetro da atividade econômica e das condições financeiras do país, respondendo a eventos políticos, crises econômicas, mudanças regulatórias e acontecimentos internacionais relevantes.

Diante disso, analisar o comportamento histórico desse índice é uma forma de compreender como o mercado reage a diferentes tipos de acontecimentos. Oscilações abruptas no índice frequentemente refletem mudanças estruturais na economia ou no ambiente institucional, como crises econômicas, rupturas políticas, alterações na política monetária ou eventos globais de grande impacto. Entre janeiro de 2005 e outubro de 2025, período que será analisado nesse estudo, o Brasil e o mundo passaram por acontecimentos que deixaram marcas claras nos mercados financeiros: a crise financeira global de 2008, que provocou quedas generalizadas nas bolsas; a intensificação da crise econômica e política brasileira a partir de 2013–2015; e, mais recentemente, o impacto da pandemia de COVID-19 em 2020. Dessa forma, é razoável esperar que tais eventos gerem alterações na dinâmica do Ibovespa, potencialmente identificáveis como pontos de mudança quando olhamos para sua série histórica.

Identificar automaticamente esses pontos é importante porque permite compreender de maneira 'quantitativa' como o mercado reage a esses diferentes choques e quais períodos

marcam mudanças significativas no comportamento do índice. Estudos desse tipo auxiliam na interpretação da estrutura temporal dos preços, no reconhecimento de comportamentos distintos de volatilidade ou tendência e no entendimento dos fatores que influenciam o mercado ao longo do tempo. Além disso, métodos de detecção de pontos de mudança podem ser úteis em contextos de previsão, gestão de risco e modelagem financeira, pois ajudam a identificar períodos em que a dinâmica do mercado se altera.

Dessa forma, vamos aplicar o algoritmo PELT na série de dados do valor de fechamento semanal do Ibovespa no período de janeiro de 2005 a outubro de 2025, com o objetivo de verificar se o método é capaz de identificar pontos de mudança associados a acontecimentos econômicos relevantes. Utilizamos a mesma função de custo empregada nas simulações (a função RBF), assim como os critérios de penalização AIC modificado, BIC e Hannan–Quinn, permitindo comparar o comportamento do algoritmo em dados reais com os padrões observados na seção de simulações. A análise busca verificar se esses critérios são sensíveis às mudanças estruturais presentes na série do Ibovespa e se o PELT é eficiente na detecção de rupturas associadas a choques econômicos e financeiros ao longo das duas últimas décadas.

Na Figura 4.1, apresentamos os dados do fechamento semanal do Ibovespa entre fevereiro de 2005 e outubro de 2025. De maneira geral, podemos observar uma tendência de crescimento ao longo do período, intercalada por episódios de volatilidade. Entre 2007 e 2009, notamos uma queda acentuada que pode estar associada à crise financeira global; nos anos seguintes, especialmente entre 2013 e 2016, o índice apresenta oscilações prolongadas provavelmente compatíveis com o período de crise econômica e instabilidade política no Brasil. Já em 2020, vemos movimentos mais abruptos na série, provavelmente correspondentes ao choque da pandemia de COVID-19, seguidos por uma recuperação e retomada do movimento de alta. A partir de 2021, verificamos um padrão de oscilação mais forte, sugerindo alternância entre momentos de otimismo e preocupação no cenário macroeconômico.

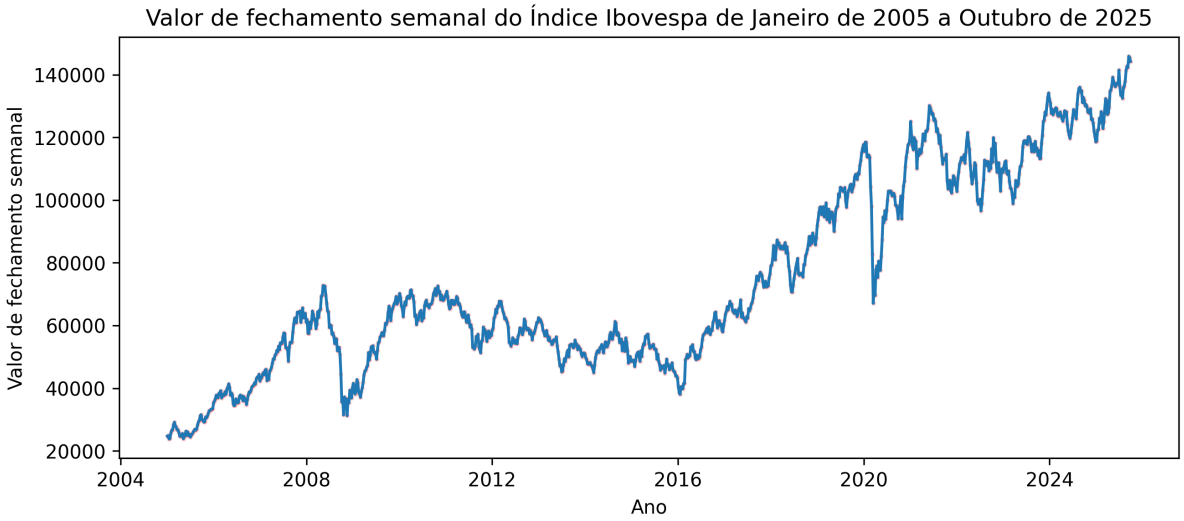


Figura 4.1: Série temporal do valor de fechamento semanal do Índice Ibovespa de Janeiro de 2005 a Outubro de 2025

Na Tabela 4.1 e na Figura 4.2, apresentamos os pontos de mudança detectados na série de acordo com cada uma das penalidades testadas. A partir da análise conjunta, podemos observar que as três penalidades identificaram alguns momentos em comum como pontos de mudança: março de 2007, agosto de 2017, dezembro de 2018 e novembro de 2023. Esses anos correspondem a períodos de maior instabilidade econômica e política, o que torna plausível que a série apresente mudanças de comportamento nessas regiões. Visualmente, é possível observar que nesses intervalos o Ibovespa altera sua tendência ou passa a oscilar com mais intensidade, o que pode justificar a detecção desses pontos pelo PELT.

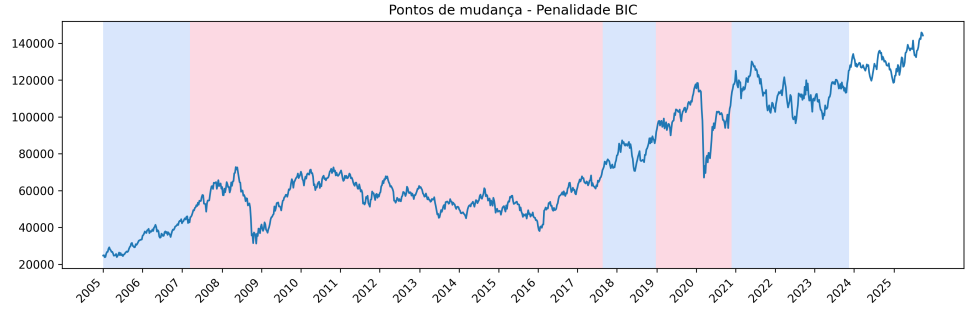
O critério HQ, assim como observado nas simulações, apresentou um comportamento mais sensível, detectando um número maior de pontos (em 2005, 2008, 2009, 2013 e 2016, por exemplo). Esses pontos adicionais geralmente correspondem a oscilações mais curtas ou movimentos intermediários da série, que podem não representar mudanças na distribuição dos dados, reforçando a tendência do HQ em produzir mais falsos positivos. Já o AIC modificado e o BIC exibiram segmentações mais parcimoniosas, com o AIC modificado sendo novamente o mais restritivo.

Um resultado que chama a atenção é que nenhuma penalidade identificou a queda abrupta de março de 2020, o evento associado ao início da pandemia de COVID-19. Embora o HQ detecte um ponto próximo à recuperação, a queda em si não aparece como uma ruptura no comportamento dos dados para o algoritmo. Esse comportamento indica que, apesar de capturar alguns períodos relevantes, o desempenho do PELT nos dados reais não foi tão satisfatório quanto esperado, especialmente diante da magnitude

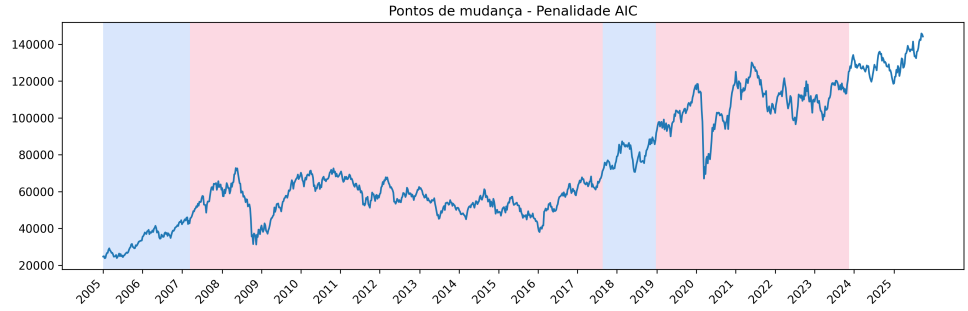
do choque de 2020.

Tabela 4.1: Pontos de mudança detectados na série histórica do Ibovespa segundo diferentes penalidades.

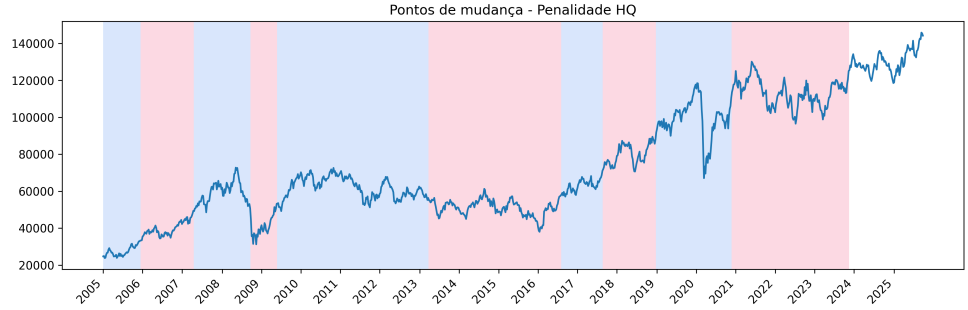
Índice	Data	Penalidade(s)
50	2005-12-12	HQ
115	2007-03-12	AIC Modificado, BIC, HQ
120	2007-04-16	HQ
195	2008-09-22	HQ
230	2009-05-25	HQ
430	2013-03-25	HQ
605	2016-08-01	HQ
660	2017-08-21	AIC Modificado, BIC, HQ
730	2018-12-24	AIC Modificado, BIC, HQ
830	2020-11-23	BIC, HQ
985	2023-11-13	AIC Modificado, BIC, HQ



(a) Pontos de mudança detectados na série do ibovespa utilizando a penalidade BIC.



(b) Pontos de mudança detectados na série do ibovespa utilizando a penalidade AIC Modificada.



(c) Pontos de mudança detectados na série do ibovespa utilizando a penalidade HQ.

Figura 4.2: Pontos de mudança detectados na série histórica do Ibovespa segundo diferentes penalidades.

4.1 Conclusão - Aplicação

Em resumo, a aplicação do PELT na série histórica do Ibovespa entre 2005 e 2025 mostrou que o algoritmo foi capaz de identificar alguns períodos historicamente associados a mudanças relevantes no comportamento do mercado, como os anos de 2007, 2017, 2018 e 2023, que surgiram de forma consistente entre as penalidades. Esses pontos correspondem a momentos de maior tensão econômica ou política e, visualmente, a série de fato apresenta mudanças no comportamento dos dados nesses intervalos. Por outro lado, também observamos que o método apresentou limitações importantes, sobretudo ao não detectar o episódio mais abrupto da série, a forte queda de março de 2020 durante o

início da pandemia de COVID-19, que seria intuitivamente um dos principais candidatos a ponto de mudança.

Além disso, o comportamento das penalidades na aplicação real refletiu o que foi observado nas simulações: o HQ detectou um número maior de pontos, incluindo diversos que parecem detectar oscilações locais sem representar uma mudança muito evidente; o AIC modificado mostrou-se mais equilibrado, identificando pontos plausíveis sem excesso de segmentação. Essa comparação reforça que, embora o algoritmo consiga captar algumas mudanças importantes, sua sensibilidade depende também da penalidade utilizada e da 'intensidade' da quebra presente na série.

Por fim, os resultados indicam que o PELT, com a função de custo e penalidades utilizadas neste estudo, apresenta desempenho razoável, mas não totalmente satisfatório quando aplicado a dados financeiros reais, especialmente em contextos de alta volatilidade e mudanças muito rápidas, como observado em 2020.

Capítulo 5

Conclusão

Este trabalho teve como objetivo estudar o funcionamento do algoritmo Pruned Exact Linear Time (PELT) e aplicá-lo à detecção de pontos de mudança em séries temporais. Para isso, o estudo foi organizado em duas etapas principais: primeiro, realizamos um conjunto de simulações para entender como o algoritmo se comporta em diferentes cenários; depois, aplicamos o método a uma série real, o fechamento semanal do Ibovespa entre 2005 e 2025, para verificar se o PELT conseguiria identificar momentos importantes da história econômica recente.

Os resultados obtidos mostraram que a escolha da penalidade faz diferença no resultado. Tanto na aplicação simulada quanto na aplicação em dados reais, o critério AIC modificado apresentou o desempenho mais equilibrado, acertando pontos importantes sem exagerar na quantidade de detecções, enquanto o HQ acabou detectando muitos pontos, inclusive em situações em que não parecia haver uma mudança real no comportamento da série. Além disso, para os dados reais, o algoritmo identificou alguns períodos históricos relevantes, mas também deixou de marcar um dos momentos mais importantes da série, a queda repentina causada pela pandemia de COVID-19 em 2020. Isso mostra que, apesar de funcionar bem em alguns casos, o método não teve o desempenho esperado em todas as situações.

Como recomendação para futuros estudos, algumas escolhas poderiam ser diferentes. Seria interessante comparar o PELT diretamente com outros métodos de segmentação, como a segmentação binária, para entender melhor as diferenças entre eles e identificar em quais situações cada método funciona melhor. Além disso, seria recomendado que todo o trabalho seja desenvolvido diretamente em Python (Foundation, 2024) desde o início. Durante o processo deste estudo, foi possível perceber que os pacotes disponíveis nessa

linguagem são mais flexíveis e permitem testar diferentes funções de custo e penalidades com muito mais facilidade. Como parte das análises foi inicialmente feita em R (R Core Team, 2023), foi necessário adaptar códigos e refazer etapas em Python, algo que poderia ter sido evitado com essa escolha desde o início.

De forma geral, o trabalho atingiu seu objetivo ao avaliar o PELT em diferentes condições e aplicar o método a uma série real. Os resultados mostraram que o algoritmo tem potencial, especialmente por ser computacionalmente pouco custoso, rápido e fácil de interpretar. Ao mesmo tempo, ficou evidente que seu desempenho depende bastante das escolhas feitas pelo pesquisador e das características da série analisada. Assim, estudos futuros poderiam explorar outras funções de custo, comparar o método com alternativas já consolidadas e testar sua aplicação em diferentes tipos de dados, ampliando a compreensão sobre quando e como o PELT oferece os melhores resultados.

Referências Bibliográficas

Foundation, P. S. (2024). Python: A programming language. https://www.python.org.

Garreau, D. e Arlot, S. (2017). Consistent change-point detection with kernels.

Killick, R., Fearnhead, P. e Eckley, I. A. (2012). Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, **107**(500), 1590–1598.

Kuo, C. (2022). *Modern Times Series Anomaly Detection*. Independently published. ISBN 9798363295751.

Morettin, P. e Toloi, C. (2018). *Análise de séries temporais: modelos lineares univariados*. BLUCHER. ISBN 9788521213529.

R Core Team (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Serinaldi, F. e Kilsby, C. G. (2016). The importance of prewhitening in change point analysis under persistence. *Stochastic Environmental Research and Risk Assessment*, **30**(2), 763–777.

Truong, C., Oudre, L. e Vayatis, N. (2019). Selective review of offline change point detection methods. *Signal Processing*, **167**, 107299.

Wambui, G. D., Waititu, G. A. e Wanjoya, A. (2015). The power of the pruned exact linear time (pelt) test in multiple changepoint detection. *American Journal of Theoretical and Applied Statistics*, **4**(6), 581.

Yao, Y. (1984). Estimation of a noisy discrete-time step function: Bayes and empirical bayes approaches.

Apêndice A

Códigos

```
##############################################################################
############################---- SIMULAÇÕES ----############################

######################---- SIMULAÇÕES CENÁRIO 1 ----#########################

pip install ruptures

#Pacotes
import numpy as np
import pandas as pd
import ruptures as rpt
import matplotlib.pyplot as plt
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# ----- 1. Simulação dos dados -----#

np.random.seed(791280)

# Parâmetros do modelo
ar_param_pos = 0.7   # parâmetro da primeira série
ar_param_neg = -0.3  # parâmetro da segunda série
n_pontos = 100       # número de pontos em cada série
num_simulacoes = 1000  # número de simulações

#Loop
series_simuladas = []

for i in range(num_simulacoes):
    #ARIMA(1,1,0) → AR(1) + diferença
    ar1 = np.array([1, -ar_param_pos])
    ar2 = np.array([1, -ar_param_neg])

    #Criando processos AR(1)
    process1 = ArmaProcess(ar1, [1])
```

```
    process2 = ArmaProcess(ar2, [1])

    #Simulando séries estacionárias
    serie1 = process1.generate_sample(nsample=n_pontos)
    serie2 = process2.generate_sample(nsample=n_pontos)

    #Integrando para obter ARIMA(1,1,0)
    serie1 = np.cumsum(serie1)
    serie2 = np.cumsum(serie2)

    #Juntando as duas séries
    serie_completa = np.concatenate([serie1, serie2])
    series_simuladas.append(serie_completa)

len(series_simuladas)
series_simuladas

import matplotlib.pyplot as plt

#Vendo uma das séries
Para ver outras é só mudar o índice
#python é -1 indexado, lembrar de subtrair o índice
plt.figure(figsize=(10, 4))
plt.plot(series_simuladas[815])
plt.title("Simulação 816")
plt.show()

#Vendo PACF das séries escolhidas de exemplo
indices = [56, 393, 535,816]
for idx in indices:
    serie =  np.diff(series_simuladas[idx - 1])
    plt.figure(figsize=(12,5))
    # PACF
    plt.subplot(1,2,2)
    plot_pacf(serie, ax=plt.gca(), lags=40, method="ywm")
    plt.title(f"PACF - Série simulada {idx}")
    plt.tight_layout()
    plt.show()


######## Detectando pontos com a penalidade BIC
# ----- 2. Loop para detecção dos pontos de mudança -----#
np.random.seed(791280)
pontos_reais = [100]
pontos_mudanca = []
series_com_acerto = []
series_com_acerto_perfeito = []
banda = 5  #tolerância (±5)
series_com_tolerancia = []
```

```
for i, serie in enumerate(series_simuladas, start=1):
    n = len(serie)
    beta = 2 #média e variância
    pen_bic = beta * np.log(n)
    algo = rpt.Pelt(model = 'rbf').fit(serie)
    changepoints = algo.predict(pen= pen_bic)

    #Removendo o último ponto (final da série)
    if changepoints and changepoints[-1] == len(serie):
        changepoints = changepoints[:-1]

    pontos_mudanca.append(changepoints)

    #Verificando acertos
    acertou = any(p in changepoints for p in pontos_reais)
    acertou_perfeito = set(changepoints) == set(pontos_reais)

    if acertou:
        series_com_acerto.append(i)
    if acertou_perfeito:
        series_com_acerto_perfeito.append(i)

    #Criando a banda de teolerância
    #Só entra se:
    #1) não acertou o ponto real
    #2) todos os changepoints detectados estão dentro do intervalo [95, 105]
    if not acertou:
        if all((pontos_reais[0] - banda) <= cp <= (pontos_reais[0] + banda) for cp in changepoints):
            if len(changepoints) > 0:  #precisa ter pelo menos um ponto detectado
                series_com_tolerancia.append(i)

    print(f"Simulação {i}: pontos detectados -> {changepoints}")

# ----- 3. Resumo dos resultados -----
print("\nResumo final:")
print(f"Número total de simulações: {num_simulacoes}")
print(f"Ponto de mudança real: {pontos_reais}")
print(f"Séries com acerto (ponto real encontrado): {len(series_com_acerto)} de {num_simulacoes}")
print(f"Séries com acerto perfeito (apenas ponto real): {len(series_com_acerto_perfeito)} de {num_simulacoes}")
print(f"Índices das séries com acerto perfeito: {series_com_acerto_perfeito}")
print(f"Índices das séries com acerto: {series_com_acerto}")
print(f"Séries com pontos APENAS dentro da banda
["f"{pontos_reais[0]-banda}, {pontos_reais[0]+banda}]: "f"{len(series_com_tolerancia)}")
print(f"Índices dessas séries: {series_com_tolerancia}")

for i in series_com_acerto_perfeito:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps = pontos_mudanca[i-1]
    rpt.display(serie, cps, figsize=(10, 4))
```

```
    plt.title(f"Simulação {i}")
    plt.show()

for i in series_com_tolerancia:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps = pontos_mudanca[i-1]
    rpt.display(serie, cps, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.show()

######## Detectando pontos com a penalidade AIC modificada

# ----- 2. Loop para detecção de pontos de mudança -----#
np.random.seed(791280)
pontos_reais = [100]
pontos_mudanca2 = []
series_com_acerto2 = []
series_com_acerto_perfeito2 = []
banda = 5  #tolerância (±5)
series_com_tolerancia2 = []

for i, serie in enumerate(series_simuladas, start=1):
    n = len(serie)
    beta = 2 #média e variância
    pen_aic = 2*beta*np.log(n)
    algo2 = rpt.Pelt(model="rbf").fit(serie)
    changepoints2 = algo2.predict(pen=pen_aic)

    #Removendo o último ponto (final da série)
    if changepoints2 and changepoints2[-1] == len(serie):
        changepoints2 = changepoints2[:-1]

    pontos_mudanca2.append(changepoints2)

    #Verificando acertos
    acertou2 = any(p in changepoints2 for p in pontos_reais)
    acertou_perfeito2 = set(changepoints2) == set(pontos_reais)

    if acertou2:
        series_com_acerto2.append(i)
    if acertou_perfeito2:
        series_com_acerto_perfeito2.append(i)

    #Criando a banda de teolerância
    #Só entra se:
    #1) não acertou o ponto real
    #2) todos os changepoints detectados estão dentro do intervalo [95, 105]
    if not acertou2:
        if all((pontos_reais[0] - banda) <= cp <= (pontos_reais[0] + banda) for cp in changepoints2):
```

```
            if len(changepoints2) > 0:  #precisa ter pelo menos um ponto detectado
                series_com_tolerancia2.append(i)

    print(f"Simulação {i}: pontos detectados -> {changepoints2}")

# ----- 3. Resumo dos resultados -----
print("\nResumo final:")
print(f"Número total de simulações: {num_simulacoes}")
print(f"Ponto de mudança real: {pontos_reais}")
print(f"Séries com acerto (ponto real encontrado): {len(series_com_acerto2)} de {num_simulacoes}")
print(f"Séries com acerto perfeito (apenas ponto real): {len(series_com_acerto_perfeito2)} de {num_simulacoes}")
print(f"Índices das séries com acerto perfeito: {series_com_acerto_perfeito2}")
print(f"Índices das séries com acerto: {series_com_acerto2}")
print(f"Séries com pontos APENAS dentro da banda
["f"{pontos_reais[0]-banda}, {pontos_reais[0]+banda}]: "f{len(series_com_tolerancia2)}")
print(f"Índices dessas séries: {series_com_tolerancia2}")

for i in series_com_acerto_perfeito2:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps2 = pontos_mudanca2[i-1]
    rpt.display(serie, cps2, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.show()

for i in series_com_tolerancia2:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps2 = pontos_mudanca2[i-1]
    rpt.display(serie, cps2, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.show()


######## Detectando pontos com a penalidade HQ

# ----- 2. Loop para detecção de pontos de mudança -----#
np.random.seed(791280)
pontos_reais = [100]
pontos_mudanca3 = []
series_com_acerto3 = []
series_com_acerto_perfeito3 = []
banda = 5 #tolerancia (±5)
series_com_tolerancia3 = []

for i, serie in enumerate(series_simuladas, start=1):
    n = len(serie)
    beta = 2 #média e variância
    pen_hq = 2*beta*np.log(np.log(n))
    algo3 = rpt.Pelt(model="rbf").fit(serie)
```

```
    changepoints3 = algo3.predict(pen=pen_hq)

    #Removendo o último ponto (final da série)
    if changepoints3 and changepoints3[-1] == len(serie):
        changepoints3 = changepoints3[:-1]

    pontos_mudanca3.append(changepoints3)

    #Verificando acertos
    acertou3 = any(p in changepoints3 for p in pontos_reais)
    acertou_perfeito3 = set(changepoints3) == set(pontos_reais)

    if acertou3:
        series_com_acerto3.append(i)
    if acertou_perfeito3:
        series_com_acerto_perfeito3.append(i)

    #Criando a banda de teolerância ---
    #Só entra se:
    #1) não acertou o ponto real
    #2) todos os changepoints detectados estão dentro do intervalo [95, 105]
    if not acertou3:
        if all((pontos_reais[0] - banda) <= cp <= (pontos_reais[0] + banda) for cp in changepoints3):
            if len(changepoints3) > 0:  #precisa ter pelo menos um ponto detectado
                series_com_tolerancia3.append(i)

    print(f"Simulação {i}: pontos detectados -> {changepoints3}")

# ----- 3. Resumo dos resultados -----#
print("\nResumo final:")
print(f"Número total de simulações: {num_simulacoes}")
print(f"Ponto de mudança real: {pontos_reais}")
print(f"Séries com acerto (ponto real encontrado): {len(series_com_acerto3)} de {num_simulacoes}")
print(f"Séries com acerto perfeito (apenas ponto real): {len(series_com_acerto_perfeito3)} de {num_simulacoes}")
print(f"Índices das séries com acerto perfeito: {series_com_acerto_perfeito3}")
print(f"Índices das séries com acerto: {series_com_acerto3}")
print(f"Séries com pontos APENAS dentro da banda
["f"{pontos_reais[0]-banda}, {pontos_reais[0]+banda}]: "f{len(series_com_tolerancia3)}")
print(f"Índices dessas séries: {series_com_tolerancia3}")


for i in series_com_acerto_perfeito3:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps3 = pontos_mudanca3[i-1]
    rpt.display(serie, cps3, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.show()

for i in series_com_tolerancia3:
```

```
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps3 = pontos_mudanca3[i-1]
    rpt.display(serie, cps3, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.savefig("mudanca_simul56_c1p3_tol.png", dpi=300, bbox_inches='tight')
    plt.show()

############ Tabela com os resultados
# número total de simulações
N = num_simulacoes

resultados = pd.DataFrame({
    "Critério": ["BIC", "AIC", "HQ"],
    "Acertos": [
        len(series_com_acerto),
        len(series_com_acerto2),
        len(series_com_acerto3),
    ],
    "Acertos perfeitos": [
        len(series_com_acerto_perfeito),
        len(series_com_acerto_perfeito2),
        len(series_com_acerto_perfeito3),
    ],
    "Acertos na banda de tolerância ": [
        len(series_com_tolerancia),
        len(series_com_tolerancia2),
        len(series_com_tolerancia3),
    ],
})

#Criando as %
resultados["Taxa acerto"] = resultados["Acertos"] / N
resultados["Taxa acerto perfeito"] = resultados["Acertos perfeitos"] / N
resultados["Taxa acerto considerando a tolerância"] = resultados["Acertos na banda de tolerância "] / N
print(resultados)


######################---- SIMULAÇÕES CENÁRIO 2 ----########################

# ----- 1. Simulação dos dados -----#

np.random.seed(791280)

# Parâmetros do modelo
ar_param_pos = 0.7   # parâmetro da primeira série
ar_param_neg = -0.3  # parâmetro da segunda série
ar_param3 = 0.2
n_pontos = 100       # número de pontos em cada série
num_simulacoes = 1000  # número de simulações
```

```
# Como no ARIMA(1,1,0), d=1 → simulamos AR(1) e depois integramos (acumulamos)
series_simuladas = []

for i in range(num_simulacoes):
    # ARIMA(1,1,0) → AR(1) + diferença
    ar1 = np.array([1, -ar_param_pos])   # sinal invertido:  aparece com sinal negativo
    ar2 = np.array([1, -ar_param_neg])
    ar3 = np.array([1, -ar_param3])

    # Cria processos AR(1)
    process1 = ArmaProcess(ar1, [1])
    process2 = ArmaProcess(ar2, [1])
    process3 = ArmaProcess(ar3, [1])

    # Simula séries estacionárias
    serie1 = process1.generate_sample(nsample=n_pontos)
    serie2 = process2.generate_sample(nsample=n_pontos)
    serie3 = process3.generate_sample(nsample=n_pontos)

    # Integra para obter ARIMA(1,1,0)
    serie1 = np.cumsum(serie1)
    serie2 = np.cumsum(serie2)
    serie3 = np.cumsum(serie3)

    # Junta as duas séries
    serie_completa = np.concatenate([serie1, serie2, serie3])
    series_simuladas.append(serie_completa)

len(series_simuladas)
len(series_simuladas[0])
series_simuladas

#vendo uma das séries simuladas
plt.figure(figsize=(10, 4))
plt.plot(series_simuladas[834])
plt.title("Simulação 835")
plt.show()

#Vendo os pacfs
idx = 835
series_diff =  np.diff(series_simuladas[idx - 1])
plt.figure(figsize=(10, 4))
# PACF
plot_pacf(series_diff, ax=plt.gca(), lags=40, method="ywm")
plt.title(f"PACF - Série simulada {idx}")
plt.tight_layout()
plt.show()

######## Detectando pontos com a penalidade BIC
```

```
np.random.seed(791280)

# ----- 2. Loop para detecção de pontos de mudança -----
pontos_reais = [100, 200]
pontos_mudanca = []
series_com_acerto = []
series_com_acerto_perfeito = []
banda = 5  # tolerância ±5
series_com_tolerancia = []

for i, serie in enumerate(series_simuladas, start=1):
    n = len(serie)
    beta = 2  #média e variância
    pen_bic = beta * np.log(n)

    algo = rpt.Pelt(model="rbf").fit(serie)
    changepoints = algo.predict(pen=pen_bic)

    #Remove o último ponto (final da série)
    if changepoints and changepoints[-1] == len(serie):
        changepoints = changepoints[:-1]

    pontos_mudanca.append(changepoints)

    #Verificando acertos
    acertou = all(p in changepoints for p in pontos_reais)
    acertou_perfeito = set(changepoints) == set(pontos_reais)

    if acertou:
        series_com_acerto.append(i)
    if acertou_perfeito:
        series_com_acerto_perfeito.append(i)

    #Verificando tolerância
    if len(changepoints) > 0:
        #pontos próximos de cada ponto real
        pontos_banda = {
            p_real: [cp for cp in changepoints if abs(cp - p_real) <= banda]
            for p_real in pontos_reais
        }
        #Vendo se tem pelo menos um cp em cada banda
        cp_cada_banda = all(len(pontos_banda[p_real]) > 0 for p_real in pontos_reais)

        #Nenhum ponto pode estar fora de TODAS as bandas
        cps_alguma_banda = set(sum(pontos_banda.values(), []))
        cp_dentro = set(changepoints).issubset(cps_alguma_banda)

        if cp_cada_banda and cp_dentro:
            series_com_tolerancia.append(i)
```

```
    print(f"Simulação {i}: pontos detectados -> {changepoints}")

# ----- 3. Resumo dos resultados -----#
print("\nResumo final:")
print(f"Número total de simulações: {num_simulacoes}")
print(f"Pontos de mudança reais: {pontos_reais}")
print(f"Séries com acerto (todos os pontos reais encontrados): {len(series_com_acerto)} de {num_simulacoes}")
print(f"Séries com acerto perfeito (apenas pontos reais): {len(series_com_acerto_perfeito)} de {num_simulacoes}")
print(f"Índices das séries com acerto perfeito: {series_com_acerto_perfeito}")
print(f"Índices das séries com acerto: {series_com_acerto}")
print(f"\nSéries com pontos APENAS dentro das bandas
"f"[{pontos_reais[0]-banda}, {pontos_reais[0]+banda}] e
"f"[{pontos_reais[1]-banda}, {pontos_reais[1]+banda}]: "f{len(series_com_tolerancia)}")
print(f"Índices dessas séries: {series_com_tolerancia}")

for i in series_com_acerto_perfeito:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps = pontos_mudanca[i-1]
    rpt.display(serie, cps, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.savefig(f"mudanca_simul{i}_c2p1.png", dpi=300, bbox_inches='tight')
    plt.show()

for i in series_com_tolerancia:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps = pontos_mudanca[i-1]
    rpt.display(serie, cps, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.show()

######## Detectando pontos com a penalidade AIC modificada

np.random.seed(791280)

# ----- 2. Loop para detecção de pontos de mudança -----
pontos_reais = [100, 200]
banda = 5
pontos_mudanca2 = []
series_com_acerto2 = []
series_com_acerto_perfeito2 = []
series_com_tolerancia2 = []

for i, serie in enumerate(series_simuladas, start=1):
    n = len(serie)
    beta = 2  # média e variância
    pen_aic = 2 * beta * np.log(n)
```

```
    algo2 = rpt.Pelt(model="rbf").fit(serie)
    changepoints2 = algo2.predict(pen=pen_aic)

    # Remove o último ponto (final da série)
    if changepoints2 and changepoints2[-1] == len(serie):
        changepoints2 = changepoints2[:-1]

    pontos_mudanca2.append(changepoints2)

    #Verificando acertos
    acertou2 = all(p in changepoints2 for p in pontos_reais)
    acertou_perfeito2 = set(changepoints2) == set(pontos_reais)

    if acertou2:
        series_com_acerto2.append(i)
    if acertou_perfeito2:
        series_com_acerto_perfeito2.append(i)

    #Verificando tolerância
    if len(changepoints2) > 0:
        #pontos próximos de cada ponto real
        pontos_banda2 = {
            p_real: [cp for cp in changepoints2 if abs(cp - p_real) <= banda]
            for p_real in pontos_reais
        }
        #Vendo se tem pelo menos um cp em cada banda
        cp_cada_banda2 = all(len(pontos_banda2[p_real]) > 0 for p_real in pontos_reais)

        #Nenhum ponto pode estar fora de TODAS as bandas
        cps_alguma_banda2 = set(sum(pontos_banda2.values(), []))
        cp_dentro2 = set(changepoints2).issubset(cps_alguma_banda2)

        if cp_cada_banda2 and cp_dentro2:
            series_com_tolerancia2.append(i)

    print(f"Simulação {i}: pontos detectados -> {changepoints2}")

# ----- 3. Resumo dos resultados -----
print("\nResumo final:")
print(f"Número total de simulações: {num_simulacoes}")
print(f"Pontos de mudança reais: {pontos_reais}")
print(f"Séries com acerto (todos os pontos reais encontrados): {len(series_com_acerto2)} de {num_simulacoes}")
print(f"Séries com acerto perfeito (apenas pontos reais): {len(series_com_acerto_perfeito2)} de {num_simulacoes}")
print(f"Índices das séries com acerto perfeito: {series_com_acerto_perfeito2}")
print(f"Índices das séries com acerto: {series_com_acerto2}")
print(f"\nSéries com pontos APENAS dentro das bandas
"f"[{pontos_reais[0]-banda}, {pontos_reais[0]+banda}] e
"f"[{pontos_reais[1]-banda}, {pontos_reais[1]+banda}]: "f{len(series_com_tolerancia2)}")
print(f"Índices dessas séries: {series_com_tolerancia2}")
```

```
for i in series_com_acerto_perfeito2:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps2 = pontos_mudanca2[i-1]
    rpt.display(serie, cps2, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.show()

for i in series_com_tolerancia2:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps2 = pontos_mudanca2[i-1]
    rpt.display(serie, cps2, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.show()

######## Detectando pontos com a penalidade HQ

np.random.seed(791280)

# ----- 2. Loop para detecção de pontos de mudança -----
pontos_reais = [100, 200]
pontos_mudanca3 = []
series_com_acerto3 = []
series_com_acerto_perfeito3 = []
banda = 5
series_com_tolerancia3 = []

for i, serie in enumerate(series_simuladas, start=1):
    n = len(serie)
    beta = 2  # média e variância
    pen_hq = 2*beta * np.log(np.log(n))
    algo3 = rpt.Pelt(model="rbf").fit(serie)
    changepoints3 = algo3.predict(pen=pen_hq)

    # Remove o último ponto (final da série)
    if changepoints3 and changepoints3[-1] == len(serie):
        changepoints3 = changepoints3[:-1]

    pontos_mudanca3.append(changepoints3)

    #Verificando acertos
    acertou3 = all(p in changepoints3 for p in pontos_reais)
    acertou_perfeito3 = set(changepoints3) == set(pontos_reais)

    if acertou3:
        series_com_acerto3.append(i)
    if acertou_perfeito3:
        series_com_acerto_perfeito3.append(i)
```

```
    #Verificando tolerância
    if len(changepoints3) > 0:
        #pontos próximos de cada ponto real
        pontos_banda3 = {
            p_real: [cp for cp in changepoints3 if abs(cp - p_real) <= banda]
            for p_real in pontos_reais
        }
        #Vendo se tem pelo menos um cp em cada banda
        cp_cada_banda3 = all(len(pontos_banda3[p_real]) > 0 for p_real in pontos_reais)

        #Nenhum ponto pode estar fora de TODAS as bandas
        cps_alguma_banda3 = set(sum(pontos_banda3.values(), []))
        cp_dentro3 = set(changepoints3).issubset(cps_alguma_banda3)

        if cp_cada_banda3 and cp_dentro3:
            series_com_tolerancia3.append(i)

    print(f"Simulação {i}: pontos detectados -> {changepoints3}")

# ----- 3. Resumo dos resultados -----
print("\nResumo final:")
print(f"Número total de simulações: {num_simulacoes}")
print(f"Pontos de mudança reais: {pontos_reais}")
print(f"Séries com acerto (todos os pontos reais encontrados): {len(series_com_acerto3)} de {num_simulacoes}")
print(f"Séries com acerto perfeito (apenas pontos reais): {len(series_com_acerto_perfeito3)} de {num_simulacoes}")
print(f"Índices das séries com acerto perfeito: {series_com_acerto_perfeito3}")
print(f"Índices das séries com acerto: {series_com_acerto3}")
print(f"\nSéries com pontos APENAS dentro das bandas
"f"[{pontos_reais[0]-banda}, {pontos_reais[0]+banda}] e
"f"[{pontos_reais[1]-banda}, {pontos_reais[1]+banda}]:
"f"{len(series_com_tolerancia3)}")
print(f"Índices dessas séries: {series_com_tolerancia3}")

for i in series_com_acerto3[::10]:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps3 = pontos_mudanca3[i-1]
    rpt.display(serie, cps3, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.show()

#### Criando tabela com os resultados
N = num_simulacoes

resultados = pd.DataFrame({
    "Critério": ["BIC", "AIC", "HQ"],
    "Acertos": [
        len(series_com_acerto),
        len(series_com_acerto2),
        len(series_com_acerto3),
```

```
    ],
    "Acertos perfeitos": [
        len(series_com_acerto_perfeito),
        len(series_com_acerto_perfeito2),
        len(series_com_acerto_perfeito3),
    ],
    "Acertos na banda de tolerância ": [
        len(series_com_tolerancia),
        len(series_com_tolerancia2),
        len(series_com_tolerancia3),
    ],
})

#Criando as %
resultados["Taxa acerto"] = resultados["Acertos"] / N
resultados["Taxa acerto perfeito"] = resultados["Acertos perfeitos"] / N
resultados["Taxa acerto considerando a tolerância"] = resultados["Acertos na banda de tolerância "] / N
print(resultados)

######################---- SIMULAÇÕES CENÁRIO 3 ----########################

# ----- 1. Simulação dos dados -----#
np.random.seed(791280)
# Parâmetros do modelo
ar_param_pos = 0.7   # parâmetro da primeira série
ar_param_neg = -0.3  # parâmetro da segunda série
num_simulacoes = 1000  # número de simulações

# Como no ARIMA(1,1,0), d=1 → simulamos AR(1) e depois integramos (acumulamos)
series_simuladas = []

for i in range(num_simulacoes):
    # ARIMA(1,1,0) → AR(1) + diferença
    ar1 = np.array([1, -ar_param_pos])   # sinal invertido:  aparece com sinal negativo
    ar2 = np.array([1, -ar_param_neg])

    # Cria processos AR(1)
    process1 = ArmaProcess(ar1, [1])
    process2 = ArmaProcess(ar2, [1])

    # Simula séries estacionárias
    serie1 = process1.generate_sample(nsample=150)
    serie2 = process2.generate_sample(nsample=50)

    # Integra para obter ARIMA(1,1,0)
    serie1 = np.cumsum(serie1)
    serie2 = np.cumsum(serie2)

    # Junta as duas séries
    serie_completa = np.concatenate([serie1, serie2])
```

```
    series_simuladas.append(serie_completa)

len(series_simuladas)
len(series_simuladas[0])
series_simuladas

#Vendo uma das séries
plt.figure(figsize=(10, 4))
plt.plot(series_simuladas[472])
plt.title("Simulação 473")
plt.show()

#Vendo PACFs
idx = 961
series_diff =  np.diff(series_simuladas[idx - 1])
plt.figure(figsize=(10, 4))
# PACF
plot_pacf(series_diff, ax=plt.gca(), lags=40, method="ywm")
plt.title(f"PACF - Série simulada {idx}")
plt.tight_layout()
plt.show()

######## Detectando pontos com a penalidade BIC

# ----- 2. Loop para detecção de pontos de mudança -----
np.random.seed(791280)

pontos_reais = [150]
pontos_mudanca = []
series_com_acerto = []
series_com_acerto_perfeito = []
banda = 5  #tolerância (±5)
series_com_tolerancia = []

for i, serie in enumerate(series_simuladas, start=1):
    n = len(serie)
    beta = 2  # média e variância
    pen_bic = beta * np.log(n)
    algo = rpt.Pelt(model="rbf").fit(serie)
    changepoints = algo.predict(pen=pen_bic)

    # Remove o último ponto (final da série)
    if changepoints and changepoints[-1] == len(serie):
        changepoints = changepoints[:-1]

    pontos_mudanca.append(changepoints)

    # Verifica acertos
    acertou = any(p in changepoints for p in pontos_reais)
    acertou_perfeito = set(changepoints) == set(pontos_reais)
```

```
    if acertou:
        series_com_acerto.append(i)
    if acertou_perfeito:
        series_com_acerto_perfeito.append(i)

    #Criando a banda de teolerância
    #Só entra se:
    #1) não acertou o ponto real
    #2) todos os changepoints detectados estão dentro do intervalo [95, 105]
    if not acertou:
        if all((pontos_reais[0] - banda) <= cp <= (pontos_reais[0] + banda) for cp in changepoints):
            if len(changepoints) > 0:  # precisa ter pelo menos um ponto detectado
                series_com_tolerancia.append(i)


    print(f"Simulação {i}: pontos detectados -> {changepoints}")

# ----- 3. Resumo dos resultados -----
print("\nResumo final:")
print(f"Número total de simulações: {num_simulacoes}")
print(f"Ponto de mudança real: {pontos_reais}")
print(f"Séries com acerto (ponto real encontrado): {len(series_com_acerto)} de {num_simulacoes}")
print(f"Séries com acerto perfeito (apenas ponto real): {len(series_com_acerto_perfeito)} de {num_simulacoes}")
print(f"Índices das séries com acerto perfeito: {series_com_acerto_perfeito}")
print(f"Índices das séries com acerto: {series_com_acerto}")
print(f"Séries com pontos APENAS dentro da banda
["f"{pontos_reais[0]-banda}, {pontos_reais[0]+banda}]: "f{len(series_com_tolerancia)}")
print(f"Índices dessas séries: {series_com_tolerancia}")

for i in series_com_acerto_perfeito:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps = pontos_mudanca[i-1]
    rpt.display(serie, cps, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.show()

for i in series_com_tolerancia:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps = pontos_mudanca[i-1]
    rpt.display(serie, cps, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.show()

######## Detectando pontos com a penalidade AIC modificada

# ----- 2. Loop para detecção de pontos de mudança -----#
np.random.seed(791280)
```

```
pontos_reais = [150]
pontos_mudanca2 = []
series_com_acerto2 = []
series_com_acerto_perfeito2 = []
banda = 5  #tolerância (±5)
series_com_tolerancia2 = []

for i, serie in enumerate(series_simuladas, start=1):
    n = len(serie)
    beta = 2  # média e variância
    pen_aic = 2*beta * np.log(n)
    algo2 = rpt.Pelt(model="rbf").fit(serie)
    changepoints2 = algo2.predict(pen=pen_aic)

    # Remove o último ponto (final da série)
    if changepoints2 and changepoints2[-1] == len(serie):
        changepoints2 = changepoints2[:-1]

    pontos_mudanca2.append(changepoints2)

    # Verifica acertos
    acertou2 = any(p in changepoints2 for p in pontos_reais)
    acertou_perfeito2 = set(changepoints2) == set(pontos_reais)

    if acertou2:
        series_com_acerto2.append(i)
    if acertou_perfeito2:
        series_com_acerto_perfeito2.append(i)

    #Criando a banda de teolerância
    #Só entra se:
    #1) não acertou o ponto real
    #2) todos os changepoints detectados estão dentro do intervalo [95, 105]
    if not acertou2:
        if all((pontos_reais[0] - banda) <= cp <= (pontos_reais[0] + banda) for cp in changepoints2):
            if len(changepoints2) > 0:  # precisa ter pelo menos um ponto detectado
                series_com_tolerancia2.append(i)

    print(f"Simulação {i}: pontos detectados -> {changepoints2}")

# ----- 3. Resumo dos resultados -----
print("\nResumo final:")
print(f"Número total de simulações: {num_simulacoes}")
print(f"Ponto de mudança real: {pontos_reais}")
print(f"Séries com acerto (ponto real encontrado): {len(series_com_acerto2)} de {num_simulacoes}")
print(f"Séries com acerto perfeito (apenas ponto real): {len(series_com_acerto_perfeito2)} de {num_simulacoes}")
print(f"Índices das séries com acerto perfeito: {series_com_acerto_perfeito2}")
print(f"Índices das séries com acerto: {series_com_acerto2}")
print(f"Séries com pontos APENAS dentro da banda
```

```
["f"{pontos_reais[0]-banda}, {pontos_reais[0]+banda}]:
"f"{len(series_com_tolerancia2)}")
print(f"Índices dessas séries: {series_com_tolerancia2}")

for i in series_com_acerto_perfeito2:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps2 = pontos_mudanca2[i-1]
    rpt.display(serie, cps2, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.show()

for i in series_com_tolerancia2:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps2 = pontos_mudanca2[i-1]
    rpt.display(serie, cps2, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.show()

######## Detectando pontos com a penalidade HQ

# ----- 2. Loop para detecção de pontos de mudança -----#
np.random.seed(791280)

pontos_reais = [150]
pontos_mudanca3 = []
series_com_acerto3 = []
series_com_acerto_perfeito3 = []
banda = 5 #tolerancia (±5)
series_com_tolerancia3 = []

for i, serie in enumerate(series_simuladas, start=1):
    n = len(serie)
    beta = 2  # média e variância
    pen_hq = 2*beta * np.log(np.log(n))
    algo3 = rpt.Pelt(model="rbf").fit(serie)
    changepoints3 = algo3.predict(pen=pen_hq)

    # Remove o último ponto (final da série)
    if changepoints3 and changepoints3[-1] == len(serie):
        changepoints3 = changepoints3[:-1]

    pontos_mudanca3.append(changepoints3)

    # Verifica acertos
    acertou3 = any(p in changepoints3 for p in pontos_reais)
    acertou_perfeito3 = set(changepoints3) == set(pontos_reais)

    if acertou3:
```

```
        series_com_acerto3.append(i)
    if acertou_perfeito3:
        series_com_acerto_perfeito3.append(i)

    #Criando a banda de teolerância
    #Só entra se:
    #1) não acertou o ponto real
    #2) todos os changepoints detectados estão dentro do intervalo [95, 105]
    if not acertou3:
        if all((pontos_reais[0] - banda) <= cp <= (pontos_reais[0] + banda) for cp in changepoints3):
            if len(changepoints3) > 0:  #precisa ter pelo menos um ponto detectado
                series_com_tolerancia3.append(i)

    print(f"Simulação {i}: pontos detectados -> {changepoints3}")

# ----- 3. Resumo dos resultados -----#
print("\nResumo final:")
print(f"Número total de simulações: {num_simulacoes}")
print(f"Ponto de mudança real: {pontos_reais}")
print(f"Séries com acerto (ponto real encontrado): {len(series_com_acerto3)} de {num_simulacoes}")
print(f"Séries com acerto perfeito (apenas ponto real): {len(series_com_acerto_perfeito3)} de {num_simulacoes}")
print(f"Índices das séries com acerto perfeito: {series_com_acerto_perfeito3}")
print(f"Índices das séries com acerto: {series_com_acerto3}")
print(f"Séries com pontos APENAS dentro da banda
["f"{pontos_reais[0]-banda}, {pontos_reais[0]+banda}]:
"f"{len(series_com_tolerancia3)}")
print(f"Índices dessas séries: {series_com_tolerancia3}")

#não tivemos acertos perfeitos, vamos ver só as com acerto
for i in series_com_acerto3[:30]:
    # O índice da série e dos pontos de mudança é i-1, pois a lista é 0-indexada
    serie = series_simuladas[i-1]
    cps3 = pontos_mudanca3[i-1]
    rpt.display(serie, cps3, figsize=(10, 4))
    plt.title(f"Simulação {i}")
    plt.show()

######## Criando tabela com os resultados

# número total de simulações (mesmo usado nos loops)
N = num_simulacoes

resultados = pd.DataFrame({
    "Critério": ["BIC", "AIC", "HQ"],
    "Acertos": [
        len(series_com_acerto),
        len(series_com_acerto2),
        len(series_com_acerto3),
    ],
    "Acertos perfeitos": [
```

```
        len(series_com_acerto_perfeito),
        len(series_com_acerto_perfeito2),
        len(series_com_acerto_perfeito3),
    ],
    "Acertos na banda de tolerância ": [
        len(series_com_tolerancia),
        len(series_com_tolerancia2),
        len(series_com_tolerancia3),
    ],
})

#Criando as %
resultados["Taxa acerto"] = resultados["Acertos"] / N
resultados["Taxa acerto perfeito"] = resultados["Acertos perfeitos"] / N
resultados["Taxa acerto considerando a tolerância"] = resultados["Acertos na banda de tolerância "] / N
print(resultados)

##############################################################################
##############################################################################
######################---- APLICAÇÃO IBOVESPA ----#########################

!pip install yfinance
!pip install ruptures

#Pacotes
import pandas as pd
import yfinance as yf
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt

#Definando o ticker
ticker = '^BVSP'

#Baixando os dados históricos para o ticker
data = yf.download(ticker, start='2005-01-07', end='2025-10-03', interval='1wk')

data.head(50)
data.shape

#Criando e organizando o data frame
df = pd.DataFrame(data)
df.columns = df.columns.droplevel(1)
df.columns.name = None

df.head(10)

#Plotando o gráfico da série
plt.figure(figsize=(10, 4))
plt.plot(df['Close'])
```

```
plt.scatter(df.index, df['Close'], color="red", s=2, alpha=0.2)
plt.title(f"Valor de fechamento semanal do Índice Ibovespa de Janeiro de 2005 a Outubro de 2025")
plt.xlabel("Ano")
plt.ylabel("Valor de fechamento semanal")
plt.savefig("serie_ibovespa.png", dpi=300, bbox_inches='tight')
plt.show()

######## Detectando pontos com a penalidade BIC

np.random.seed(791280)

pontos_mudanca1 = []
ts_ibov = df["Close"].to_numpy().reshape(-1, 1)
n = len(ts_ibov)
beta = 2  #média e variância
pen_bic = beta * np.log(n)
algo1 = rpt.Pelt(model = 'rbf').fit(ts_ibov)
changepoints1 = algo1.predict(pen= pen_bic)
if changepoints1 and changepoints1[-1] == len(ts_ibov):
        changepoints1 = changepoints1[:-1]
pontos_mudanca1.append(changepoints1)

pontos_mudanca1

#Transformando os índices para datas
cpt1 = [cp for cp in changepoints1 if cp < len(ts_ibov)]
datas_cp1 = df.index[[cp - 1 for cp in cpt1]]
print("Índices dos changepoints:", cpt1)
print("Datas dos changepoints: ")
for i, d in zip(cpt1, datas_cp1):
    print(f"  índice {i} -> data {d.date()}")

#Plotando os pontos encontrados com as datas
datas = df.index
cp_idx = [cp - 1 for cp in cps1 if cp <= len(datas)]

fig, ax_list = rpt.display(serie, cps1, figsize=(12, 4))
ax = ax_list[0]   # eixo principal
ax.set_title("Pontos de mudança - Penalidade BIC")

anos_unicos = sorted(set(datas.year))
ticks_anos = []
labels_anos = []
for ano in anos_unicos:
    datas_do_ano = datas[datas.year == ano]
    if len(datas_do_ano) > 0:
        primeira_data = datas_do_ano[0]
        ticks_anos.append(datas.get_loc(primeira_data))
        labels_anos.append(str(ano))
```

```
ax.set_xticks(ticks_anos)
ax.set_xticklabels(labels_anos, rotation=45, ha="right")

plt.tight_layout()
plt.show()

######## Detectando pontos com a penalidade AIC modificada

np.random.seed(791280)

pontos_mudanca2 = []
ts_ibov = df["Close"].to_numpy().reshape(-1, 1)
n = len(ts_ibov)
beta = 2 #média e variância
pen_aic = 2*beta *np.log(n)
algo2 = rpt.Pelt(model = 'rbf').fit(ts_ibov)
changepoints2 = algo2.predict(pen= pen_aic)
if changepoints2 and changepoints2[-1] == len(ts_ibov):
        changepoints2 = changepoints2[:-1]
pontos_mudanca2.append(changepoints2)

pontos_mudanca2

#Transformando índices para datas
cpt2 = [cp for cp in changepoints2 if cp < len(ts_ibov)]
datas_cp2 = df.index[[cp - 1 for cp in cpt2]]
print("Índices dos changepoints:", cpt2)
print("Datas dos changepoints: ")
for i, d in zip(cpt2, datas_cp2):
    print(f"  índice {i} -> data {d.date()}")

#Plotando os pontos encontrados com as datas
datas = df.index
cp_idx = [cp - 1 for cp in cps2 if cp <= len(datas)]

fig, ax_list = rpt.display(serie, cps2, figsize=(12, 4))
ax = ax_list[0]   # eixo principal
ax.set_title("Pontos de mudança - Penalidade AIC")

anos_unicos = sorted(set(datas.year))
ticks_anos = []
labels_anos = []

for ano in anos_unicos:
    datas_do_ano = datas[datas.year == ano]
    if len(datas_do_ano) > 0:
        primeira_data = datas_do_ano[0]
        ticks_anos.append(datas.get_loc(primeira_data))
        labels_anos.append(str(ano))
```

```
ax.set_xticks(ticks_anos)
ax.set_xticklabels(labels_anos, rotation=45, ha="right")

plt.tight_layout()
plt.show()

######## Detectando pontos com a penalidade HQ

np.random.seed(791280)

pontos_mudanca3 = []
ts_ibov = df["Close"].to_numpy().reshape(-1, 1)
n = len(ts_ibov)
beta = 2  #média e variância
pen_hq = 2*beta*np.log(np.log(n))
algo3 = rpt.Pelt(model = 'rbf').fit(ts_ibov)
changepoints3 = algo3.predict(pen= pen_hq)
if changepoints3 and changepoints3[-1] == len(ts_ibov):
        changepoints3 = changepoints3[:-1]
pontos_mudanca3.append(changepoints3)

pontos_mudanca3

#Transformando índices para datas
cpt3 = [cp for cp in changepoints3 if cp < len(ts_ibov)]
datas_cp3 = df.index[[cp - 1 for cp in cpt3]]
print("Índices dos changepoints:", cpt3)
print("Datas dos changepoints: ")
for i, d in zip(cpt3, datas_cp3):
    print(f"  índice {i} -> data {d.date()}")



#Plotando os pontos encontrados com as datas
datas = df.index  # índice datetime
cp_idx = [cp - 1 for cp in cps3 if cp <= len(datas)]

fig, ax_list = rpt.display(serie, cps3, figsize=(12, 4))
ax = ax_list[0]   # eixo principal
ax.set_title("Pontos de mudança - Penalidade HQ")

anos_unicos = sorted(set(datas.year))
ticks_anos = []
labels_anos = []

for ano in anos_unicos:
    datas_do_ano = datas[datas.year == ano]
    if len(datas_do_ano) > 0:
        primeira_data = datas_do_ano[0]
        ticks_anos.append(datas.get_loc(primeira_data))
        labels_anos.append(str(ano))
```

```
ax.set_xticks(ticks_anos)
ax.set_xticklabels(labels_anos, rotation=45, ha="right")
plt.tight_layout()
plt.show()
```