ÁTILA DA VEIGA

APLICAÇÃO DE ALGORITMOS HEURÍSTICOS NA OTIMIZAÇÃO DOS PARÂMETROS DE *TRADING* COM PARES COINTEGRADOS

São Paulo
2022

ÁTILA DA VEIGA

APLICAÇÃO DE ALGORITMOS HEURÍSTICOS NA OTIMIZAÇÃO DOS PARÂMETROS DE *TRADING* COM PARES COINTEGRADOS

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Financeiro – MBA.

São Paulo
2022

ÁTILA DA VEIGA

APLICAÇÃO DE ALGORITMOS HEURÍSTICOS NA OTIMIZAÇÃO DOS PARÂMETROS DE *TRADING* COM PARES COINTEGRADOS

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Financeiro – MBA.

Área de Concentração:

Engenharia Financeira

Orientador:

Bruno Augusto Angélico

São Paulo
2022

AGRADECIMENTOS

Aos amigos da Sole Capital, por toda sua valiosa ajuda e parceria; desde a influência no tema deste trabalho, passando pelas lições sobre prática de mercado e sobre programação e até na cessão da base de dados, sem a qual a Parte 3 teria sido impossível. Espero que este trabalho possa lhes ser útil de alguma forma.

À minha mãe Gilda Mª S. do Carmo e a meu pai Lintney N. da Veiga por todo seu esforço e sacrifício dedicados à minha formação e àquilo que não se traduz em palavras.

E a minha companheira Julia, pela incansável dose de motivação diária que me faz continuar sempre que possível e recomeçar quando necessário, mas sempre juntos.

RESUMO

Como consequência da expansão do *trading algorítmico* nos mercados financeiros globais; maximizar a eficiência dessas estratégias se tornou uma tarefa indispensável e desafiadora. Em paralelo, com a grande evolução do campo de *machine learning*, novos e sofisticados algoritmos estão à disposição para as mais variadas aplicações. Inserido neste contexto, este trabalho, busca analisar se a aplicação desses algoritmos de otimização heurística na definição dos parâmetros de negociação da popular estratégia de *pairs trading* é capaz de consistentemente melhorar o seu retorno, versus o uso de parâmetros arbitrários e genéricos. Para atingir tal objetivo, fazemos a análise de cointegração com vários pares de ativos e selecionamos alguns dos melhores. Também construímos 4 variações de algoritmos de otimização baseados no PSO e no Algoritmo Genético. Por fim, fazemos a otimização dos parâmetros de montagem, reversão e *stop-loss* desses pares, usando dados históricos por meio de um *backtest*. Os resultados dos experimentos apresentam evidências inconclusivas; sugerindo que este problema é sujeito a um grande risco de sobreajuste na otimização, exigindo maiores cuidados e futuros estudos.

Palavras-Chave – cointegração, *pairs trading*, particle swarm optimization (PSO), otimização heurística.

ABSTRACT

As consequence of algorithmic trading expansion throughout global financial markets; maximizing the efficiency of those strategies has become an indispensable and challenging endeavor. Concurrently, great advances in the field of machine learning are making available newer and more sophisticated algorithms that can be applied to a diverse range of problems. In this context, this work tries to analyze if the application of heuristic optimization algorithms to the definition of the trading parameter of the popular pairs trading strategy is capable of consistently improve its profitability against arbitrary and generic parameters. To achieve that, we analyzed for cointegration several pairs of equities and selected the best. We also built 4 versions of optimization algorithms based on PSO and Genetic Algorithm. Finally, we optimized the inception, reversion, and stop-loss parameters of those pairs, using historical price data with a backtest. The experimental results were mixed, suggesting that this problem is prone to overfitting issues as result of the optimization, which require increased caution and future investigation.

Keywords – cointegration, heuristic optimization, pairs trading, particle swarm optimization (PSO).

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

ACF	Função de autocorrelação
ADF	Teste de Dickey–Fuller aumentado
ARIMA	Modelo autorregressivo integrado de médias móveis
BOVA11	Ishares Ibovespa Fundo de Indice (ETF)
BOVB11	Bradesco Ibovespa Fundo de Indice (ETF)
CMIG4	Companhia de Energia de Minas Gerais PN
ENGI11	Energisa Unit
ENGI4	Energisa PN
ESE/ELS	Variação do algoritimo PSO, desonvolvida por Zhan, Zhi-Hui et al.
ETF	Fundos de investimento negociados em bolsa
FIND11	Fundo Itaú replicação do IFNC (ETF)
g_{best}	Partícula com melhor resultado global do PSO
GA	Algoritmo Genético
GGBR3	Gerdau ON
GOAU3	Metalúrgica Gerdau ON
GOAU4	Metalúrgica Gerdau PN
MWRR	Money Weighted Rate of Return
ON/PN	Papéis ordinários e preferenciais da mesma empresa
PACF	Função de autocorrelação parcial
PETR3	Petrobrás ON
PETR4	Petrobrás PN
PIBB11	Fundo Itaú replicação do Índice Brasil 50 (ETF)
PSO	Particle swarm optimization
SANB11	Banco Santander Brasil Unit
SANB4	Banco Santander Brasil PN
SAPR11	Companhia de Saneamento do Paraná (SANEPAR) Unit
SAPR4	Companhia de Saneamento do Paraná (SANEPAR) PN
SFFE	Quantidade de chamadas da função objetivo até $1^{\underline{o}}$ sucesso
SMAC11	Fundo Itaú de Small Caps (ETF)
SMAL11	Fundo iShares de Small Caps (ETF)
TA/kSFFE	Taxa de acerto dividido por SFFE
TAEE11	Transmissora Aliança de Energia Elétrica Unit
TAEE3	Transmissora Aliança de Energia Elétrica ON
TAEE4	Transmissora Aliança de Energia Elétrica PN

SUMÁRIO

1 INTRODUÇÃO

1.1 Objetivo

Ainda que o campo das "finanças quantitativas" tenha se desenvolvido desde a primeira metade do século XX – com obras seminais como a teoria moderna de portfólio de Harry Markowitz, o cálculo estocástico de Itô ou a precificação de derivativos com Black&Scholes entre tantos outros – foi somente no final desse mesmo século que o mundo viu a aplicação direta e sistemática das finanças quantitativas no mundo do *trading*.

A chamada "Revolução Quant"[1] – que teve entre seus protagonistas figuras como Jim Simons e Robert Mercer, um matemático e o outro cientista da computação – foi a profunda transformação dos mercados resultante da união das finanças quantitativas com a execução sistemática no mercado por meio de sistemas computacionais.

Estima-se[2] que abordagem de *trading algorítmico* responda por algo em torno de 85% do volume financeiro negociados nos mercados globais. Dentre os agentes que operam desta forma há toda sorte de investidores institucionais como bancos e corretoras globais, fundos de investimento e até clubes de investimento e pessoas físicas. Esses agentes dispõe dos mais diferentes níveis de tecnologia, recursos e patrimônio sob gestão. Uma das consequências diretas dessa grande presença do *trading algorítmico* é a dificuldade de se encontrar estratégias rentáveis e eficientes ao longo do tempo. Ou seja, é preciso ser cada vez mais eficiente, ágil e preciso no desenvolvimento e parametrização dos algoritmos.

Inserido nesse contexto, esse trabalho tem como objetivo investigar se a aplicação de otimizadores heurísticos para definição dos parâmetros de montagem, reversão e *stop-loss* da estratégia de *pairs trading* são capazes de torná-la mais eficiente e lucrativa. Como se trata apenas de um trabalho de conclusão de curso, nossos objetivos são modestos. Iremos fazer uma revisão teórica seguida de alguns experimentos com objetivo puramente

[1] Do livro de autoria do jornalista Gregory Zuckerman a sobre a vida de Jim Simons, entitulado "The Man Who Solved the Market: How Jim Simons Launched the Quant Revolution".

[2] [3]

exploratório e de alcance restrito à implementação feita neste trabalho.

1.2 Estrutura do trabalho

O trabalho está organizado em quatro partes. Na primeira parte temos apenas a presente introdução. Na Parte 2, temos a revisão dos principais conceitos e teorias que são importantes para o tema deste trabalho e que serão, direta ou indiretamente, utilizados na parte experimental.

Os conceitos abordados na Parte 2 pertencem aos três campos que o tema deste trabalho pertence: *pairs trading*, análise de cointegração de séries temporais e algoritmos de otimização heurística.

A Parte 3 aborda a implementação construída para este trabalho dos algoritmos de otimização e de um *backtest*, que simula a estratégia de *pairs trading* com alguns ativos listados na B3. É também na Parte 3 que fazemos a aplicação do otimizador no *pairs trading* e analisamos os resultados obtidos.

Por fim, a Parte 4 contém as considerações finais, as sugestões de tópicos para investigação futura, bem como anexos com alguns dados e códigos utilizados na Parte 3.

2 REFERENCIAL TEÓRICO

Nesta Parte 2 trazemos uma síntese da teoria necessária para se construir os modelos e algoritmos empregados na Parte 3 e se delinear as conclusões apresentadas na Parte 4. Os tópicos abordados aqui são (i) a estratégia de *pairs trading*, (ii) cointegração de séries temporais e (iii) PSO e GA, ambos algoritmos de otimização heurística.

2.1 Pairs Trading

Pairs trading ou como é vulgarmente chamada de *Long&Short*[1] é , a grosso modo, uma estratégia que consiste em se tomar uma posição comprada em um ativo simultaneamente a uma posição vendida em outro, visando lucrar com uma distorção do preço relativo de ambos ativos. Veremos mais em detalhe esses conceitos mais adiante; porém antes vejamos um pouco do histórico desta estratégia.

Estratégias desse tipo estão presentes desde o início dos mercados de ações. Uma evidência disso é a estratégia com *ações irmãs* de Jesse Livermore no início do século XX[2]. Porém, foi somente com a expansão dos *hedge funds* na segunda metade do século passado e desenvolvimentos nas áreas de tecnologia da informação e comunicações que essa estratégia se popularizou. Em sua forma moderna, geralmente é atribuído à Nunzio Tartaglia e sua equipe no banco Morgan Stanley em 1987 a primeira implementação sistemática de *pairs trading*[3]. Desde então, a estratégia de popularizou sendo comum encontrá-la nas lâminas de vários fundos de investimento em operação.

[1] Rigorosamente, a estratégia *Long&Short* é uma espécie da qual *pairs trading* é gênero. Porém, diante do fato que coloquialmente os termos são empregados como sinônimo e que não é objetivo deste trabalho abordar as demais estratégias; faremos uso intercambiável dos termos neste trabalho.

[2] [4, p.21]

[3] [5, p.73]

2.1.1 Estratégia de *trading*

As principais características do *pairs trading* são as de que ele é: (i) *market-neutral*, (ii) de *reversão à média* e (iii) do tipo *arbitragem estatística*.

2.1.1.1 Modelo CAPM

Para definir de maneira mais formal o que é a característica de neutralidade de mercado do *pairs trading*, precisamos antes dos conceitos do CAPM, Capital Asset Pricing Model, proposto por William Sharpe. No CAPM o retorno esperado r_ω de um portfólio ω é modelado como um somatório de componentes de retorno:

$$E[r_\omega] = \beta_\omega(E[r_m] - r_f) + r_f + E[\theta_\omega] \tag{2.1}$$

O termo $(r_m - r_f)$ que representa o *prêmio de risco de mercado*, ou seja, o excedente de retorno esperado do mercado com relação ao retorno livre de risco. Geralmente, como ativo livre de risco é considerado um título público que remunera à taxa básica de juros r_f. Já o θ_ω pode ser interpretado como componente residual de retorno, descorrelacionado dos demais componentes e cujo valor teórico esperado é igual a zero[4]. Por fim, β é a sensibilidade do portfólio ao retorno esperado de mercado; definido como:

$$\beta_\omega = \frac{cov(r_\omega, r_m)}{var(r_m)} = \rho_{\omega,m}\frac{\sigma_\omega}{\sigma_m} \tag{2.2}$$

Esta formulação implica que, se o mercado tiver retorno positivo, o retorno esperado do portfólio será também positivo por um fator β, $\forall\beta > 0$. Dito de forma mais coloquial, o β indica o quão "correlacionado" um portfólio é com relação ao mercado.

2.1.1.2 Neutralidade de mercado

Agora podemos definir que as estratégias *market neutral* são aquelas cujo portfólio tem $\beta_\omega = 0$. Para se obter isto, é preciso comprar uma unidade do ativo 1 e simultaneamente vender x unidades do ativo 2 gerando:

$$\begin{aligned} E[r_\omega] =& \beta_1(E[r_m] - r_f) + r_f + E[\theta_1] - x\left(\beta_2(E[r_m] - r_f) + r_f + E[\theta_2]\right) \Longrightarrow \\ E[r_\omega] =& (\beta_1 - x\beta_2)(E[r_m] - r_f) + r_f(1 - x) + E[\theta_1 - x\theta_2] \end{aligned}$$

Para que o portfólio fique descorrelacionado com os movimentos de mercado, ou seja $\beta_\omega = 0$, é necessário que $x = \frac{\beta_1}{\beta_2}$. A estratégia de *pairs trading* implica uma cuidadosa

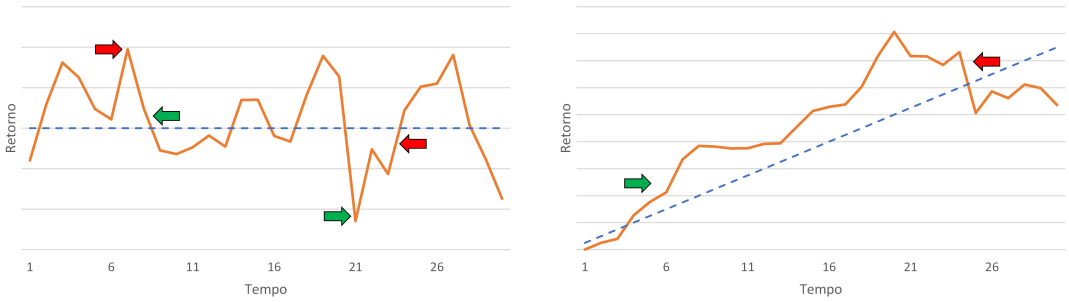
[4] [5, p.4]

escolha de x. Note que, uma vez que $\beta_\omega = 0$ o que resta é o termo $r_f(1-x)$, cujo retorno esperado é conhecido e o termo do retorno residual $E[\theta_1 - x\theta_2]$. As estratégias de *pairs trading* devem buscar prever o comportamento deste termo residual, de modo a obter lucro com a operação.

2.1.1.3 Reversão à média

Visto que o valor esperado $E[\theta] = 0$; logo devemos esperar que $E[\theta_1 - x\theta_2] = 0$. Ou seja, o termo residual deve apresentar forte comportamento de reversão à média. O termo residual pode ser equiparado ao que é popularmente conhecido como *spread*, ou seja a série temporal da relação dos preços dos dois ativos, ponderados por suas quantidades. Veremos em detalhes na seção 2.2 como encontrar pares de ativos que satisfaçam esta condição; bem como comparar essas estratégias às suas contrapartes, chamadas de *direcionais* ou *trend following* usando séries temporais. Por ora, vejamos apenas um exemplo esquemático do *trading design* de ambas modalidades na Fig.1.

Figura 1: Exemplo esquemático dos tipos de estratégia



(a) reversão à média (b) direcional

A linha laranja representa o preço ou retorno do ativo. As setas verdes representam sinal hipotético de compra. As setas vermelhas, sinal hipotético de venda. Note que no paradigma de reversão à média, a montagem da operação se faz quando o preço se afasta da média – linha pontilhada em (a); e no direcional, busca-se acompanhar a evolução da linha de tendência – linha pontilhada em (b).

2.1.1.4 Arbitragem estatística

Agora, conhecendo o que é o *spread* e sua relação com o termo de retorno residual, podemos partir para a última característica da definição de *pairs trading* utilizada neste trabalho – a *arbitragem estatística*. Nessa modalidade de *pairs trading* se faz uso de ferramental estatístico e de séries temporais sobre a série formada pelo *spread* para se definir quais pares realizar a operação e se prever o momento adequado para montá-la.

Isto explica a parte da "estatística" na arbitragem estatística. Convém fazer algumas observações sobre a parte da "arbitragem".

A definição de livro-texto de arbitragem é aquela situação cuja distorção nos preços de dois ativos é tal que é possível montar uma posição capaz de auferir lucro – ou no mínimo não sofrer prejuízo – incorrendo risco zero. A estratégia de *pairs trading* definitivamente não é livre de risco e, portanto, não se enquadra nesta definição canônica. O termo é empregado aqui no sentido de *arbitragem de preços relativos*, ou seja, busca-se explorar distorções nos preços que apresentam certa *relação* histórica ou matemática ao longo do tempo.

Abordagens alternativas de *pairs trading* A arbitragem estatística não é a única abordagem de *pairs trading* e talvez sequer seja a mais comum. Ela difere, por exemplo, da *arbitragem de risco de pares*; abordagem que busca explorar os desvios de preço que podem surgir entre os papéis de duas empresas num contexto de aquisição de controle societário[5]. A fundamentação econômica neste caso, reside em eventuais desvios na paridade das quantidades das ações na empresa consolidada. Num caso hipotético onde não há riscos, incertezas e especulação sobre o processo de aquisição, deve se esperar que não haverá desvios, portanto o termo residual de retorno é zero e nenhum lucro pode ser auferido com esse *trade*.

Uma outra abordagem importante é a abordagem *fundamentalista*. Neste caso, mediante o criterioso estudo dos balanços financeiros, planos de investimento e direção das empresas de um determinado setor ou atividade o analista projeta um intervalo de *spread* entre os preços das empresas que seja coerente com suas projeções. Na eventualidade do mercado desviar-se desse intervalo, monta-se a posição na expectativa de que o mercado revise suas estimativas e os preços relativos retornem ao patamar previsto.

Há ainda a abordagem da *análise técnica*. Essa se refere aos métodos de análise dos gráficos dos preços em conjunto com uma variedade de indicadores que acusam ao analista as oportunidades de se montar a operação. Esses indicadores são baseados nas séries históricas: (i) dos preços ou do *spread*; (ii) nos volumes financeiros e (iii) em padrões visuais formados pelo gráfico.

Um indicador muito utilizado para se montar estratégias de Long&Short pela análise técnica são as *bandas de Bollinger* (Fig.2). Não vamos aprofundar este tema, mas apenas apontar que o *setup* deste indicador consiste: (i) numa média móvel da razão dos preços

[5] [5, p.139-149]

(geralmente 21 dias); (ii) duas médias móveis afastadas da média central por dois desvios-padrão. Quando a média central encontra-se estacionada e o preço "rompe"uma das linhas superior ou inferior, é uma indicação para montar a operação. Devido a sua ampla utilização; na Parte 3 utilizaremos este critério de 2 desvios para construir um *benchmark* contra o qual nosso modelo será testado.

Figura 2: Indicador de bandas de Bollinger

Gráfico do Ibovespa de nov-21 a abr-22 mostrando as bandas de Bollinger. A linha preta representa a (i) média móvel de 21 dias e (ii) a linhas azuis, a média móvel espaçada por dois desvios da média central. Fonte: captura de tela de www.investing.com.

2.1.2 Fundamentação econômica

No item anterior, já mencionamos a fundamentação econômica das estratégias de arbitragem de risco e da arbitragem fundamentalista; que é a paridade das ações em um evento corporativo no primeiro caso e a performance das empresas, no segundo. Iremos brevemente observar a fundamentação econômica da estratégia de arbitragem estatística e da análise técnica, que são muito parecidas; se não idênticas.

A simples tendência de reversão à média, ainda que estabelecida por cuidadosa análise estatística, não se basta como racional econômico para explicar esta tendência, tampouco o porquê se espera que o termo $\theta_1 - x\theta_2$ seja zero. É a hipótese de eficiência dos mercados que estabelece a conexão da arbitragem estatística com sua fundamentação econômica. Segundo esta hipótese toda a informação conhecida sobre a realidade dos ativos em questão está refletida em seus preços. Porém, por vezes no curto prazo, pequenas flutuações ou desvios podem surgir e essas são o foco dos arbitradores estatísticos. O movimento em conjunto desses agentes, reforça a eficiência dos mercados, eliminando rapidamente estes

desvios[6].

Essa dependência da hipótese de mercado eficiente pode não ser segura o suficiente para vários agentes no mercado. Por esta razão eles procuram restringir o universo de combinações possíveis de ativos para aqueles que ofereçam um *rationale* econômico adicional. Alguma das restrições dos ativos pareados são:

Mesma empresa No mercado brasileiro, uma empresa pode emitir papéis com direito a voto (ações ordinárias ou ON) e papéis que recebem dividendos com preferência, porém sem voto (ações preferenciais ou PN). É possível encontrar casos onde o termo residual é grande o suficiente para se obter lucro com uma operação de Long&Short envolvendo as ON e PN da mesma empresa. Pode-se dizer que neste caso a fundamentação econômica é absoluta.

ETF's Vários EFT's e índices buscam replicar uma mesma carteira, com mínimas diferenças de portfólio. Ou ainda, índices que refletem movimentos econômicos correlacionados, como por exemplo ETFs de empresas exportadoras e o câmbio.

Intra-setorial Nesta modalidade, o universo de combinações é restrito às empresas que operam num mesmo setor ou que estão sujeitas aos mesmo fatores de risco sistêmico. Esta é uma maneira de comparar diretamente no que diferem duas empresas, isolando-se os demais efeitos sistêmicos. Esta modalidade carrega mais risco que as anteriores; pois através das suas atividades as empresas podem se diferenciar, modificar sua condições de competitivade, desconfigurando o comportamento estacionário do *spread*.

2.1.3 *Trading design*

Já sabemos que a estratégia de *pairs trading* requer um par de ativos cujo termo de retorno residual apresente tendência à média $E[\theta_1 - x\theta_2] = 0$ e que um portfólio deste tipo é construido com dois ativos, sendo que para cada unidade do ativo 1 devemos ter x unidades do ativo 2 de modo a satisfazer $x = \frac{\beta_1}{\beta_2}$. Para completarmos a visão geral sobre a execução dessa estratégia devemos tecer alguns comentários sobre sua montagem, reversão e desafios práticos envolvidos.

[6] [4, p.77-79]

2.1.3.1 Parâmetros do *trade*

O desafio de definir os *parâmetros* do *trade* é o tema central deste trabalho e é assunto da maior importância para a execução prática desta estratégia.

Montagem O desafio da montagem do *trade* implica definir a magnitude do desvio que o *spread* precisa apresentar para se abrir uma posição com boa expectativa de lucro. Vimos um exemplo disso na seção 2.1.1.4 quando mencionamos as bandas de Bollinger. Naquela técnica, uma posição de Long&Short é aberta assim que o preço romper a faixa de dois desvios-padrão da média, seja para baixo ou para cima. Não há nada intrinsecamente especial com relação aos dois desvios; porém é um nível amplamente aceito e utilizado no mercado.

A definição de quantos desvios se utilizar suscita um *trade-off* entre (i) uma entrada mais tardia (mais desvios) com potencial de lucro maior, porém menos frequente ou (ii) uma entrada mais próxima à média, mais frequente e com ganhos menores. Esta decisão tem grande impacto no retorno total que esta estratégia pode entregar; bem como afeta sobremaneira o risco do portfólio.

Reversão A reversão diz respeito ao atingimento do objetivo do *trade*. Este parâmetro pode ser definido como um valor financeiro, o atingimento da média do *spread* ou mesmo uma "distância percorrida" em termos de desvio-padrão. Neste trabalho utilizamos como parâmetro uma fração da volatilidade histórica do *spread* em direção à média. Optamos por esta abordagem devido a facilidade de implementação.

Stop-loss Diferente dos dois parâmetros anteriores, o *stop-loss* não é um parâmetro fundamental do *trade*; mas sim, uma técnica de gestão de risco. Ele significa encerrar uma posição que não está convergindo para a média conforme as expectativas, com vistas a evitar um eventual agravamento das perdas geradas por este erro. Neste trabalho utilizamos como *stop-loss* uma fração da volatilidade histórica do *spread* em direção oposta à média.

***Designs* alternativos** Pode-se dizer que o desenho para montagem da estratégia que apresentamos aqui é um das alternativas mais simples e diretas dentre as quais se vê no mercado. Optamos por seguir desta maneira, dado que nosso objetivo focal é analisar a contribuição dos algoritmos de otimização na definição desses parâmetros ou mesmo na

otimização de outros quaisquer.

Convém apenas mencionar algumas alternativas mais sofisticadas, dentre a imensa diversidade que se pratica no mercado. Por exemplo, o uso de previsão em séries temporais, onde se modela a evolução do *spread* com modelos ARIMA, GARCH, etc. Há também o uso de *trailing stops*; em que o encerramento da posição é feito por um nível que "acompanha" a alta do ativo, permitindo que um erro na expectativa de convergência que favoreça o *trade* não seja prematuramente encerrado. Há também abordagens não-paramétricas, que prescindem que o modelo arbitre níveis específicos, como aquela proposta por Vidyamurthy[7]. Numa interseção com a análise técnica, há estratégias que usam vários indicadores gráficos simultaneamente que geram um "índice de força" a medida que mais instrumentos acusam favoravelmente ao *trade*. Por fim, vale mencionar o uso conjugado destas técnicas com a análise fundamentalista das empresas; o que, por óbvio, deixa de ser uma estratégia totalmente automatizada e estatística, mas pode trazer grandes benefícios em termos de assertividade e retorno.

2.1.4 Considerações para aplicação prática

Para encerrar esta seção, vamos enumerar alguns dos desafios práticos que surgem na implementação das estratégias de *pairs trading*. O objetivo disso é já antecipar e dar transparência das escolhas feitas no desenvolvimento da metodologia dos experimentos na Parte 3.

Liquidez É de extrema importância que ambos os papéis tenham volume de negociação diária suficiente para se permitir tanto a entrada, quanto a saída da totalidade da posição sem afetar excessivamente o mercado. A este fenômeno de impacto no preço devidao a baixa liquidez, chama-se *price slippage*.

Custos A lei dos grande números joga a favor das estratégias baseadas em estatística. Porém, isso implica fazer várias operações no mercado, o que, por sua vez, joga a favor da rentabilidade das corretoras em detrimento do seu portfólio. Por esta razão, deve-se dar especial atenção aos custos de corretagem, emolumentos, impostos etc e preferencialmente incluí-los na modelagem.

Venda a descoberto A parte "short" do Long&Short requer cuidados adicionais. Sendo um mercado de balcão (OTC), é preciso garantir que há contratos suficientes disponíveis para se alugar antes de se montar a operação. Além dos custos normais de

[7] [5, p.118-138]

operação, há a taxa de aluguel – que remunera o cedente do aluguel – e as taxas de corretagem específicas para operações de venda a descoberto.

Garantias Ainda como consequência da parte "short" a B3 exige depósito de garantia de no mínimo 90% [8] do valor financeiro da operação descoberta.

Tamanho da posição O tamanho de cada posição de *pairs trading* com relação ao portfólio como um todo, ou mesmo, a decisão sobre qual financeiro fazer cada operação de Long&Short é, por si só, um tópico que merece muitas páginas a respeito; mas foge ao nosso escopo. Como sugestão de leitura a quem possa interessar, o critério de Kelly determina estatisticamente o tamanho da "aposta" que maximiza o retorno global após grande número de rodadas.

2.2 Cointegração

Como vimos na seção anterior, para que um par de ativos seja elegível à estratégia de *pairs trading* é indispensável que seu *spread* apresente a característica de reversão à média. Na presente seção, apresentaremos os fundamentos teóricos da *cointegração* de séries temporais e um dos métodos para obtê-la, desenvolvido por Engle & Granger[9]. No entanto, antes, precisamos apresentar superficialmente alguns conceitos básicos de séries temporais, indispensáveis para a formulação desse método.

2.2.1 Séries estacionárias

2.2.1.1 Séries temporais e estacionariedade

Uma série temporal discreta é definida como uma sequência ordenada no tempo de uma dada variável aleatória; formalmente chamado de um *processo estocástico* definido como[10]:

$$\begin{aligned} &\{y(z,t), z \in \zeta, t \in \tau\} \\ &\{y\}_{t=1}^{T} = \{y_1, y_2, ..., y_{T-1}, y_T\} \end{aligned} \tag{2.3}$$

Onde t representa o índice que indica o tempo, pertencente ao intervalo temporal τ. Já $y(*,t)$ representa a variável aleatória associada ao instante t e z representa o processo estocástico de formação desta variável aleatória, que pertence um dado espaço ζ. O

[8] Valor do deságio válido em outubro-22, aplicados às LFT emitidas pelo Tesouro Federal. Outros tipos de ativo tem deságios distintos.

[9] [6]

[10] [7, p.3]

estudo das séries temporais se ocupa de modelar este processo estocástico z, seus padrões e sua estrutura interna de modo a explicar e prever o comportamento de y.

Uma série temporal é dita *estacionária em covariância*[11] se apresenta as seguintes características:

$$\begin{aligned} E\left[y_t\right] &= \mu < \infty,\ \forall t \in \tau \\ E\left[(y_t - \mu)(y_{t-j} - \mu)\right] &= \gamma_j,\ \forall t, j \in \tau \text{ e } j \in \mathbb{Z} \end{aligned} \tag{2.4}$$

Isto significa, para qualquer t que se escolha, a série sempre terá uma média finita igual a μ e uma covariância igual γ_j; ou seja, para qualquer janela de duração j a série terá a mesma média e uma covariância que depende apenas de j.

2.2.1.2 Ruído branco

A série estacionária mais simples que há, se chama *ruído branco* e é um processo ϵ_t caracterizado pelas condições descritas pelas equações (2.5). Frequentemente ϵ_t é uma variável aleatória extraída de uma distribuição normal $N \sim (0, \sigma^2)$, sendo então chamada de *ruído branco Gaussiano*. Na Fig.3 o gráfico[12] da série oscila em torno de sua média zero e em ambos os gráficos de ACF[13] e PACF[14] indicam que não há qualquer tipo de autocorrelação significativa.

$$\begin{aligned} E[\epsilon_t] &= 0 \\ E[\epsilon_t^2] &= \sigma^2 \\ E[\epsilon_{t_1}\epsilon_{t_2}] &= 0,\ \forall\ t_1 \neq t_2 \end{aligned} \tag{2.5}$$

2.2.1.3 Autocorrelação e modelo AR(p)

A *autocorrelação* ou *correlação serial* é uma propriedade que algumas séries temporais apresentam de que a série se parece com uma versão de si própria defasada por alguns instantes de tempo. A covariância de duas variáveis aleatórias é dada por

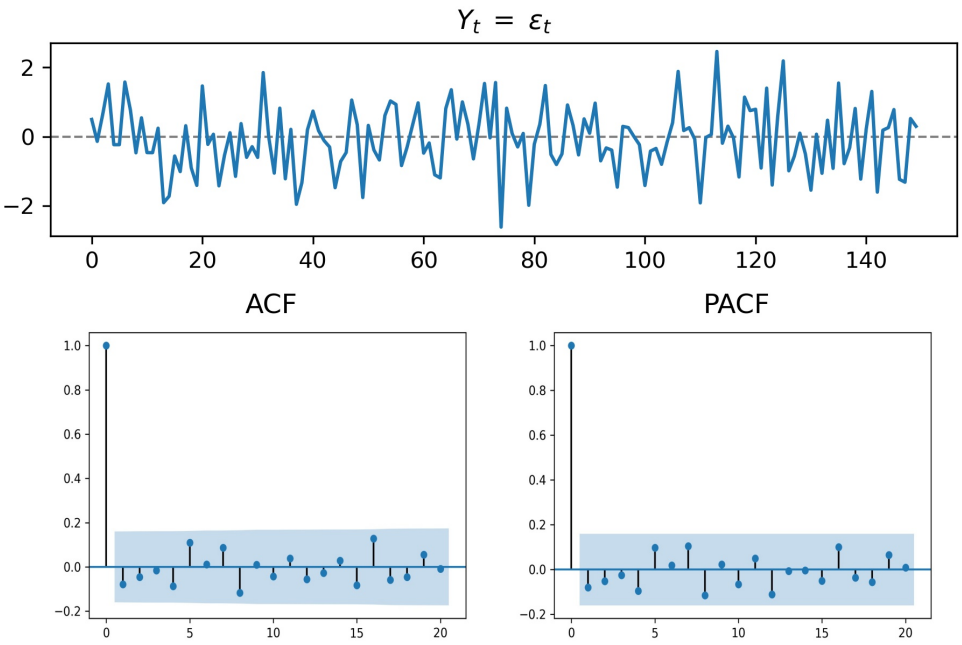
[11] Também chamada de estacionariedade em sentido amplo, é a característica mais utilizada na manipulação de séries temporais. Há também a chamada *estacionariedade em sentido estrito*, na qual a distribuição conjunta das probabilidades de duas sequencias diferentes da série são iguais, independentemente de qual seja o tempo escolhido. A estacionariedade em sentido estrito implica necessariamente a estacionariedade em sentido amplo. Neste trabalho, nos referimos à estacionaridade no sentido amplo sempre que mencionarmos que uma série é estacionária.

[12] os gráficos nesta seção foram construídos a partir de séries artificiais geradas por números pseudo-aleatórios. Essas séries utilizam o algoritmo de séries aleatórias do pacote *numpy* do Python, com *seed*=42, portanto elas podem ser comparadas entre si no sentido de que o termo estocástico ϵ_t é idêntico.

[13] ACF ou função de autocorrelação. Veja seção 2.2.1.3

[14] PACF ou função de autocorrelação parcial. Veja seção 2.2.1.3

Figura 3: Gráfico de ruído branco gaussiano, ACF e PACF



$cov(X, Y) = E\left[(X - E[X])(Y - E[Y])\right]$. Para se comparar a covariância de um mesmo processo estocástico que seja estacionário, basta aplicarmos um atraso j na série; resultando em $cov(y_t, y_{t-1}) = E\left[(y_t - \mu)(y_{t-1} - \mu)\right] = \gamma_j$.

ACF Como vimos que, para séries estacionárias, a média é constante e independe do tempo; pudemos substituir os valores esperados das janelas de y por μ. Portanto, ao se substituir o termo de autocovariância γ na equação geral para correlação, derivamos a *equação de autocorrelação* ou ACF, conforme (2.6)[15]. Por fim, dado que em casos práticos dispomos apenas de uma amostra da série, a autocorrelação amostral se dá pela equação (2.7). A função de ACF desempenha papel fundamental na identificação do q das séries que tenham características que possam ser modeladas com MA(q).

$$corr(X, Y) = \frac{cov(X, Y)}{\sqrt{\sigma_X^2 \sigma_Y^2}} \Longrightarrow \frac{\gamma_j}{\gamma_0} = \rho_j \tag{2.6}$$

$$\hat{\rho}_j = \frac{\sum_{t=j+1}^{T} (y_t - \hat{\mu})(y_{t-j} - \hat{\mu})}{\sum_{t=1}^{T} (y_t - \hat{\mu})^2} \tag{2.7}$$

PACF A *função de autocorrelação parcial* é construída a partir da ACF, e explicita correlação condicional para cada termo adicionado. O PACF é útil para se determinar a

[15] [8, p.51-52]

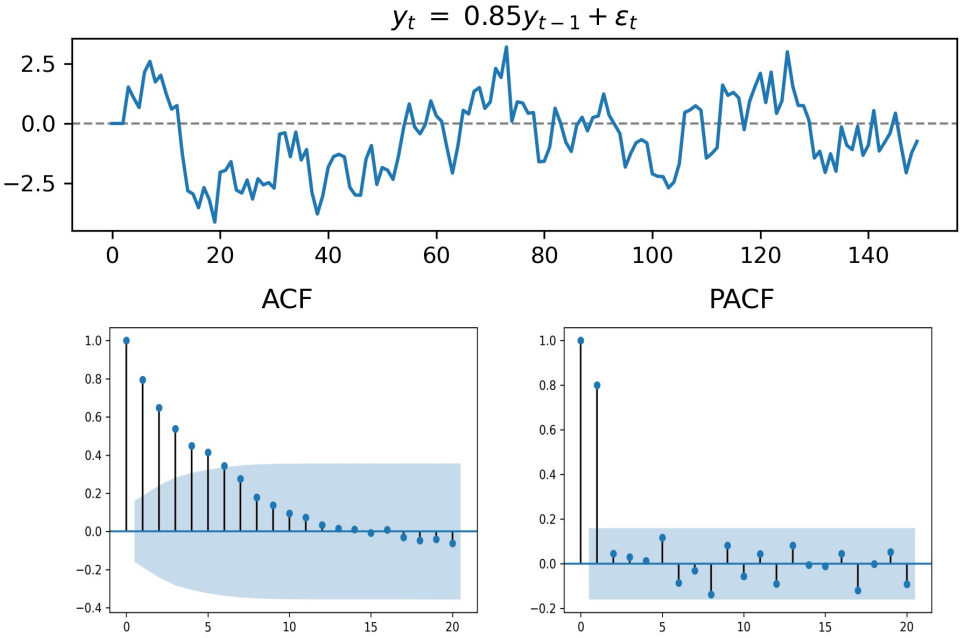
ordem q de um modelo AR(p).

AR(q) Na sua forma mais simples – autocorrelação de primeira ordem ou AR(1) – a modelagem da série é dada pela equação (2.8). O valor assumido por ϕ é de especial interesse; pois além de indicar a "persistência" do termo AR, a estacionariedade do modelo AR(1) depende que a condição $|\phi| < 1$ seja verdadeira[16]. Se $|\phi| > 1$ é fácil perceber que, num longo período de tempo, a série diverge para $|y_\infty| = \infty$ devido ao acúmulo dos choques estocásticos. Um caso especial, chamado de *random walk* acontece quando $|\phi| = 1$, que veremos em detalhe na seção 2.2.2.1. O modelo AR(1) pode ser generalizado para um modelo de ordem p, chamado de AR(p), conforme descrito em (2.9).

$$y_t = \phi y_{t-1} + \epsilon_t \tag{2.8}$$

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_{p-1} y_{t-p+1} + \phi_p y_{t-p} + \epsilon_t \tag{2.9}$$

Figura 4: Gráfico de série AR(1) e seus ACF e PACF



2.2.1.4 Modelo MA(q)

Um processo de média móvel de primeira ordem MA(1) é definido conforme a equação (2.10). Diferentemente do modelo AR, o modelo MA sempre será estacionário, para

[16] As condições para estacionariedade de modelos de ordem superior são mais complexas e exigem maior formulação, que foge ao escopo deste trabalho.

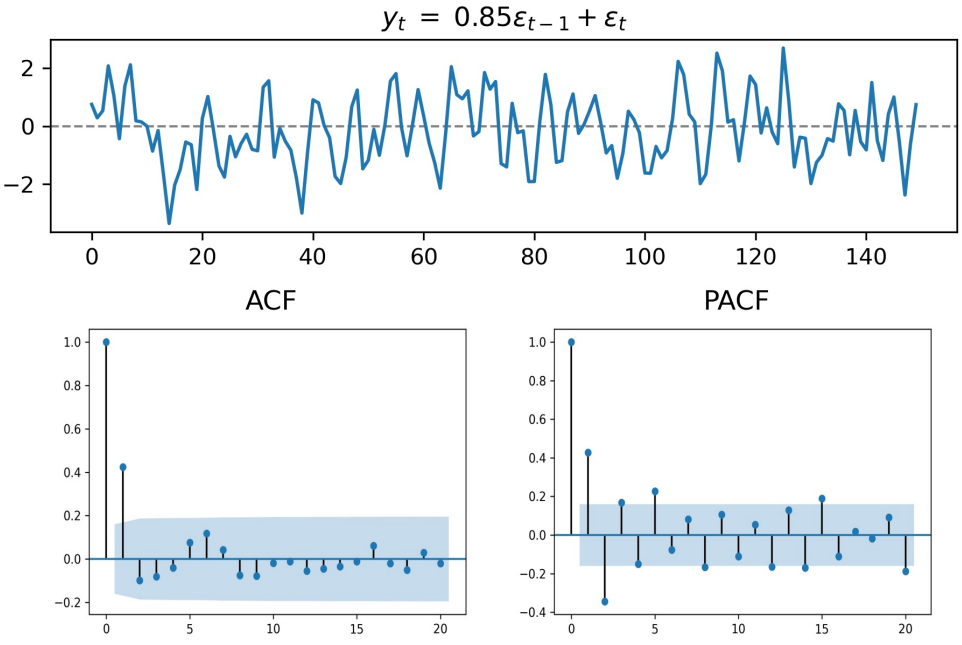
quaisquer valores que o coeficente θ possa assumir. Na (2.11) vemos a generalização para o modelo com q atrasos.

$$y_t = \theta\epsilon_{t-1} + \epsilon_t \tag{2.10}$$

$$y_t = \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + ... + \theta_{q-1}\epsilon_{t-q+} + \theta_q\epsilon_{t-q} + \epsilon_t \tag{2.11}$$

Ao passo que uma série AR(p) é identificada por p atrasos relevantes no PACF e um padrão decrescente ou alternante no ACF; uma série MA(q) apresenta q períodos com autocorrelação significativa na ACF e por um padrão decrescente ou alternante em seu PACF[17].

Figura 5: Gráfico de série MA(1) e seus ACF e PACF



Modelo ARMA(p,q) Para fechar a discussão sobre modelos de séries estacionárias mencionaremos o modele ARMA(p,q); que nada mais é que um modelo AR de ordem p combinado com um modelo MA de ordem q. A equação que descreve este processo pode ser obtida somando as equações (2.9) e (2.11).

[17] [7, p.12]

2.2.2 Séries não-estacionárias

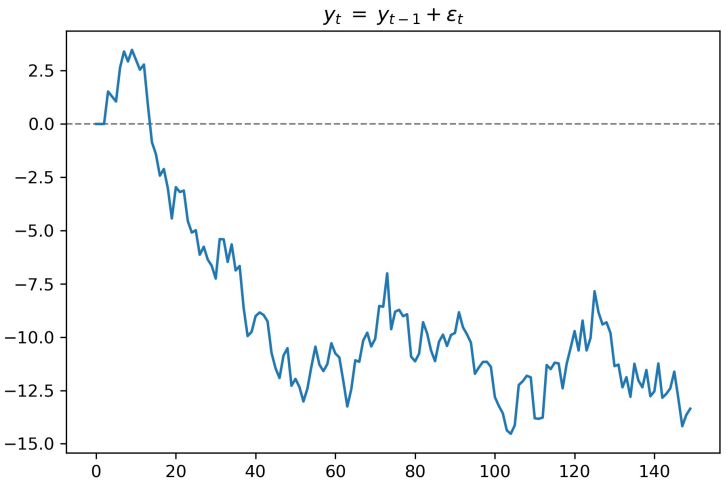
2.2.2.1 *Random Walk*

O processo estocástico definido pela equação (2.12) é chamado de *random walk*. Note que ele é idêntico a uma modelo AR(1) com $\phi = 1$, violando a condição de estacionariedade, conforme dito acima. Sabendo que ϵ_t tem uma distribuição simétrica em torno da média, y_t tem 50% de chances de aumentar ou de diminuir, o que se reflete no longo prazo como uma série que evolui sem nenhuma tendência ou padrão discernível (ver Fig.6).

$$y_t = y_{t-1} + \epsilon_t \tag{2.12}$$

O processo *random walk*, também chamado de *difference-stationary* é amplamente uti-

Figura 6: Gráfico da série *random walk*



lizado em finanças para se modelar uma série temporal de preços sujeitas a choques de log retornos que não tem comportamento de reversão à média nem tem previsibilidade prática[18].

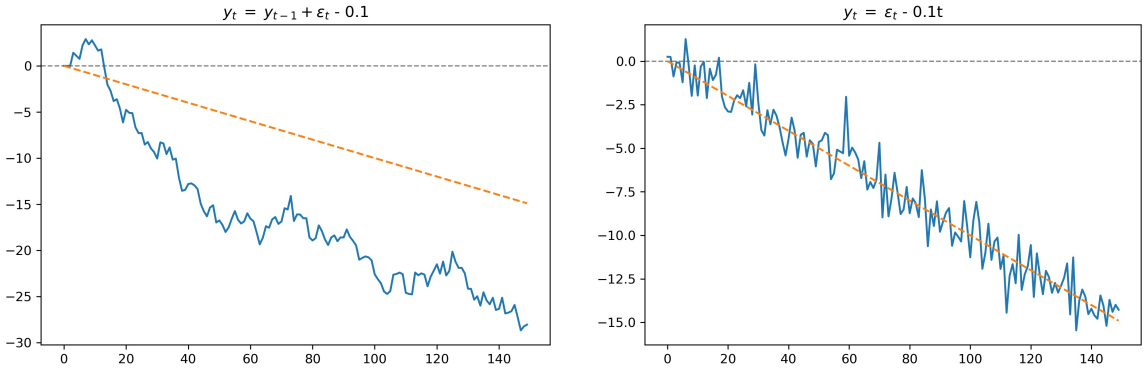
Random walk* com *drift Uma variação do modelo de *random walk* é obtida introduzindo-se uma constante d na equação (2.12) resultando em: $y_t = d + y_{t-1} + \epsilon_t$ (ver Fig.7). Esse termo adiciona uma tendência que evolui com o tempo $t \cdot d$; muito diferente do que acontece quando adicionamos uma constante qualquer no modelo ARMA. No caso do ARMA, a constante tem o efeito de redefinir a média da série.

[18] [9, p.72]

Série de tendência estacionária Convém mencionar aqui as diferenças do *random walk* com *drift* para um modelo estacionário com tendência do tipo: $y_t = \beta_0 + \beta_1 t + \epsilon_t$. Neste caso, o termo de ruído branco é estacionário em torno da média $\beta_0 + \beta_1 t$, que evolui com tempo. E a variância é constante, idêntica à variância do ruído branco (2.3). Esta série pode ter o termo de tendência eliminado através de uma regressão linear, resultando em uma série estacionária. O mesmo não pode ser obtido com o *random walk* com *drift*, que precisa ter sua raiz unitária removida, como veremos adiante.

Figura 7: Gráfico da série *random walk* com drift.

Figura 8: Gráfico da série tendência estacionária.



Linha pontilhada representa o termo de tendência d no modelo random walk *e β no modelo de tendência estacionária; ambos -0.1.*

2.2.2.2 Raiz unitária

Como vimos, a presença de um coeficiente $\phi = 1$ no modelo AR é suficiente para torná-lo não-estacionário. Quando se reescreve a equação (2.9) na forma da equação característica do modelo, encontraremos pelo menos uma raiz fora do círculo unitário[19], daí advém o nome *raiz unitária*. O modelo *random walk* é o protótipo mais simples de um modelo que contém uma raiz unitária. Obviamente que que não podemos utilizar um modelo ARMA para modelar uma série não-estacionária; no entanto há uma técnica, chamada de *diferenciação* que permite eliminar a raiz unitária tornando-a apta a ser modelada com ARMA.

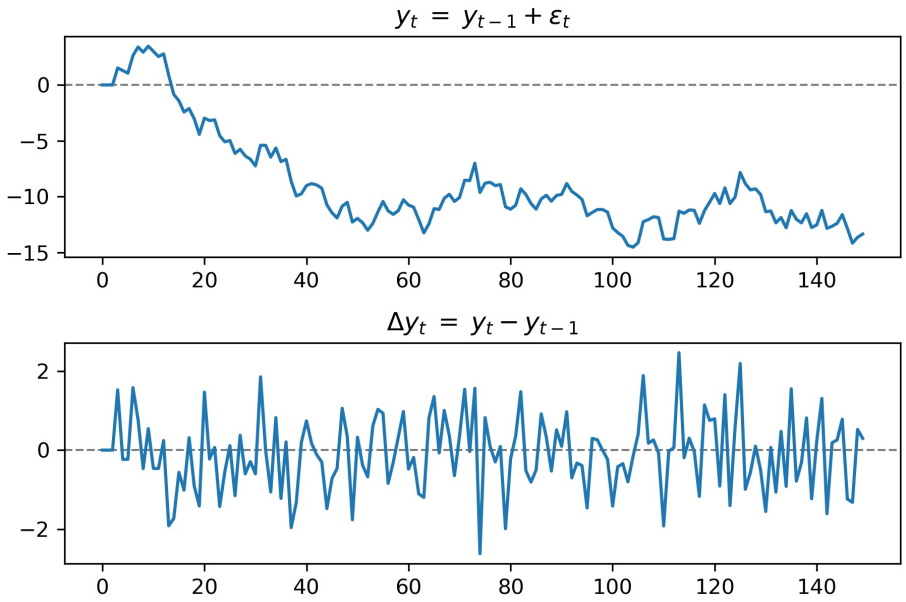
A técnica de diferenciação consiste em se subtrair a série y_t de uma versão atrasada de si própria y_{t-1}. É possível que a série contenha múltiplos de raízes unitárias, o que requer sucessivas diferenciações até se obter um modelo estacionário. Nomeia-se esse modelo com diferenciação de ARIMA(p,d,q) onde o "I" significa *integrado*[20] e o parâmetro d significa

[19] Diz respeito ao círculo unitário inscrito num plano complexo que mapeia as raízes da equação característica.

[20] O termo integrado é utilizado, pois é o inverso da operação de diferenciação.

a ordem de integração, ou seja, a quantidade de diferenciações aplicadas à série. Em finanças, o procedimento de calcular os retornos de uma série de preços – série esta não-estacionária – pela diferença dos logaritmos dos preços, se assemelha ao procedimento de eliminação a raiz unitária.

Figura 9: Série *random walk* vs. série $I(1)$



Teste de raiz unitária Para se detectar a presença de uma raiz unitária em uma série – e portanto se ela é estacionária ou não – é amplamente utilizado o teste de Augmented Dickey-Fuller (ADF). O teste de ADF tem a capacidade de determinar a presença de raizes unitárias mesmo em modelos com ordem de diferenciação $d > 1$.

Imagine uma série AR(p) que pode conter tanto *drift* quanto tendência estacionária para a qual se deseja testar a presença de raízes unitárias. Usando regressão (equação 2.13[21]) é possível construir um estimador ψ sobre o qual são montadas seguintes hipóteses do teste: $\begin{cases} H_0 : \psi = 1 \\ H_1 : \psi \neq 1 \end{cases}$. Aos rejeitar-se a hipótese nula, pode-se dizer que a série não contém nenhuma raiz unitária e é, portanto, estacionária.

Dickey e Fuller calcularam as distribuições deste estimador ψ para diferentes restrições (cada uma representando um dado cenário i) aplicadas à regressão, onde o termo c_i pode ser: $(i = 1)$ sem drift e sem tendência, $(i = 2)$ com drift e sem tendência e $(i = 3)$ com drift e com tendência. A hipótese nula é rejeitada quando a estatística de interesse τ_i (2.14) para o tipo de série i é menor que um dado valor crítico de τ_{i^*}, conforme cálculos

[21] [9, p.76-77]

elaborados por Dickey e Fuller para cada intervalo de confiança e tamanho da amostra[22]

$$x_t = c_i + \psi x_{t-1} + \sum_{i=1}^{p-1} \phi_i x_{t-q} + \epsilon_t \tag{2.13}$$

$$\tau_i = \frac{\hat{\psi} - 1}{\sigma_{\hat{\psi}}} \tag{2.14}$$

Procedimento de teste O teste de ADF requer como parâmetro o tipo de modelo que melhor descreve a série, aquilo que chamamos de cenários i na seção anterior. Isto requer um conhecimento prévio sobre a existência desses termos constantes ou de tendência; o que em situações práticas geralmente não é possível. Além disso, a qualidade do teste ADF é prejudicada a cada restrição que é excluída do termo c_i. Por este motivo, como guia prático, a seguinte sequência é recomendada[23].

1. Inicia-se o teste assumindo o modelo mais amplo, com *drift* e com tendência. Se rejeitada H_0, conclui-se que a série é estacionária e o teste está encerrado.

2. Caso contrário, testa-se o modelo apenas com a tendência. Se rejeitada H_0 conclui-se que a série é estacionária e encerra-se o teste.

3. Por fim, anula-se o termo do *drift* e repete-se o teste. Caso H_0 permanece válida, conclui-se que a série tem definitivamente uma raiz unitária.

2.2.3 Teste de Engle & Granger

2.2.3.1 Definição e intuição

Agora temos a fundamentação teórica mínima para abordarmos o teste de cointegração de séries temporais. Duas séries y_t e x_t são ditas *cointegradas* se (i) ambas são são processos estocásticos não-estacionários, ou seja, integradas de ordem I(1); e (ii) que exista uma relação linear do tipo $u_t = y_t - \beta x_t$ onde a série resultante u_t é estacionária[24]

Uma noção intuitiva de cointegração é a de que mesmo que duas séries se pareçam com *random walk* variando aleatoriamente, a "distância" entre essas duas séries apresenta certo

[22] James MacKinnon apresentou valores alternativos para os valores críticos, no seu paper de 2010 pela Queen's University, *Critical Values for Cointegration Tests*. Na parte 3, utilizamos estes valores para se determinar o p-value do teste ADF.

[23] [10]

[24] Numa definição mais ampla que permite incluir ordens de cointegração d; é possível de dizer que séries cointegradas apresentam uma relação linear que resulte numa série que tenha ordem de integração $d' < d$.

padrão de estabilidade ou, mais rigorosamente, um padrão estacionário. A cointegração indica uma relação de longo prazo entre as séries, e portanto, que os choques estocásticos tem origem num fenômeno que afeta ambas variáveis de alguma forma.

Existem várias técnicas diferentes para se testar a cointegração. Dentre elas destaca-se o método de "duas etapas" de Engle&Granger e o método de Johansen. A principal diferença entre eles é que o primeiro funciona exclusivamente para sistemas bivariados, ou seja, só pode testar a cointegração entre duas séries, ao passo que Johansen pode fazê-lo para sistemas multivariados. Apesar da superioridade do teste Johansen com respeito ao número de variáveis e, segundo alguns autores[25] também apresenta vantagens na performance empírica; para o os objetivos deste trabalho o método de Engle&Granger se faz suficiente – dado que a estratégia de *pairs trading* se basta na análise de pares de séries. A importância do teste de cointegração para nossos objetivos está clara agora. Conforme vimos na seção 2.1.1 a estratégia de *pairs trading* tem como condição a reversão à média e exatamente esta informação é obtida quando dois ativos, cujos preços são séries não-estacionárias, passam num teste de cointegração.

2.2.3.2 Metodologia do teste

O método de Engle&Granger consiste de duas etapas. A primeira é estimar o coeficiente β da relação linear entre as séries não-estacionárias y_t e x_t, conforme equação 2.15. Isto é feito mediante uma regressão linear de mínimos quadrados onde β é o coeficiente linear e c é o intercepto[26]. Os resíduos desta regressão dão origem à série u_t. Caso a regressão seja aceitável para o intervalo de confiança do coeficiente β e o R^2 desejados, se procede à segunda etapa.

$$u_t = y_t - \beta x_t + c \tag{2.15}$$

A premissa sobre a qual o teste se baseia é a que se $u_t \sim I(0)$ então a relação linear 2.15 implica relação de cointegração entre as séries. Caso $u_t \sim I(d \geq 1)$, então não há essa relação. Daí derivam-se as hipóteses do teste de Engle&Granger que podem ser testadas por meio da aplicação do teste ADF sobre os resíduos u_t para se determinar se a série u_t não possui uma raiz unitária. No caso da série não ter raiz unitária, diz-se que y_t e x_t são séries cointegradas. Vale notar que o valor de τ_i obtido pelo teste ADF deve ser comparado com a tabela elaborada por MacKinnon, dado que β foi estimado, por não ser

[25] [11]

[26] É recomendável adicionar-se o intercepto nas aplicações em finanças, dado que *a priori* não há nenhuma garantia de que a série obtida tenha média zero.

conhecido a princípio[27]. Além disso, o cenário i de τ_i deve respeitar o modelo empregado na equação (2.15); no caso proposto aqui, deve-se tomar o cenário ($i = 2$) com *drift* e sem tendência[28].

2.2.4 Cointegração vs. correlação

É comum presenciar entre os *traders* o debate acerca das vantagens de usar cointegração ou correlação como métodos para seleção de pares de ativos. Essa disputa foi inclusive objeto de estudo da dissertação de mestrado de Ana Rayes; onde a autora concluiu mediante estudo empírico do mercado brasileiro que em termos de retorno financeiro bruto o método de correlação se mostrou superior. Porém, em termos de rentabilidade, o método de cointegração se destaca, devido à sua maior taxa de acerto[29]. Aqui, abordaremos brevemente os aspectos teóricos envolvidos neste debate, que sugerem a inadequação da correlação para a seleção dos pares.

Imagine uma regressão linear do tipo $y_t = \beta x_t + u_t$, para qual as variáveis podem ser não-estacionárias. Caso não exista um β que resulte em resíduos I(0), então o método dos mínimos quadrados pode dar origem a uma dita *regressão espúria*[30], cujos resultados não são confiáveis. Geralmente as regressões espúrias apresentam coeficientes com p-value baixo e com R^2 muito alto. Isto é fácil de se perceber devido ao fato do denominador do R^2 ser computado[31] como $\sum_{t=1}^{T}(y_t - \overline{y_t})$. Se y_t é uma série *random walk*, erroneamente se assume que tenha uma média constante e, à medida que o tamanho da amostra cresce o denominador aumenta muito; o que resulta: $\lim_{T\to\infty} R^2 = 1$. Existem vários métodos para se corrigir este problema[32], no entanto, não sem evocar outros problemas. Por este motivo, os métodos de cointegração são mais indicados quando as variáveis apresentam comportamento não-estacionário.

---

[27] [12, p.54]

[28] Não obstante, o método sequencial descrito na seção 2.2.2.2 pode ser empregado como uma medida de cautela adicional.

[29] [13]

[30] A demonstração matemática completa pode ser encontrada em [14, p.557-561]

[31] $R^2 = 1 - \dfrac{\sum_{t=1}^{T} \hat{\epsilon_t^2}}{\sum_{t=1}^{T}(y_t - \overline{y_t})}$

[32] [14, p.561]

2.3 Algoritmos de otimização

2.3.1 História

Historicamente[33] o conceito de otimização sempre foi central em matemática. Desde a Antiguidade, problemas de geometria apresentavam oportunidades para aplicação de técnicas de otimização, tais como o problema da rainha Dido de Cartago que desejava maximizar a área de sua cidade para um dado perímetro. Este problema – cuja solução é um semicírculo – foi estudado pelo matemático grego Zenodoro (200 a.C. - 140 a.C.). Os conceitos de cálculo também tem papel importante no estudo de otimização e convergência de séries infinitas a valores bem definidos.

Porém, foi com o desenvolvimento da computação em meados do século XX que o uso de algoritmos de otimização numérica experimentou grandes avanços, tanto no escopo de aplicação quanto na diversidade de ferramentas. Um marco importante em programação linear foi a introdução do método *Simplex* por George Dantzig (1914–2005). Com a expansão da capacidade computacional, novas aplicações de algoritmos de otimização patrocinam uma grande revolução em curso nas áreas de engenharia e de inteligência artificial.

2.3.2 Visão geral

2.3.2.1 Definição

Genericamente, um problema de otimização pode ser definido como:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{sujeito a} \quad & x \in \chi \end{aligned} \tag{2.16}$$

Sendo que $f(x)$ é a *função objetivo* que se busca otimizar – nesta formulação, se busca encontrar o valor *mínimo* de $f(x)$. As variáveis desta função são descritas por x, que é um vetor n-dimensional que contém os valores que estas n-variáveis podem assumir $x = [x_1, x_2, \ldots, x_n]$. E χ representa o conjunto de *soluções possíveis* que as x_n variáveis podem assumir, em outras palavras, é o espaço de busca do otimizador. As soluções contidas em χ que satisfazem o problema de otimização são representadas por x^*.

[33] [15, p.2-5]

2.3.2.2 Classificações

Quando χ não pode assumir qualquer valor, dito de outra forma, que o conjunto de soluções possíveis é restrito a um dado intervalo de valores, dizemos que o problema é *restrito*.

Quando a função $f(x)$ possui apenas um único vetor x^*, dizemos que a função é *unimodal*. Ao contrário, quando mais de dois vetores distintos são solução para o problema de otimização, dizemos que a função é *multimodal*.

Quando for possível diferenciar a função $f(x)$ podemos lançar mão de uma série algoritmos chamados de *determinísticos*. Nestes casos, o processo de otimização segue uma série de passos em que o valor da próxima iteração é inteiramente definido pela formulação do algoritmo. Geralmente o estudo da primeira e da segunda derivada permitem analiticamente concluir se a otimização atingiu um ponto de mínimo (ainda que um mínimo local) quando as seguintes condições forem satisfeitas:

$$f'(x) = 0$$
$$f''(x) > 0$$

A primeira condição assegura que uma pequena perturbação em x não afetará o resultado de $f(x)$. Já a segunda condição estabelece que o ponto em questão encontra-se em um "vale" ou região côncava. Em conjunto, estas condições asseguram que o ponto em análise trata-se de um ponto de mínima. Alguns exemplos deste tipo de algoritmo são:

- Método de Newton
- Método do gradiente descendente
- Momentum
- Adam

Porém, quando não for possível diferenciar $f(x)$, seja pela complexidade analítica envolvida ou pela impossibilidade de formulá-la devido a complexidade do fenômeno real envolvido; um outro tipo de algoritmo de otimização deve ser empregado, esses são chamados *heurísticos* [34] Neste tipo de algoritmo, a precisão e a certeza em torno da condição de minimização é substituída por um resultado aproximado. Em muitos casos, é intro-

[34]Otimizadores heurísticos são a escolha ideal quando um problema é considerado não ter solução em tempo polinomial (NP-completo). [16, p.10]

duzido um elemento estocástico no processo de otimização. Alguns exemplos deste tipo são:

- Métodos estocásticos
- Métodos diretos
- Métodos populacionais ou naturais

A seguir, vamos aprofundar nos métodos populacionais ou naturais que serão empregados neste trabalho.

2.3.2.3 Métodos de otimização natural ou populacional

A grande diferença entre os métodos de otimização natural versus os demais métodos é que eles tem uma probabilidade muito maior de encontrar um ponto de mínimo global ao invés de ficar preso em um mínimo local[35]. Isto se dá, porque ao invés de iniciar o processo de otimização partindo de um único ponto no espaço de busca, os algoritmos naturais simulam uma "população" de soluções possíveis com posição inicial estrategicamente dispersa; cobrindo um espaço maior na busca, já na primeira iteração. Geralmente esses algoritmos tem elementos estocástico que produzem variações aleatórias cuidadosamente controladas para ampliar sua capacidade de encontrar soluções ótimas.

Muitos desses algoritmos buscam inspiração em fenômenos da natureza, que é pródiga em otimizar seus recursos.[36] Daí surge a denominação de métodos *naturais*. Alguns exemplos desse tipo são:

Simulated Annealing Inspirado em termodinâmica, este algoritmo simula o processo de variação da temperatura – representada pela amplitude do termo estocástico – na formação de cristais – estes representam o conjunto das melhores soluções. Iterativamente as soluções sub-ótimas são substituídas por soluções melhores, à medida que a "temperatura" é reduzida.

Ant colony É inspirado na capacidade das formigas de encontrar o caminho mais curto em direção ao alimento através de feromônios. Nesta aplicação, a intensidade dos

[35]Obviamente que essa probabilidade depende do ponto de partida escolhido para iniciar-se o algoritmo. Caso a região onde se encontra o mínimo global seja conhecida, a afirmação feita não procede.

[36]A título de ilustração deste ponto, é o teorema elaborado por Thomas C. Hales em 1999, que demonstra que o formato hexagonal – encontrado nas colmeias de abelhas – é o formato que minimiza o perímetro das subdivisões internas para uma dada área. Assim as abelhas otimizam o uso de cera.

feromônios são representados pela distância (qualidade da solução avaliada pela função objetivo) até o ponto ótimo através de um grafo ou árvore de possibilidades. Este design é bastante utilizado para solucionar o problema do "caixeiro viajante" ou para otimizar mecanismos de roteirização.

Algoritmo Genético A inspiração deste vem do processo de seleção natural. Às melhores soluções para o problema é dado o direito de gerarem descendentes para a próxima iteração. Perturbações aleatórias simulam os efeitos da mutação genética que, se melhorarem o resultado da função objetivo, irão se perpetuar nas próxima gerações.

Particle Swarm Neste método, cada solução individual tem uma velocidade e memória do espaço de busca do problema, bem como conhecimento do melhor resultado encontrado pelo "cardume". Quando as iterações são graficadas, os movimentos dos indivíduos se parece com o movimento de um cardume de peixes, ou de um grupo de pássaros.

Neste trabalho, iremos aprofundar teoricamente sobre estes dois últimos métodos nas seções 2.3.3 e 2.3.4.

2.3.2.4 Teorema NFLT

Dados os objetivos deste trabalho, cabe fazer menção ao teorema NFLT ou *no free lunch teorem*; nas palavras dos autores:

> *"Roughly speaking, we show that for both static and time dependent optimization problems, the average performance of any pair of algorithms across all possible problems is identical. This means in particular that if some algorithm's performance is superior to that of another algorithm over some set of optimization problems, then the reverse must be true over the set of all other optimization problems."*[37]

Poder-se-ia argumentar que diante do teorema NFLT, não faz sentido comparar-se a performance de dois algoritmos, especialmente quanto mais gerais sejam as funções objetivos ou tipos de problemas a que esses algoritimos sejam aplicados. Porém, como observado por Eberhart et al.[38], a condição do teorema – de que a *média de todas as*

[37] [17]
[38] [18, p. 299-302]

funções objetivo são iguais – não é uma condição trivial. Isto significa dizer que uma infinidade de funções objetivo sem qualquer relação com fenômenos reais ou qualquer aplicação útil precisa ser considerada com mesma importância. Ou seja, se se retirassem essas "funções inúteis" o teorema não se manteria verdadeiro, mantendo aberto o caminho para se buscar aperfeiçoar e desenvolver otimizadores cada vez mais eficientes.

2.3.3 Particle Swarm Optimization

2.3.3.1 Modelo básico

Introdução O algoritmo de Particle Swarm Optmization foi introduzido por Kennedy e Eberhart em 1995[39]. O algortimo é inspirado na interação social de indivíduos vivendo em grandes aglomerados como cardumes de peixes ou grupo de aves. O movimento aparentemente sincronizado desses animais quando buscam comida ou fogem de predadores se assemelha ao comportamento das partículas ao longo das iterações deste algoritmo. Cada partícula é uma possível solução para a função objetivo e seu movimento ao longo do espaço de busca é definido pelos seguintes princípios, que sintetizam a mecânica desse otimizador[40]:

- **Avaliação** Cada indivíduo avalia sua situação atual através do resultado da função objetivo.
- **Comparação** Cada indivíduo compara o seu resultado atual com a sua melhor avaliação individual, ou seja, as partículas detém memória.
- **Mimetização** Cada indivíduo imita o indivíduo que obteve o melhor desempenho de todos, convergindo para a melhor solução encontrada pelo grupo.

Essas características estão refletidas no movimento de cada partícula, a que se denomina *velocidade*. A velocidade de cada partícula é atualizada a cada iteração, na forma descrita pela equação 2.17.

[39] [19]
[40] [18]

Formulação Na formulação mais básica deste otimizador o vetor *velocidade* $v_{i,j}$ de uma partícula j na iteração i é dada por:

$$\overrightarrow{v}_{i,j} = \omega \cdot \overrightarrow{v}_{i-1,j} + c_1 \cdot \overrightarrow{r}_{1\,i,j} \odot (\overrightarrow{p}_{i-1} - \overrightarrow{x}_{i-1,j}) + c_2 \cdot \overrightarrow{r}_{2\,i,j} \odot (\overrightarrow{g}_{i-1} - \overrightarrow{x}_{i-1,j}) \quad (2.17)$$

Onde:

ω é a constante inercial.

c_1 e c_2 são os coeficientes cognitivo e social, respectivamente.

$r_{1\,i,j}$ e $r_{2\,i,j}$ são vetores estocásticos extraídos de uma distribuição uniforme com intervalo $[0, 1]$.

$x_{i-1,j}$ é a posição da partícula j na iteração $i-1$, ou seja, o vetor que comporta os valores para uma dada solução da função objetivo.

$p_{i-1,j}$ é o vetor que representa a melhor solução encontrada por esta partícula, também chamado de p_{best}.

g_{i-1} é o vetor que representa a melhor solução encontrada por todo o grupo de indivíduos, também chamado de g_{best}.

Note que os elementos dessa soma vetorial representam as três características apresentadas acima; bem como os três padrões de comportamento que marcam o funcionamento do PSO:

- $\omega \cdot \overrightarrow{v}_{i-1,j}$ representa o elemento inercial que, quando assume valores grandes, favorece a exploração de novas regiões do espaço de solução, ou *exploration*.
- $c_1 \cdot \overrightarrow{r}_{1\,i,j} \odot (\overrightarrow{p}_{i-1} - \overrightarrow{x}_{i-1,j})$ representa o elemento cognitivo ou a memória da partícula. Quando este termo é grande, se favorece a exploração de regiões próximas ao p_{best}, ou *exploitation*.
- $c_2 \cdot \overrightarrow{r}_{2\,i,j} \odot (\overrightarrow{g}_{i-1} - \overrightarrow{x}_{i-1,j})$ representa o elemento social das partículas. Quando este termo é grande, se favorece a convergência das partículas em direção ao g_{best}.

A subtração vetorial envolvendo p_{i-1}, g_{i-1} e a posição da partícula em $i-1$ determina a magnitude da velocidade com a qual a partícula se deslocará em direção a estes respectivos pontos. Tão mais distante uma partícula esteja destes pontos, mais rapidamente ela tenderá a voltar a eles. Note que quando $p_{i-1} \simeq g_{i-1} \simeq x_{i-1}$ a partícula terá velocidade

próxima de zero e tenderá a permanecer no mesmo ponto, ou seja, terá convergido para uma solução; exceto se o termo inercial for grande.[41]

Parâmetros Geralmente, nessa versão básica do PSO, os valores dos coeficientes são definidos como c_1, $c_2 \in [1.49, 2]$ e o $\omega \in [0.9, 0.4]$.[42]; já nos modelos adaptativos esses valores são iterativamente calculados, como veremos em 2.3.3.2 e 2.3.3.3.

A partir de análises de convergência elaboradas por Shi and Eberhart in 1998[43] se definiu uma relação entre os três parâmetros largamente utilizada nos modelos de PSO:

$$c_1 = c_2 = \frac{(\omega + 1)^2}{2} \tag{2.18}$$

Com relação ao tamanho da população, costuma-se defini-la a partir do número de dimensões do espaço de procura, de acordo com a relação: $m = 10 + 2\sqrt{2d}$ onde m é o tamanho da população e d o número de dimensões. Experimentalmente[44], demonstrou-se que a performance do PSO é menos sensível ao tamanho da população que outros algoritmos populacionais. Este resultado foi obtido para $m \in [20, 100]$.

2.3.3.2 Modelos Adaptativos - Linear

Na formulação básica do PSO, os três coeficientes são constantes ao longo de toda a iteração. A literatura apresenta inumeráveis modificações e adições à formulação apresentada na seção anterior. Muitas dessas alternativas se diferem por adotar valores dinâmicos para os coeficientes ao longo da iteração; modulando o comportamento do algoritmos entre as fases de *exploration, exploitation* e *convergência*. Neste trabalho abordaremos duas dessas alternativas: a linear e a ESE/ELS.

No modelo linear, cada um dos três coeficientes c_1, c_2 e ω é restrito um intervalo, geralmente pertencente aos $\mathbb{R}^+$ e a cada iteração o valor de um coeficiente qualquer x é calculado segundo uma relação linear na forma:

$$x_i = \begin{cases} \frac{g}{G}(x_{max} - x_{min}) + x_{min}, & \text{para x decrescentes} \\ \frac{g}{G}(x_{min} - x_{max}) + x_{max}, & \text{para x crescentes} \end{cases} \tag{2.19}$$

Sendo que g é o índice da atual iteração, G é o valor máximo de iterações e que $x \in$

[41] Isto exemplifica a importância da correta seleção dos coeficientes. Neste caso um ω excessivamente grande pode impedir que o algoritmo convirja para qualquer solução.
[42] [20, p.4]
[43] [21]
[44] [21, p. 14]

$[x_{min}, x_{max}]$. Vale notar que o modelo exige um parâmetro adicional G que define exatamente o valor de iterações necessárias para o algoritmo concluir sua busca.

A performance do modelo linear depende de uma cuidadosa seleção dos intervalos de cada coeficiente, bem como se eles crescem ou decrescem ao longo do processo de otimização. Usualmente faz-se que c_1 decresça e c_2 cresça; forçando as partículas a inicialmente explorarem diferentes regiões, próximas ao seu p_{best} antes de iniciarem um movimento de convergência. Isto aumenta a capacidade do algoritmo de encontrar o mínimo global, escapando de regiões sub-ótimas. Também é comum fazer ω decrescer, facilitando a convergência.

2.3.3.3 Modelos Adaptativos - ESE/ELS

O modelo ESE/ELS aqui descrito foi baseado no modelo proposto por Zhan, Zhi-Hui et al. em seu artigo de 2010[45]. O objetivo dos autores é construir um modelo de PSO que reduza o problema de convergência prematura a um mínimo local – muito comum no PSO tradicional – e que seja computacionalmente eficiente; dois objetivos que tem relação inversa entre si.

Uma diferença importante entre o modelo linear e o modelo ESE/ELS é que a adaptação nos coeficientes feita pelo primeiro é (i) dependente do tempo e (ii) o estado/fase da otimização é definida exogenamente. No modelo ESE/ELS os coeficientes são (i) automaticamente ajustados de acordo com a fase da otimização e (ii) a fase é definida como uma função do estado interno das partículas num dado instante.

ESE - adaptação dos coeficientes Os coeficientes c_1 c_2 e ω são controlados ao longo da iteração pelo mecanismo ESE. Vamos abordá-lo na mesma sequencia de passos que o algoritmo percorre:

Definição do estado do otimizador O primeiro passo é identificar em qual estado o otimizador se encontra, ou seja, se as partículas estão em regime de *exploration*, *exploitation*, *convergencia* ou *jumping out*.[46] Esta definição é feita calculando-se a

[45] [22]

[46] A fase *jumping out* é uma fase introduzida pelo autor para permitir às partículas escaparem de um mínimo local, inclusive em funções objetivo cujo ponto de mínimo global evolui com o tempo. Esta fase é caracterizada quando a g_best está distante do algomerado de partículas.

distância média de todas as dimensões de cada partícula segundo a fórmula:

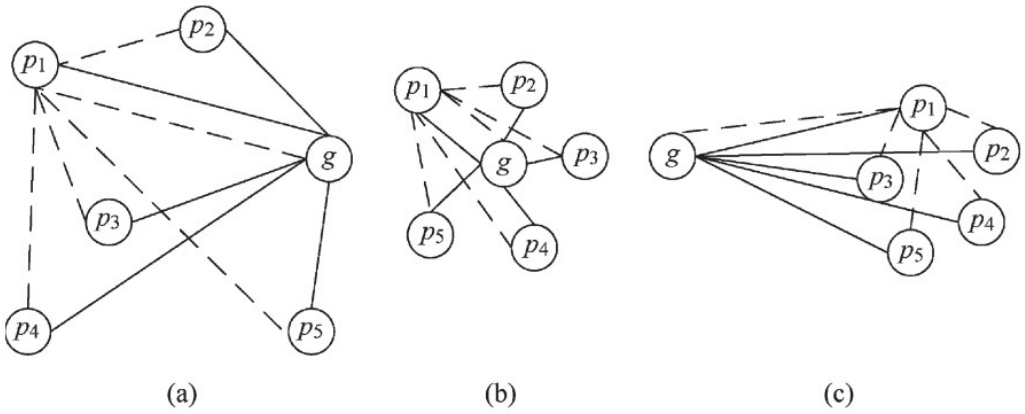
$$d_i = \frac{1}{1-N} \sum_{j=1, j\neq 1}^{N} \sqrt{\sum_{k=1}^{D} (x_{k,i} - x_{k,j})^2} \tag{2.20}$$

Onde N é o número de partículas, D as dimensões do espaço de busca, i e j são índices para duas partículas distintas e d_i é distância média da i-ésima partícula. Denotando-se o d_i da partícula com a melhor solução nesta iteração como d_g, podemos definir o *fator evolucionário* f:

$$f = \frac{d_g - d_{min}}{d_{max} - d_{min}} \tag{2.21}$$

A configuração espacial desses estados pode ser vista esquematicamente na Fig.10 para um χ de $D = 2$.

Figura 10: Posição esquemática das partículas em cada fase



Distribuição das partículas conforme fator evolucionário*: (a)*$d_g \approx d_i d_{pi}$ exploration*, (b)* $d_g << d_{pi}$ *convergência ou* exploitation*, (c)* $d_g >> d_{pi}$ jumping out*. Fonte: [22]*

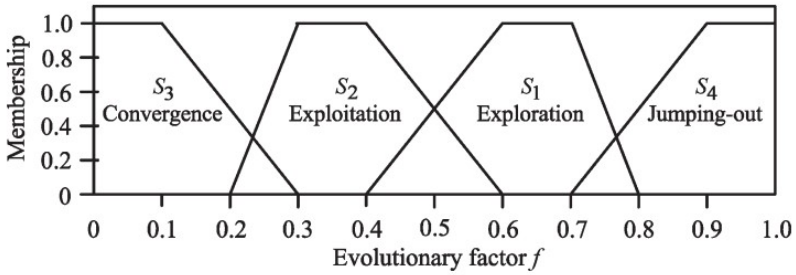
O *fator evolucionário* serve para informar o estado interno em que o otimizador se encontra. A classificação desse estado em fases é feita com uma lógica de classificação *fuzzy*, esquematicamente demonstrada na Fig. 11

Ajustes nos coeficientes Uma vez conhecido a fase na qual o otimizador se encontra, é possível fazer ajustes nos coeficientes de modo a tornar o algoritmo mais eficiente. Para o coeficiente inercial é feito um mapeamento em uma sigmóide que varia monotonicamente com f:

$$\omega(f) = \frac{1}{1 + 1.5e^{-2.6f}} \tag{2.22}$$

Dessa forma, enquanto estivermos na fase de convergência ou *exploitation* o fator inercial será pequeno e não impedirá a busca com passos pequenos por melhorias marginais no

Figura 11: Classificação da fase em função do estado f



Membership *representa a probabilidade de um determinado estado ser escolhido segundo uma lógica* fuzzy *para cada valor de* f*. Fonte: [22]*

resultado. Ao contrário, nas fases de *exploration* e *jumping out*, um ω grande favorecerá o algoritmo escapar de mínimos locais. Na inicialização, $\omega = 0.9$

Os coeficientes social e cognitivo são ajustados com pequenas perturbações δ a cada iteração definidas a partir da fase do otimizador. δ é uma variável aleatória extraída de uma distribuição uniforme cujo intervalo é definido pelo estado do otimizador, conforme a Tabela 1.

Tabela 1: Perturbações dos coeficientes de acordo com a fase

Fase	$\boldsymbol{c_1}$	$\boldsymbol{c_2}$
exploration	$\delta \in [0.05, 0.1]$	$\delta \in [-0.05, -0.1]$
exploitation	$\delta \in [0.01, 0.05]$	$\delta \in [-0.01, -0.05]$
convergência	$\delta \in [0.01, 0.05]$	$\delta \in [-0.01, -0.05]$
jumping out	$\delta \in [-0.05, -0.1]$	$\delta \in [0.05, 0.1]$

Por fim, é imposto aos coeficientes os limites: (i) $c_i \geq 1.5,\ i = 1, 2$ e (ii) $3 \leq c_1 + c_2 \leq 4$. Caso em alguma iteração (i) e/ou (ii) sejam violadas, os coeficientes devem ser normalizados utilizando:

$$c_i = \frac{c_i}{c_1 + c_2},\ i = 1, 2 \tag{2.23}$$

ELS - Aprendizagem elitista Enquanto o mecanismo de ESE atua diretamente nos coeficientes do otimizador, o ELS provê uma forma de simultaneamente melhorar o resultado do g_{best} ou saltar de um mínimo local durante a fase de convergência mediante correção feita diretamente na posição. Esta adaptação é chamada de elitista, pois promove uma perturbação – calculada conforme a equação (2.24) – em uma das dimensões que compõe a posição da melhor partícula. Esta perturbação só será mantida caso o resultado da função objetivo seja menor após a perturbação. Caso contrário, o novo valor $P_{d_{ESE}}$ é utilizado na partícula com a pior performance, aumentando assim as chances dela se

aproximar de uma região mais promissora.

$$P_{d_{ESE}} = P_d + (\chi_{d,max} - \chi_{d,min}) \cdot N(\mu, \sigma^2) \tag{2.24}$$

Na equação do ELS (2.24), d corresponde ao índice da dimensão aleatoriamente selecionada para sofrer a perturbação. P_d é a posição da melhor partícula com respeito à dimensão d, $\chi_{d,max}, \chi_{d,min}$ são os limites do intervalo de valores possíveis para a dimensão d. E o N representa uma variável aleatória extraída de uma distribuição normal com média $\mu = 0$ e desvio padrão σ, chamado de *"taxa de aprendizado elitista"*. Esta taxa é parametrizada linearmente com a evolução das iterações na forma:

$$\sigma = \frac{g}{G}(\sigma_{min} - \sigma_{max}) + \sigma_{min} \tag{2.25}$$

Deste modo, o tamanho da perturbação elitista é grande no início do processo, favorecendo a partícula a escapar de um mínimo local, e menor ao final, favorecendo o refinamento do resultado encontrado pela partícula g_{best}.

2.3.4 Algoritmo Genético

O algoritmo genético (GA), da mesma forma que o PSO, é um otimizador heurístico, estocástico e inspirado na natureza. Porém, ao invés de mimetizar o comportamento social de grupos de animais, o GA busca inspiração no processo de *seleção natural*; onde aos indivíduos mais adaptados é permitido procriar, passando adiante seus genes. Este processo de seleção do indivíduo mais adaptado é o veículo no qual o algoritmo encontra a solução que soluciona o problema de otimização.

Em geral, o GA consiste em uma população – que é um conjunto de soluções do problema – e cada individuo possui um *genoma* – um vetor contendo as variáveis do problema. A primeira etapa do processo – *seleção* – é a avaliação da função objetivo para cada indivíduo. Os indivíduos com as melhores soluções passam para a fase seguinte, chamada de *crossover*, onde os melhores indivíduos tem seus genomas mesclados, dando origem a novos indivíduos. Na sequência, os indivíduos passam por pequenas perturbações nos seus genomas; na fase conhecida por *mutação*. Feito isso, volta-se a avaliar o resultado na nova população, dando continuidade a uma nova iteração. Veremos cada uma dessas fases mais em detalhes abaixo.

2.3.4.1 Genoma e seleção

Como dissemos, um genoma é uma solução possível para o problema de otimização. O genoma contém d genes, que correspondem às d dimensões do espaço de soluções viáveis, χ. Algumas versões do GA fazem um processo de *decodificação* dos genes; convertendo os valores pertencente aos $\mathbb{R}$ para uma sequência binária sobre a qual serão aplicados os processos de *crossover* e *mutação*. Outras pulam esta etapa, trabalhando sempre com valores $\mathbb{R} \in \chi$.[47]

O processo de *seleção* consiste em atribuir a cada indivíduo o resultado obtido pela função objetivo; tomando-se seus genes como valores para a função. É feito um *ranking* dos indivíduos, geralmente com valores padronizados por uma função *mini-max*. Com base nesse ranking, é performada a escolha de uma fração da população que será submetida à próxima etapa. Esta escolha pode ser feita utilizando uma variedade de operadores distintos. Alguns deles são:

Elitista Os indivíduos selecionados são escolhidos ordenadamente, daquele que tem o melhor resultado para o pior; até se chegar na quantidade de "pais" – geralmente igual a metade da população total. Este método tem a característica de favorecer rápida convergência ao custo de se perder diversidade nas soluções, aumentando o risco de ficar preso em um mínimo local.

Competição Dentre a população total, são selecionados aleatoriamente k indivíduos; e dentro deste subgrupo, os indivíduos com melhor fitness são escolhidos para o crossover. Este método, ao contrário do anterior, tem capacidade de manter a diversidade alta, ao custo de uma convergência mais lenta e até eventualmente a perda do melhor indivíduo na iteração.

Roleta Este método atribui uma probabilidade de seleção proporcional ao fitness de cada indivíduo. A "roleta" é construída a partir da probabilidade cumulativa de todos os indivíduos. Por fim, uma variável aleatória é extraída do intervalo [0, 1] que corresponderá ao indivíduo selecionado. Este método pode ser interpretado como um meio termo entre os anteriores em termos da manutenção da diversidade. (Ver Fig.12).

Seleção Estocástica Universal Visto como um aprimoramento da Roleta, pois esse

[47] O emprego do processo de decodificão pode ser profícuo quando as universo das variáveis é ele próprio binário. Nos demais casos a decodificação pode inserir uma complexidade indesejada para a manutenção das soluções dentro do espaço válido de busca. [15, p.150]

método ao invés de retirar aleatoriamente o valor que indica o selecionado[48]; os indivíduos são extraídos por intervalos *igualmente espaçados*, assegurando ainda mais a manutenção da diversidade.

Figura 12: Roleta

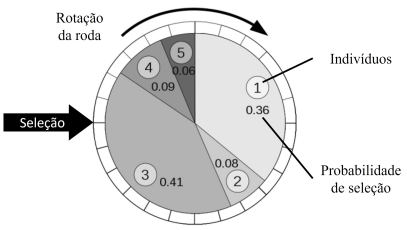
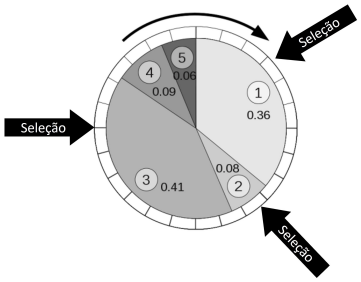


Figura 13: Seleção Estocástica Universal



2.3.4.2 *Crossover*

Da mesma forma que na *seleção* uma variedade de operadores pode ser empregada, no crossover há várias maneiras de se gerar novas combinações de genes a partir dos indivíduos selecionados. Mencionamos duas mais comuns:

Ponto único Imagine o genoma como um vetor com os valores das variáveis em otimização. No método do ponto único, uma variável aleatória indica em qual índice deve ser feito um recorte nesse vetor. Os indivíduos "filhos" terão a primeira parte do vetor originada de uma dos "pais" e a segunda parte do outro "pai".

Múltiplos Pontos Opera da mesma maneira que o método do ponto único, porém ao invés de um único índice que gera dois vetores menores, são feitos vários recortes do vetor original, e cada parte atribuída a um "filho".

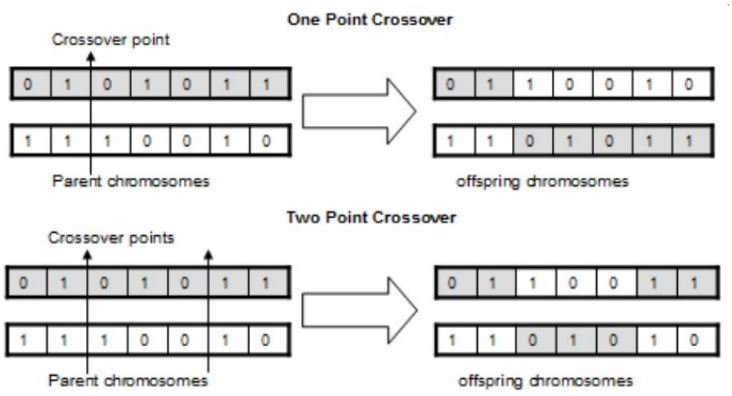
Convém notar que algumas aplicações do GA – especialmente aquelas que fazem o processo de decodificação dos genes descrito acima – adicionam um parâmetro chamado de *taxa de crossover* que consiste em um ensaio de Bernoulli para determinar se o *crossover* deve ocorrer naquele ponto ou não.

2.3.4.3 Mutação

Dentre todos os parâmetros do processo do GA, é à mutação que sua performance é mais sensível[49]. É principalmente a esta etapa que o GA deve sua classificação como um

[48] A seta representada na figura 12
[49] [23]

Figura 14: Operadores de *crossover*. Fonte: [1]



otimizador estocástico. Nesta fase os genes sofrem uma perturbação que ajuda a gerar diversidade de genomas, sem a qual o algoritimo rapidamente convergiria para uma das soluções encontradas pela população inicial. Geralmente a mutação de um dado gene (uma dimensão do espaço de procura de um indivíduo) é dada por:

$$g_{mutated} = B(1,p) \cdot [1 + N(0,1)] \cdot g_{original} \tag{2.26}$$

Onde g é o valor da variável, $N(0,1)$ é uma variável aleatória extraída de uma normal; em alguns casos utiliza-se uma distribuição PERT[50] ou mesmo distribuição uniforme. $B(1,p)$ é uma variável aleatória extraída de uma distribuição binomial, mais especificamente um experimento de Bernoulli. O parâmetro p é chamado de *taxa de mutação*. Após a fase de mutação, a iteração está completa e repete-se o processo; fazendo-se nova avaliação da função objetivo e seleção das melhores soluções.

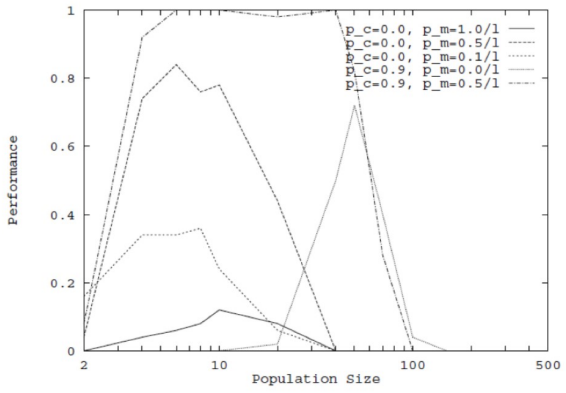
2.3.4.4 População

Mencionamos a importância da taxa de mutação (e da taxa de crossover) por isso cabe também mencionar que o GA também é sensível ao tamanho da população. É evidente que uma população muito pequena tenderá a convergir para um mínimo local dada a pouca diversidade que existe entre suas alternativas de solução, porém uma população muito grande também prejudica a performance do algoritmo; como pode-se ver na Fig.15. Isto se deve ao fato de que boas soluções encontradas por um indivíduo levam muitas iterações para se "espalhar" através de toda a população. Infelizmente, assim como para os demais parâmetros do GA, não existe um tamanho de população ótimo ou ideal[51]. A correta seleção dos parâmetros depende do tipo de problema e também das interações

[50] Foi a opção que utilizamos para ter maior controle sobre a dispersão da mutação. Mas resultados similares à normal podem ser obtidos utilizando-se $\lambda = 4$ na distribuição PERT.

[51] [2, p.29]

Figura 15: Tamanho da população e performance. Fonte: [2]



entre os parâmetros escolhidos.

2.3.5 Tópicos sobre implementação dos algoritmos

Nesta seção abordaremos alguns tópicos sobre a implementação dos algoritmos, que pode ser interpretada como um complemento das seções 2.3.3 e 2.3.4. Estes temas são relevantes tanto para a performance e funcionamento dos algoritmos, quanto para fundamentar alguns dos critérios e testes apresentados na Parte 3.

2.3.5.1 Inicialização

A etapa de inicialização (definição da posição inicial no PSO e dos genes da população inicial no GA) é a etapa individual de maior impacto na performance dos algoritmos. Esta conclusão é intuitiva; pois, em essência, o que um algoritmo faz é "explorar" o espaço de soluções viáveis em busca do mínimo global. Perceba que se todas as partículas iniciarem em um único ponto, a quantidade de informação obtida já na primeira iteração sobre o espaço de busca fica limitada a esse único ponto.

Por esta razão que geralmente a inicialização é feita extraindo-se variáveis aleatórias de uma distribuição uniforme – utilizando séries *pseudo-aleatórias* geradas em computador – com intervalo igual ao espaço de busca. Esta medida já aumenta drasticamente a informação obtida na iteração inicial pelo conjunto de indivíduos.

Algumas abordagens, como a proposta por Jamil, M. et al.[52] buscam melhorar ainda mais a diversidade inicial substituindo a extração de números *pseudo-aleatórios* por séries de números *quasi-aleatórios*, que apresentam baixa *discrepância*. A *discrepância* é defi-

[52] [24]

Figura 16: Distribuição Uniforme (pseudo-aleatória)

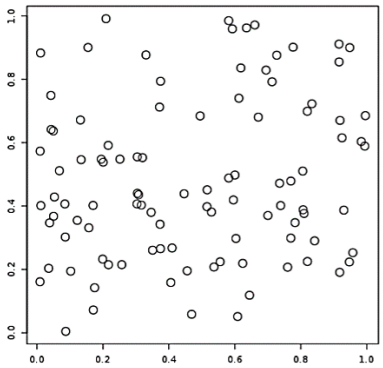
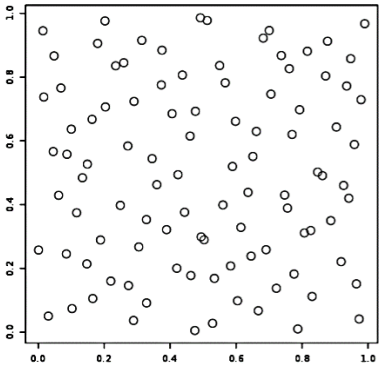


Figura 17: Distribuição da sequência de Sobol (quasi-aleatória)



nida como o desvio que uma variável aleatória tem da distribuição de probabilidade real. Segundo os autores supracitados, mesmo os geradores de series pseudo-aletória mais modernos, tem uma discrepância maior do que os geradores quasi-aleatórios, como aquele que extrai números de uma sequência de Sobol[53]. Na Fig.16 e Fig.17 é possível comparar a inicialização utilizando uma distribuição uniforme vs. Sobol. Fica evidente que a última é capaz de extrair informação de uma área maior, dado o maior "afastamento" inicial das posições; melhorando significativamente o resultado dos algoritmos[54].

2.3.5.2 Restrições no espaço de busca

Na formulação básica de um problema de otimização (2.16) vimos que o vetor com as variáveis deve pertencer a χ para ser considerada uma solução viável. A forma como o algoritmo lida com essas restrições tem enorme impacto na performance do otimizador. Isto porque, dada a natureza estocástica do GA e do PSO, é extremamente provável que uma partícula ou indivíduo se afaste do espaço definido por χ afetando o funcionamento normal do algoritmo[55]

Helwig et al. demonstraram[56] que a probabilidade das partículas aleatoriamente inicializadas estarem muito próximas às regiões limítrofes de χ cresce exponencialmente com o número de dimensões. Isto significa que em problemas com espaço de busca hiperdimen-

[53] A sequência de Sobol é uma sequência de baixa discrepância. De modo simplista, a sequencia de Sobol usa potências de base 2 para gerar partições de um intervalo unitário e posteriormente reordena esses números aleatoriamente.

[54] [24, p.6]

[55] Como veremos ao longo desta seção, o PSO exige mais cuidados com relação ao tratamento das restrições do que o GA. Porque no primeiro, existem vários fatores estocásticos que podem levar uma partícula a explorar regiões não factíveis, dado o seu funcionamento baseado em soma vetorial. Já no GA devemos nos preocupar apenas com a etapa de mutação.

[56] [21, p.56-57]

sional o mecanismo de tratamento das restrições é tão importante quanto os parâmetros fundamentais do otimizador. Existem várias abordagens para lidar com as restrições; aqui abordaremos apenas duas:

Penalidade Os métodos de penalidade – um dos mais comuns – lançam mão do próprio mecanismo de avaliação do resultado da função objetivo para dirigir o algoritmo de volta para região de busca. Esta abordagem consiste em substituir o resultado obtido pela função objetivo para valores inaceitáveis ou mesmo o infinito quando x conter pelo menos uma variável não contida em χ.

Reparo A abordagem de reparo atua diretamente na posição ou na velocidade da partícula (PSO) ou no valor do gene (GA) para "recolocá-lo" dentro do espaço de busca. Citamos aqui alguns dos métodos de reparo aplicáveis ao PSO – esquematizados na Fig.18:

Proximidade A partícula é "reinicializada" exatamente sobre o ponto limite (lb_d para limite inferior e ub_d para limite superior) com relação àquela dimensão d da qual ela evadiu o espaço[57]:

$$x_{i,j,d} = \begin{cases} lb_d, & \text{se } x_{i,j,d} < lb_d \\ ub_d, & \text{se } x_{i,j,d} > ub_d \\ x_{i,j,d} & \forall x \in \chi_d \end{cases} \tag{2.27}$$

Encolhimento Neste método o vetor de velocidade inteiro é reduzido de modo que a partícula "pare" sobre o ponto de limite. Note que neste método o vetor é alterado em todas as dimensões e não só naquela que teve seus limites ultrapassados.

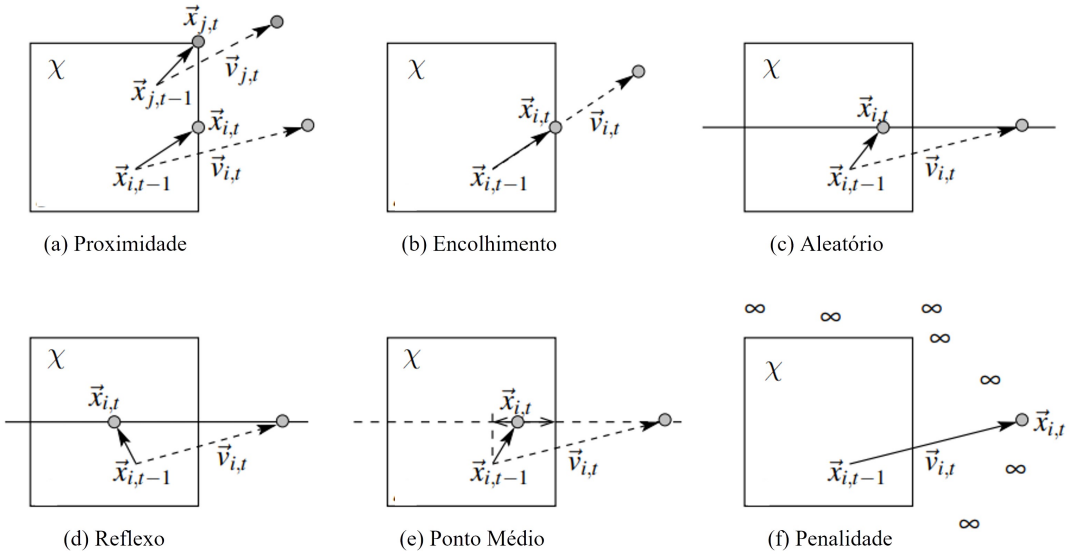
Aleatório Utilizando-se uma distribuição uniforme dentro do intervalo χ, define-se um novo valor para x, desprezando-se o anterior. Este método também é empregado no GA.

Reflexo O vetor de velocidade é recalculado para se obter sua versão refletida com relação ao limite da dimensão evadida.

Ponto médio Neste método é calculado o ponto médio interno entre a posição $x_{i-1,j,d}$ e o limite evadido.

[57] Este método pode ser usado no processo de mutação de GA, sendo inclusive o utilizado por este trabalho na Parte 3.

Figura 18: Métodos de reparo das posições das partículas. Fonte (adaptado): [2]



No caso do PSO, após o reparo na posição, também pode fazer sentido reparar a velocidade da partícula, para que ela não retorne a pontos fora do espaço. Os principais métodos de reparo de velocidade são:

Ajuste A velocidade é ajustada de modo que: $\overrightarrow{v}_{i,j} = \overrightarrow{x}_{i,j} - \overrightarrow{x}_{i-1,j}$.

Zero A velocidade é forçada para zero: $\overrightarrow{v}_{i,j} = [0, ..., 0]$.

Inversão O sentido do vetor velocidade é invertido, forçando a partícula a retornar para regiões mais centrais.

2.3.5.3 Critério de parada

Os critérios de parada dependem da aplicação concreta que se faz do algoritmo e precisam ter significado dentro do contexto do problema. Não obstante, geralmente arbitra-se que um otimizador deve interromper seu processo diante de quaisquer das seguintes condições satisfeitas:

- Quando o número de iterações exceder um valor máximo previsto.
- Quando o resultado encontrado está suficientemente próximo de um critério de qualidade informado pelo problema. Este critério só é possível quando se conhece de antemão o valor que $f(x)$ assume próximo de condições ótimas.

- Quando a média dos resultados da função objetivo $f(x)$ de cada uma das soluções/ indivíduos m não variou mais do que um dado limite[58] δ ao longo de g iterações passadas desde a útlima geração G. Isto indica que o algoritmo encontra-se estacionado em algum mínimo da função, não trazendo melhoras relevantes:

$$\frac{1}{g}\sum_{k=1}^{g+1}\left[\frac{1}{m}\sum_{i=1}^{m} f(x_{i,G-k})\right] \leq \delta \tag{2.28}$$

Chamamos de *critérios de parada* ao invés de *critérios de convergência*, porque como visto na seção 2.3.2.2 algoritmos heurísticos não permitem comprovar que a solução encontrada é ótima; como por exemplo através da análise da matriz Hessiana da função. Por esta razão que os critérios de parada buscam avaliar a *tendência de melhora* do algoritmo. Cabe ainda notar que nos casos onde a primeira condição é atingida; porém a última, ainda não, é indicativo de que o valor máximo de iterações é excessivamente pequeno ou de que o algoritmo está mal desenhado para o problema.

2.3.5.4 Funções de teste

As funções teste ou funções *benchmark* são funções especialmente construídas para testar e comparar sob mesmo critério a performance de algoritmos de otimização. Existe uma enorme variedade de funções teste, que buscam recriar as condições e desafios à otimização que os problemas do mundo real podem oferecer. Assume-se que se um algoritmo performar bem em uma função teste, cujo valor mínimo global é conhecido, pode-se se esperar que ele também tenha bom desempenho nas suas aplicações práticas. Existem algumas classificações usadas frequentemente quando se trata de funções teste. Vejamos algumas delas:

Continuidade Funções contínuas são aquelas em que o limite da função quando ela se aproxima de um dado valor a, tem o mesmo resultado quando esta mesma função é avaliada em a. Geralmente, funções descontínuas apresentam dificuldade adicional para otimizadores.

Modalidade Modalidade diz respeito à presença ou não de mínimos locais. Funções ditas *unimodais* tem um único mínimo, que é também o mínimo global. São consideradas mais fáceis de se resolver e são úteis para se testar a capacidade de refinamento/precisão do algoritmo. Geralmente algoritmos de gradiente descendente costumam ter performance muito superior ao algoritmos populacionais. Funções

[58] Na Parte 3, para os teste realizados, utilizamos $\delta = 10^{-6}$

multimodais são aquelas que apresentam mínimos locais. São consideradas mais difíceis e são úteis para avaliar a capacidade do algoritmo de analisar todo o espaço de busca e discernir a região onde se encontra o mínimo global.

Separabilidade Funções *separáveis* são aquelas em que é possível se otimizar a função objetivo isolando-se cada uma de suas dimensões (2.29). Já nas funções *inseparáveis* há uma *interdependência* entre as dimensões do problema, obrigando-se a avaliar a função como um todo, com as interações entre variáveis. As primeiras são consideradas mais difícieis de se otimizar.

$$\frac{\partial f(\overrightarrow{x})}{\partial x_i} = g(x_i) \cdot h(\overrightarrow{x}) \tag{2.29}$$

Platô Algumas funções multimodais podem apresentar platôs – regiões sub-ótimas onde o gradiente da função é muito próximo ou igual a zero – circundados por regiões com resultados piores[59]. Dado que a região de platô não oferece nenhuma informação útil para qual direção o algoritmo deve se mover, esse tipo de função é desafiadora para alguns algoritmos

Dimensionalidade Quanto maiores forem as dimensões do vetor x mais difícil, e até impossível, para se solucionar o problema; especialmente se a função for não linear[60].

Na Tabela 2 apresentamos as 14 funções que selecionamos para testar os otimizadores na Parte 3 deste trabalho. Escolhemos um conjunto de diversas funções que apresentam as diferentes propriedades descritas acima de modo a construir algoritmos versáteis.

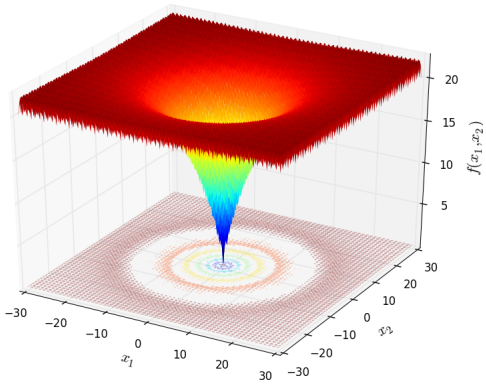
[59] A função f_8 "Matyas", descrita na Tabela 2 apresenta esta propriedade.
[60] [25, p.5]

Tabela 2: Funções teste

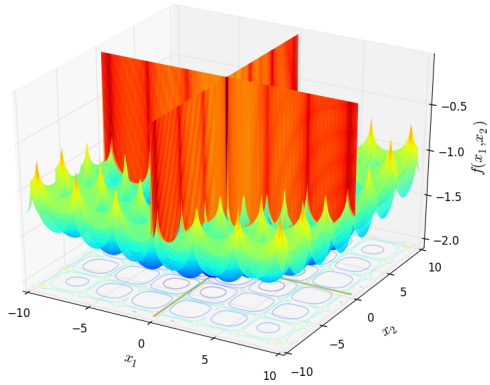
f_n.	Nome	Class.	Fórmula	Domínio	Mínimo
f_1	Ackley	C,M,N,2	$f_x = -20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)} + 20 + e$	$x_i \in [-32, 32]$	$f(0,0) = 0$
f_2	Cross-in-tray	C,M,N,2	$f(x) = -0.0001\left(\left\lvert e^{\left\lvert 100 - \frac{\sqrt{x_1^2+x_2^2}}{\pi}\right\rvert}\sin(x_1)\sin(x_2)\right\rvert + 1\right)^{0.1}$	$x_i \in [-10, 10]$	$f(1.34, -1.34)^* = -2.06$
f_3	Degrau	D,U,S,2	$f(x) = \sum_{i=1}^{n}\left(\lfloor \lvert x_i\rvert \rfloor + 0.5\right)^2$	$x_i \in [-100, 100]$	$f(0,0) = 0$
f_4	Drop-wave	C,M,N,2	$f(x) = -\frac{1+\cos\ 12\sqrt{\sum_{i=1}^{n} x_i^2}}{2+0.5\sum_{i=1}^{n} x_i^2}$	$x_i \in [-5.12, 5.12]$	$f(0,0) = -1$
f_5	Eggholder	C,M,N,2	$f(x) = -x_1 \sin\left(\sqrt{\lvert x_1 - x_2 - 47\rvert}\right) - (x_2 + 47)\sin\left(\sqrt{\left\lvert \frac{1}{2}x_1 + x_2 + 47\right\rvert}\right)$	$x_i \in [-512, 512]$	$f(512, 404.2) \approx -959.64$
f_6	Esférica	C,U,S,2	$f(x) = \sum_{i=1}^{d} x_i^2$	$x_i \in [-5.12, 5.12]$	$f(0,0) = 0$
f_7	Esférica-5D	C,U,S,5	$f(x) = \sum_{i=1}^{d} x_i^2$	$x_i \in [-5.12, 5.12]$	$f(0, ..., 0) = 0$
f_8	Griewank	C,M,N,2	$f_x = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)$	$x_i \in [-600, 600]$	$f(0,0) = 0$
f_9	Holder table	C,M,S,2	$f(x) = -\left\lvert e^{\left\lvert 1 - \frac{\sqrt{x_1^2+x_2^2}}{\pi}\right\rvert}\sin(x_1)\cos(x_2)\right\rvert$	$x_i \in [-10, 10]$	$f(8.05, 9.66)^* \approx -19.2$
f_{10}	Matyas	C,U,N,2	$f(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$	$x_i \in [-10, 10]$	$f(0,0) = 0$
f_{11}	Rastrigin	C,M,S,2	$f(x) = 10n\sum_{i=1}^{n}\left[x_i^2 - 10\cos(2\pi x_i)\right]$	$x_i \in [-5.12, 5.12]$	$f(0,0) = 0$
f_{12}	Rastrigin-5D	C,M,S,5	$f(x) = 10n\sum_{i=1}^{n}\left[x_i^2 - 10\cos(2\pi x_i)\right]$	$x_i \in [-5.12, 5.12]$	$f(0, ..., 0) = 0$
f_{13}	Schaffer#2	C,M,N,2	$f(x) = 0.5 + \frac{\cos^2\left(\sin(x_1^2 - x_2^2)\right) - 0.5}{1 + 0.001(x_1^2 + x_2^2)^2}$	$x_i \in [-100, 100]$	$f(0,0) = 0$
f_{14}	Tripé	D,M,N,2	$f(x) = p(x_2)[1 + p(x_1)] + x_1 + 50p(x_2)[1 - 2p(x_1)] + x_2 + 50[1 - 2p(x_2)]$, `sendo que` $p(x_i) = 1\ \forall x_i \geq 0, p(x_i) = 0\ \forall x_i < 0$	$x_i \in [-100, 100]$	$f(0, -50) = 0$

*Legenda das classificações. Continuidade: Contínua(C), Descontínua(D); Modalidade: Unimodal(U), Multimodal(M); Separabilidade: Separável(S), Não-separável(N); Dimensionalidade: # de dimensões. * Funções que apresentam mais de um ponto mínimo global.* Fonte: [25]

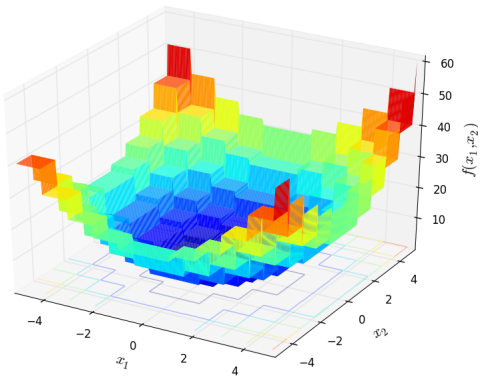
Figura 19: Gráfico das funcões teste para $d = 2$



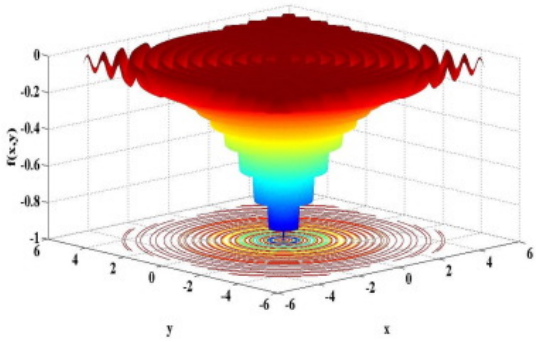
(a) f_1



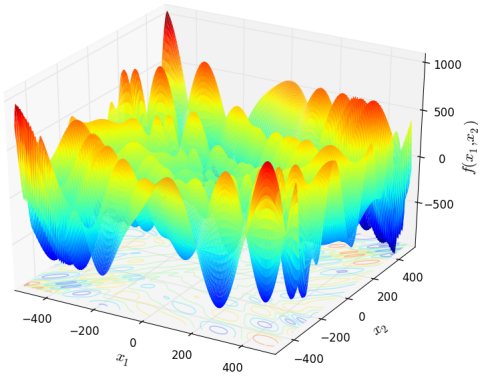
(b) f_2



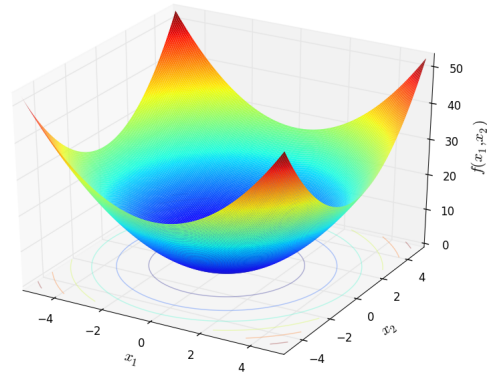
(c) f_3



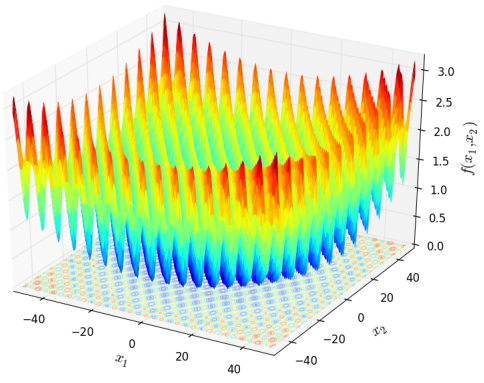
(d) f_4



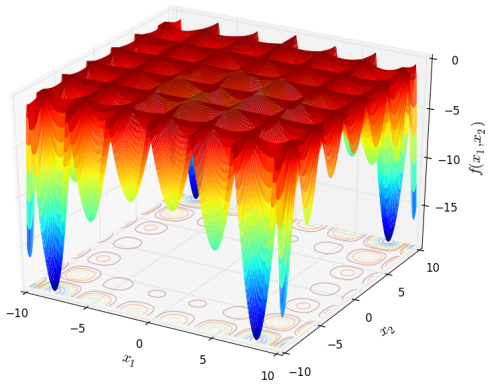
(e) f_5



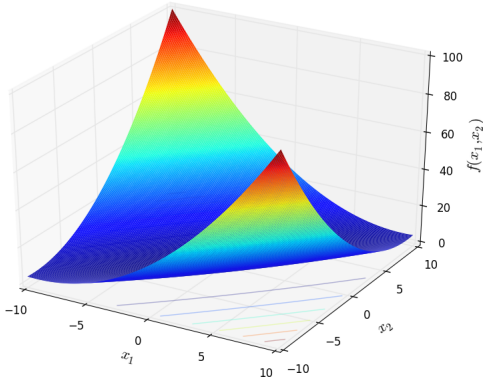
(f) f_6



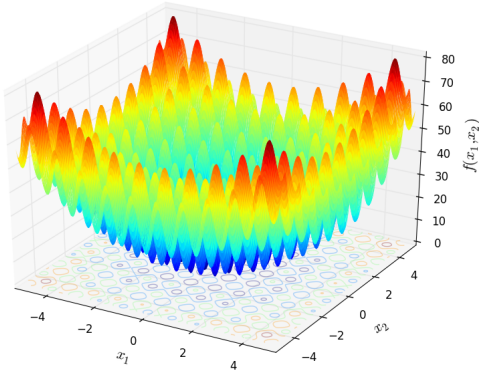
(g) f_8



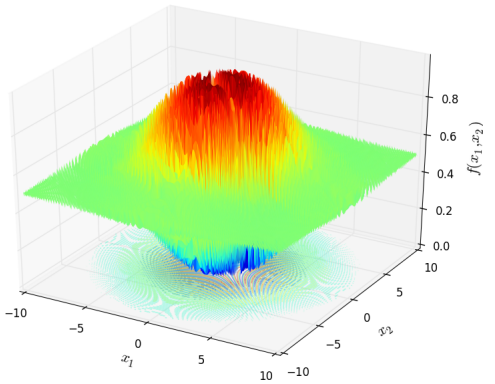
(h) f_9



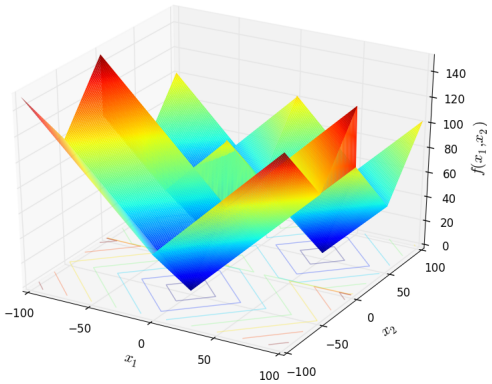
(i) f_{10}



(j) f_{11}



(k) f_{13}



(l) f_{14}

3 RESULTADOS EXPERIMENTAIS

Nesta Parte 3, fazemos a aplicação dos conceitos vistos na Parte 2 em um estudo de caso. Para atingir o objetivo deste trabalho esta Parte está estruturada, em linhas gerais, da seguinte forma: (i) é conduzida análise de *equities* na Bolsa de Valores brasileira - B3 com objetivo de identificar pares de *equities* que apresentem a característica de cointegração; (ii) é construído um modelo de *backtest* para a análise da performance da estratégia de Long&Short dos pares selecionados em (i); e (iii) construção de algoritmos de otimização do tipo GA e PSO e testados contra as funções *benchmark* e por fim (iv) esses algoritmos são aplicados no modelo de *backtest* da estratégia e seus resultados analisados.

3.1 Metodologia

Como se trata apenas de um trabalho de conclusão de curso, sem a pretensão de ser um trabalho científico, tomamos a liberdade de prescindir de um minucioso dissecamento das justificativas de cada uma das decisões envolvendo a modelagem desses experimentos. Não obstante, faremos breves comentários ao longo das próximas seções sobre as decisões mais importantes, dando a devida transparência e permitindo a replicação, ainda que precária, dos experimentos por quem assim eventualmente desejá-lo.

Implementação Toda a implementação foi elaborada pelo autor, utilizando Python 3.8 e algumas bibliotecas populares, entre elas: *matplotlib, numpy, pandas, scipy* e *statsmodels*. No Anexo B estão transcritos os código-fonte das partes mais relevantes. Não apresentaremos pseudo-código das implementações, dado que a intuição pode ser obtida analisando-se o código em Python – linguagem essa que tem como característica a facilidade de leitura. O critério para seleção de quais trechos do código foram incluídos no anexo são: (i) o código representa implementação dos conceitos discutidos na Parte 2; (ii) o código pode ser utilizado sem grandes ajustes, sendo portanto útil ao leitor. Com este critério, obviamos toda a parte de *boiler-plate*, preparação e limpeza de dados, conexão

com banco, geração de gráficos e visualizações, gerenciamento de rotinas e etc. Quando necessário, explicamos no corpo do texto funções e transformações realizadas pelos códigos não transcritos.

Base de dados Os dados utilizados[1] são basicamente dados de mercado das ações ou ETF's negociadas na B3, coletados diretamente dos serviços de *live trading*. Em resumo: (i) preços de fechamento de setembro de 2019 até outubro de 2022 – usados para as análises de cointegração; (ii) preços de negociação *intraday* coletados a cada 60 minutos entre março de 2022 e outubro de 2022 – usados nas análises de cointegração e no *backtesting*; (iii) volumes de negociação diária (do ativo e do seu aluguel) de 2018 até 2022; (iv) taxas médias de aluguel e (v) classificação dos ativos por setores, elaborado pela B3[2] – os três últimos itens, utilizados para seleção de pares. Importante observar que as séries de preço foram ajustadas pelos seus eventos corporativos, tais como: (i) pagamento de dividendos e JCP[3]; (ii) *splits* e *implits* e (iii) bonificações.

3.2 Seleção dos pares

3.2.1 Análise de cointegração

Método A análise de cointegração que implementamos aqui é o método de duas etapas de Engle&Granger, conforme discutimos na seção 2.2.3. No Anexo B.1 estão os principais códigos dessa implementação. Vale adicionar ao que foi discutido, que nesta implementação cada par é testado para cointegração duas vezes. A diferença entre essas duas rodadas é a ordem em que as séries de preço são tomadas como variável dependente e independente na estimação do regressor β. Esta inversão é recomendável, pois os resíduos podem ser diferentes dependendo da ordem das séries, o que afeta a "qualidade" do regressor supramencionado.

Construção dos pares A formação dos pares foi feita mediante a combinação de um conjunto de 224 *equities* – entre elas ações, *units* e índices negociados na B3 – tomados dois a dois. Isto resultou em pouco mais de 24 mil pares submetidos individualmente ao teste de Engle&Granger. Com vistas a analisar a efetividade das restrições setoriais aplicadas

[1] Esses dados de mercado foram coletados, organizados e tratados pela Sole Capital, para uso interno nas suas atividades, e foram gentilmente cedidos para realização deste trabalho.

[2] Listagem disponível em: `https://www.b3.com.br/pt_br/produtos-e-servicos/negociacao/renda-variavel/-empresas-listadas.html`

[3] Juros sobre Capital Próprio.

à formação de pares – abordadas na seção 2.1.2 – classificamos os pares de acordo com a proximidade setorial dos papéis. Ou seja, a B3 classifica o setor de atividade da empresa em três níveis (i) grande setor econômico, (ii) setor econômico e (iii) sub-setor econômico. Nossa classificação dos pares corresponde ao mais alto nível de congruência setorial entre os dois papéis. Aqueles pares que não tem nenhum nível em comum, marcamos como "Nenhum". Ademais, os papéis que pertencem a uma mesma empresa, classificamos[4] como "ON/PN".

Análise em janelas de tempo Cada um dos pares foi testado independentemente para cointegração com suas séries temporais recortadas em seis janelas distintas: (i) 5 anos (1260 dias úteis); (ii) 3 anos (756 dias úteis); (iii) 2 anos (504 dias úteis); (iv) 1 ano (252 dias úteis); (v) 6 meses (126 dias úteis); (vi) um mês (147 horas de negociação) e (vii) uma semana (35 horas de negociação). Todas as janelas partem do dado mais recente t_0 e avançam n períodos em direção ao passado até t_{-n}. Para as cinco primeiras janelas foram usados preços de fechamento diários e para as duas últimas, preços a cada 60 minutos de negociação; assegurando-se dessa forma quantidade de dados razoável para cada teste.

O intuito de se recortar em várias janelas é avaliar a "estabilidade" da relação de cointegração. Por exemplo, um par que apresente várias janelas cointegradas sugere haver uma forte relação; ao passo que outro onde apenas uma janela resultou cointegrada, pode estar passando apenas por uma relação momentânea; especialmente se for nas janelas de mais curto prazo. Em modelos de *trading* mais sofisticados, o uso combinado das janelas cointegradas mais curtas, inseridas dentro de janelas cointegradas mais longas, permite definir vários níveis distintos para execução com diferentes relações de risco retorno para um mesmo par. Em nosso caso, utilizaremos as janelas apenas para avaliar a "força" da cointegração medindo a frequência de janelas cointegradas.

[4]Essa identificação foi feita comparando-se a igualdade dos primeiros quatro dígitos do símbolo de negociação dos papéis.

Ranking dos resultados Dado o grande número de análises que resultam desses pares e de suas várias janelas de tempo, fez-se necessário elaborar um *ranking* dos resultados. Para tal, criamos um critério arbitrário, que chamamos de *coint score*, para medir a "qualidade" de cada cointegração. O critério é calculado conforme equação 3.1.

$$c_{score} = R^2 \cdot f(\beta_{p-value}) \cdot (1 - ADF_{p-value}) \cdot g(ADF_\tau) \ ,$$

$$\text{sendo que a função } \ f(\beta_{p-value}) = \begin{cases} 0, \text{ quando } \beta_{p-value} > \alpha \\ 1, \text{ quando } \beta_{p-value} \leq \alpha \end{cases}$$

$$\text{sendo que a função } \ g(\tau) = \begin{cases} 1, \text{ quando } \tau \leq \tau_{99\%} \\ 0.95, \text{ quando } \tau_{99\%} \leq \tau \leq \tau_{95\%} \\ 0.9, \text{ quando } \tau_{95\%} \leq \tau \leq \tau_{90\%} \\ 0 \text{ quando } \tau \geq \tau_{90\%} \end{cases} \tag{3.1}$$

O intuito é representar a performance do teste de Engle&Granger em seus dois passos. Primeiro, aqueles testes cuja estimação do coeficiente linear β seja estatisticamente significativa e tenha apresentado um R^2 alto[5] terá um *coint score* maior, dado pelos primeiros dois fatores do produtório. Já a segunda etapa do teste é representada pelos dois últimos fatores da expressão 3.1, indicando a confiança acerca da inexistência de raiz unitária na série do *spread*. Dessa forma, quanto mais próximo de 1 for o *coint score* maior a "qualidade" da cointegração do par. Um *coint score* igual a zero pode ser interpretado como uma *proxy* para falha no teste de Engle&Granger.

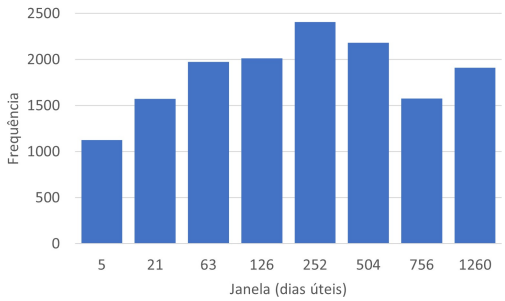
3.2.2 Resumo descritivo da análise de cointegração

Com base nos testes de cointegração realizados, com todas as janelas de tempo, elaboramos algumas estatísticas descritivas dos resultados conforme os gráficos a seguir. De maneira geral, podemos extrair algumas observações:

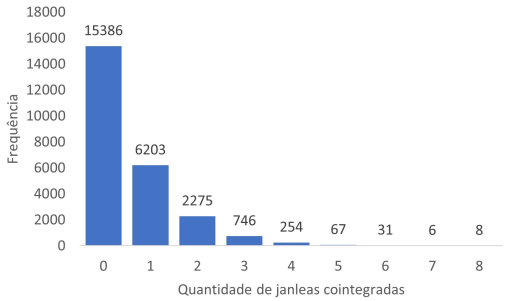
- A frequência de pares cointegrados é baixa: apenas 7.3% dos testes foram capazes de rejeitar a hipótese nula para um α =0.5%.
- Dentre os pares cointegrados, 88% tem apenas uma ou duas janelas; o que pode sugerir uma cointegracão circunstancial, especialmente se forem janelas mais curtas. O número absoluto de pares com mais de cinco janelas cointegradas também chama

[5]Como deve se esperar que o R^2 seja muito alto, dado que se trata de uma regressão de dois processos estocásticos, tal como vimos na seção 2.2.4.

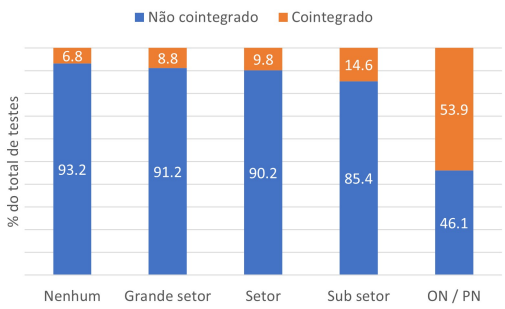
Figura 20: Resumo dos testes de cointegração



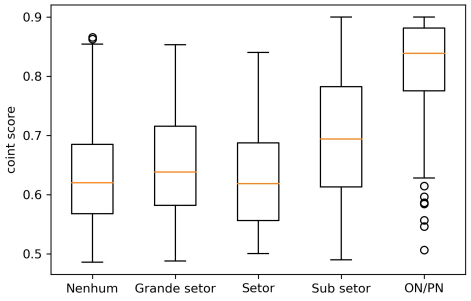
(a) Frequência de cointegração por janelas de tempo



(b) Quantidade de janelas cointegradas do mesmo par



(c) Percentual de cointegração por setor



(d) Coint score dos pares cointegrados por setor

a atenção: 112 pares, ou seja, 0.45% dos pares testados (Fig.20b). Dentre os pares com 7 ou 8 janelas, há uma grande dominância dos ETF's de índices.

- As proporções de cointegrações não variam significativamente em função do tamanho da janela. Quando se compara o valor absoluto de cointegrações, a janela mais curta, de 5 dias, tem uma frequência menor (Fig.20a). No entanto isso poderia ser reflexo de um conjunto de dados menor. Uma afirmação mais contundente a este respeito requereria avaliar estas mesmas janelas com dados de frequência menor – 15 minutos, por exemplo.

- Analisando o dado agrupado por proximidade setorial; pode-se perceber a frequência muito maior (53.9%) de janelas cointegradas dentre pares formados por papéis da mesma empresa (Fig.20c). Por outro lado, também não é desarrazoado argumentar que essa frequência deveria ser muito maior uma vez que se trata da mesma empresa.

- Ainda sobre os dados por setor; pode se notar uma aparente tendência de aumento da frequência de cointegração a medida que o par tenha papéis cada vez mais próximos setorialmente – o que é consonante com a intuição. No entanto, a magnitude com que essa frequência cresce é digna de nota. Poder-se-ia esperar um salto entre os

pares que não tem nenhuma relação setorial com aqueles que tem pelo menos um nível e, no entanto, não é o que acontece. Isto sugere que dentre os fundamentos econômicos para cointegração, há outros fatores relevantes além da proximidade setorial.

- Quanto a qualidade da cointegração das janelas, podemos observar também que nos casos de ON/PN a mediana é claramente superior aos demais agrupamentos, inclusive com certo *skweness* em direção aos *coint score* mais altos (Fig.20d). Já nos demais grupos; a distribuição da qualidade dos pares é indistinta. O mesmo ocorre no agrupamento dos dados por janelas de tempo; ou seja, não há distinção quanto à qualidade em função da duração da janela de tempo.

3.2.3 Pares selecionados para os experimentos

Critérios Para seguir à etapa dos experimentos com os otimizadores; selecionamos 12 pares dentre aqueles criados e testados, conforme seção anterior. A seleção foi feita aplicando-se os seguintes critérios sequencialmente: (i) exclusão de pares com condições desfavoráveis à negociação em situações reais; (ii) ordenamento por *coint score* e seleção dos 30 pares com os maiores valores; (iii) seleção arbitrária de 12 pares, dentre esses 30, de modo a buscar o equilíbrio na representação de exemplares de diferentes setores e também de diferentes classes de proximidade setorial. Os critérios (ii) e (iii) são simples, porém o (i) merece pouco mais de comentários.

Exclusão de pares por critérios de realismo Com vistas a tornar a aplicação experimental, que se seguirá, um pouco mais realista[6] ou próxima do que seria um contexto de negociação real; excluímos da seleção os pares que não atenderam quaisquer dos seguintes critérios:

Liquidez Ambos papéis devem ter no mínimo R$ 250,000.00 de volume negociado diariamente, de modo reduzir o impacto de *price slippage*.

Liquidez de aluguel O volume negociado de contratos de aluguel não pode ser inferior a R$ 100,000.00 diários. Esta restrição, aplicada simultaneamente a ambos os papéis, assegura que o par pode ser montado tanto numa posição "comprada no *spread*" quanto numa posição "vendida no *spread*".

[6] Ver também seção 3.4.5 sobre as limitações da implementação do *backtest*.

Volatilidade Volatilidade do *spread* deve estar no intervalo de 5% a 70% ao ano. Esta medida pode ser interpretada como uma medida de mitigação de risco.

Janelas cointegradas Pares precisam ter, no mínimo, 4 janelas aprovadas no teste de cointegração.

É óbvio que esses parâmetros de "realismo" são totalmente arbitrários e dependem enormemente das condições reais de operacionalização da estratégia. Elementos como o valor financeiro das ordens, custos operacionais, método de envio do lote, entre outros fariam os valores acima variar radicalmente. Buscamos aqui, tão somente, eliminar pares flagrantemente inviáveis.

Tabela 3: Pares Selecionados

Ticker 1	Ticker 2	Prox. Setorial	Coint score	Janelas cointegradas
BOVA11	PIBB11	ETF's	0.886	8
TAEE4	TAEE3	ON/PN	0.665	6
SAPR11	SAPR4	ON/PN	0.660	6
ENGI11	ENGI4	ON/PN	0.646	7
GGBR3	GOAU3	Sub setor	0.629	6
GOAU3	GOAU4	ON/PN	0.607	6
FIND11	BOVB11	ETF's	0.513	5
CMIG4	TAEE11	Sub setor	0.456	4
SANB4	SANB11	ON/PN	0.443	4
PETR3	PETR4	ON/PN	0.433	4
BOVB11	BOVA11	ETF's	0.357	4
SMAC11	SMAL11	ETF's	0.341	4

Resultados de cointegração Na Tabela 3 trazemos os 12 pares selecionados formados a partir de 21 ativos distintos, juntamente da quantidade de janelas cointegradas que esses pares possuem e da média dos seus respectivos *coint scores* nas oito janelas de tempo testadas.

Já na Tabela 4, trazemos os resultados detalhados do teste de cointegração da janela de tempo de 2 anos (504 dias úteis). Nela, os papéis estão identificados como "Var dep" (variável dependente) e "Var indep" (variável independente) denotando a ordem em que foram utilizados na regressão linear. Consequentemente, o β representa o fator – também chamado de *hedge ratio* – que deve ser aplicado sobre o valor financeiro do papel identificado como "variável independente".

Tabela 4: Resultado dos testes de cointegração – Janela de 504 dias úteis

Var dep	Var indep	Coint score	β	R^2	ADF p-value	ADF τ_{crit}
SMAC11	SMAL11	0.899	0.520	0.999	0.000	10%
BOVB11	BOVA11	0.899	1.038	0.999	0.000	10%
TAEE4	TAEE3	0.899	0.991	0.999	0.000	10%
PETR3	PETR4	0.892	1.178	0.997	0.006	10%
SAPR11	SAPR4	0.891	5.500	0.990	0.000	10%
SANB4	SANB11	0.886	0.502	0.984	0.000	10%
BOVA11	PIBB11	0.878	0.522	0.980	0.004	10%
GOAU3	GOAU4	0.798	0.998	0.924	0.040	10%
GGBR3	GOAU3	0.786	1.821	0.901	0.030	10%
FIND11	BOVB11	0.760	1.034	0.854	0.010	10%
CMIG4	TAEE11	0.674	0.236	0.770	0.027	10%
ENGI11	ENGI4	0.658	4.156	0.746	0.020	10%

Por fim, na Fig.21 temos, para cada um dos 12 pares, a série do logaritmo dos retornos diários de ambos papéis na janela de 2 anos. Mostramos também a série da razão entre os dois preços e o *spread*, calculado utilizando-se o β ou *hedge ratio*.

Figura 21: Séries temporais dos pares selecionados

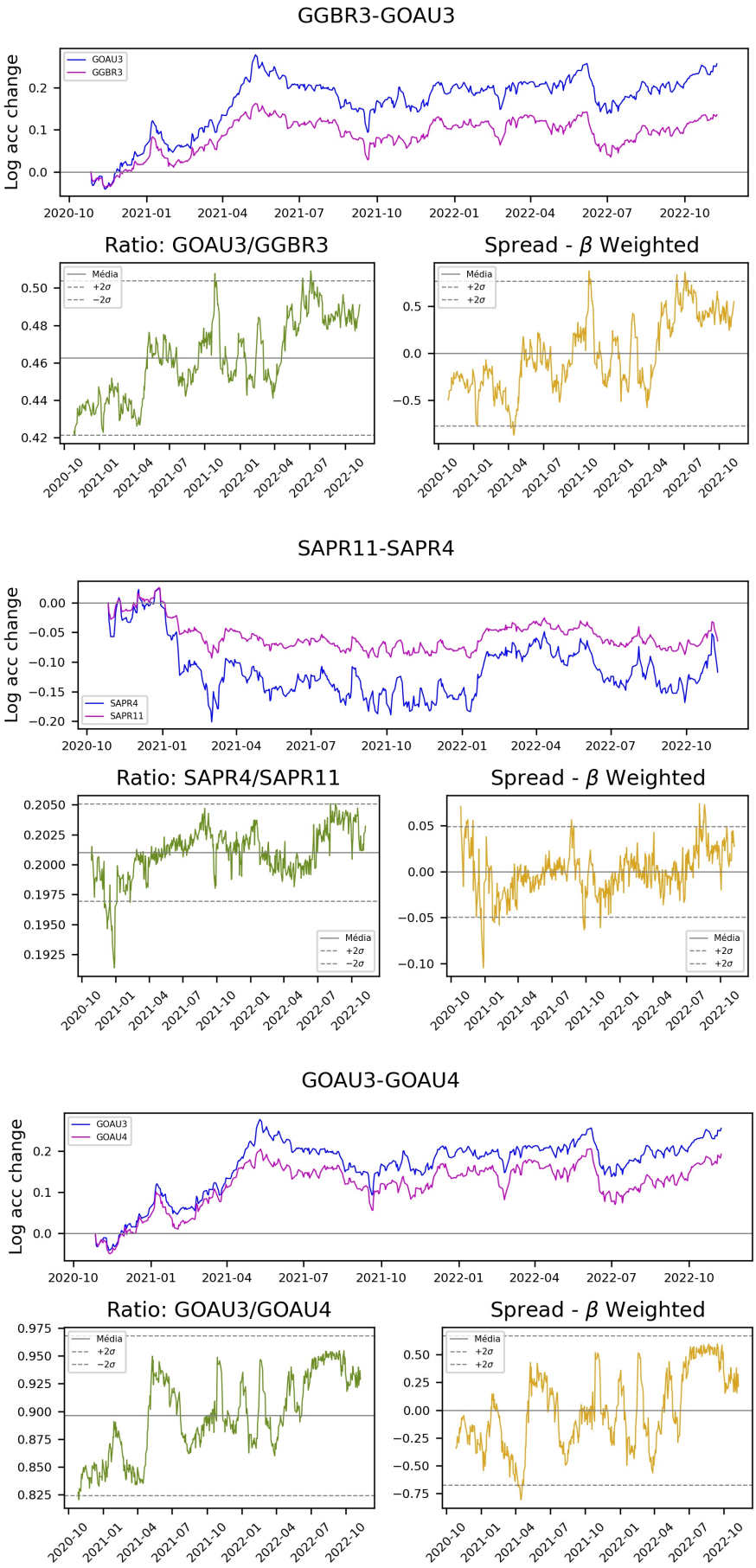


Figura 21: Séries temporais dos pares selecionados

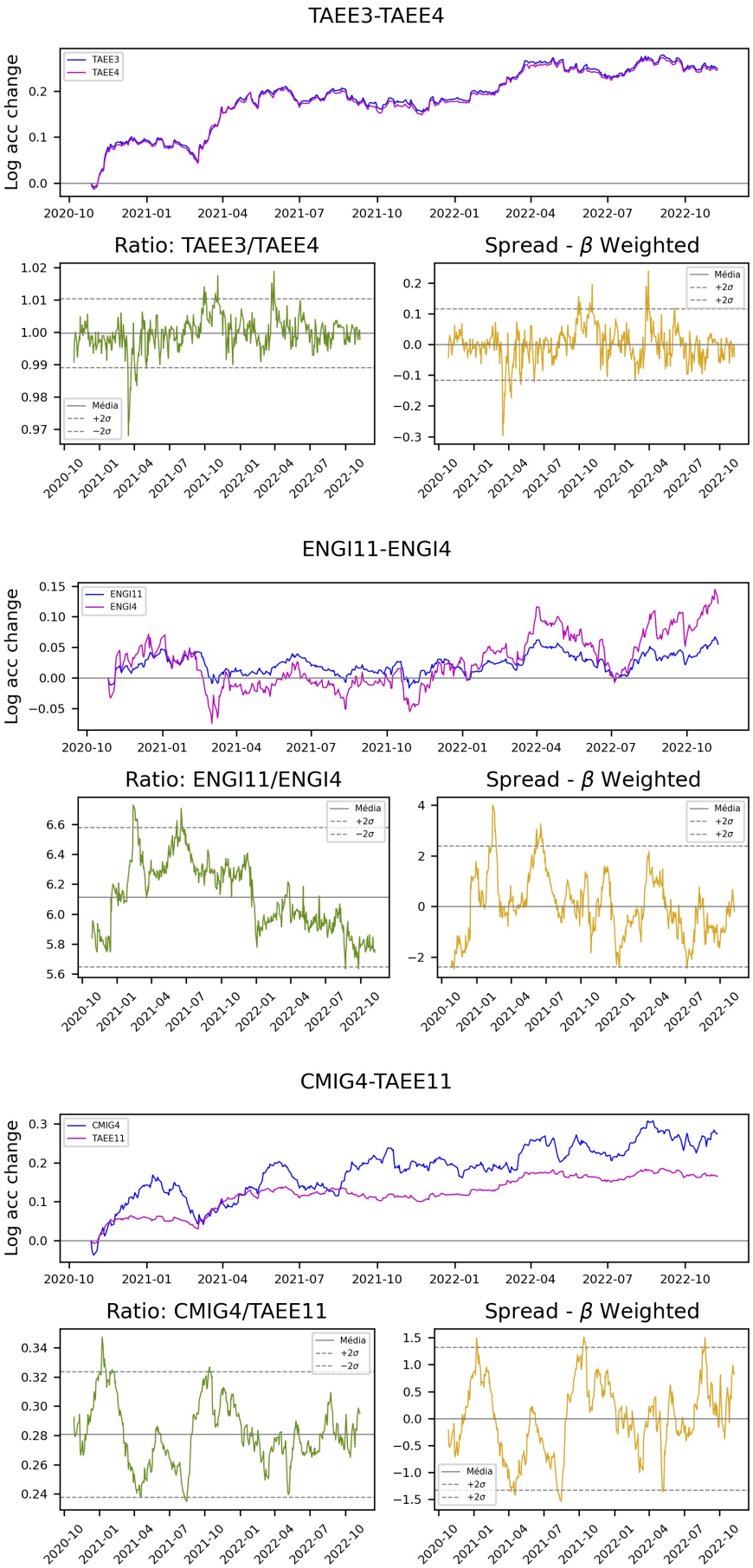


Figura 21: Séries temporais dos pares selecionados

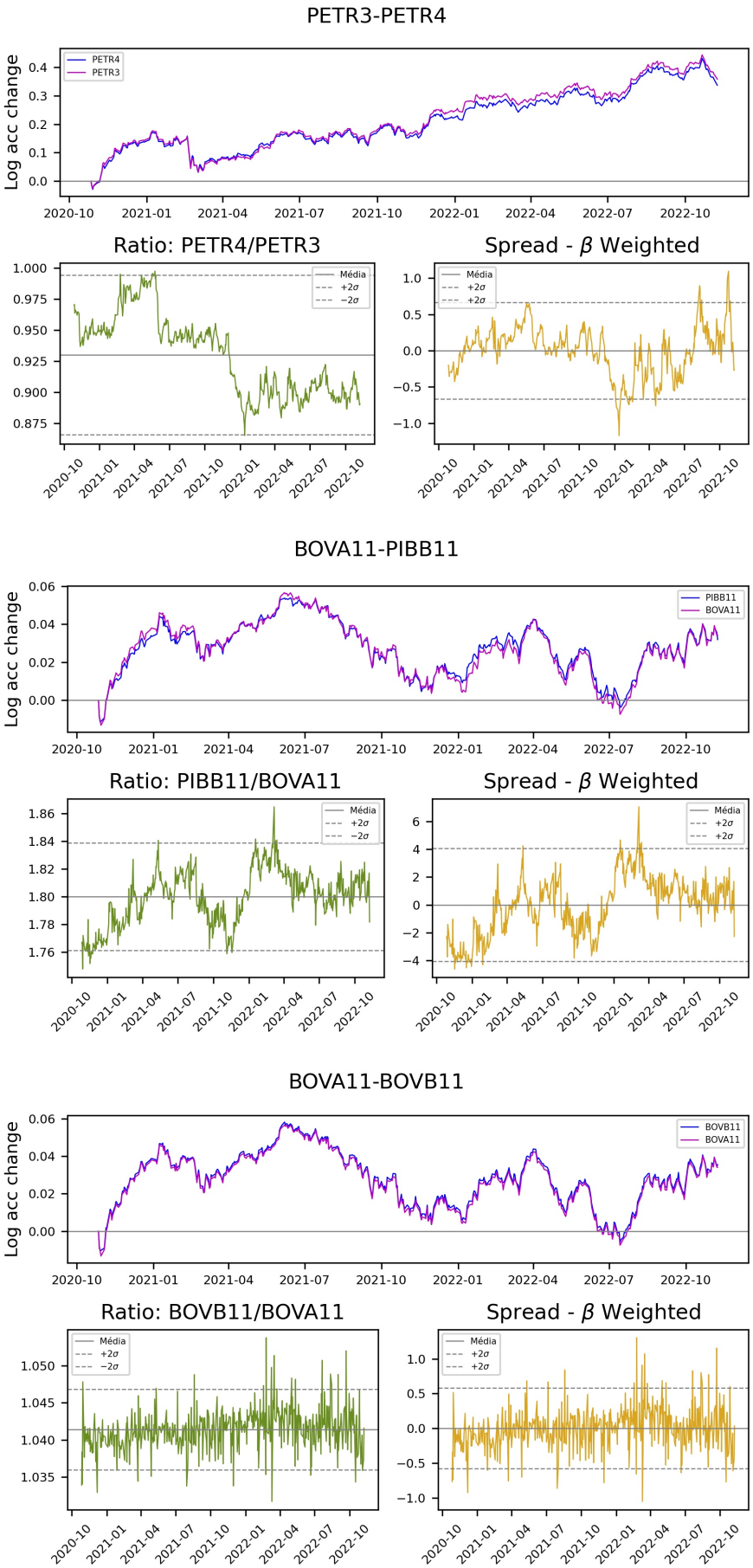
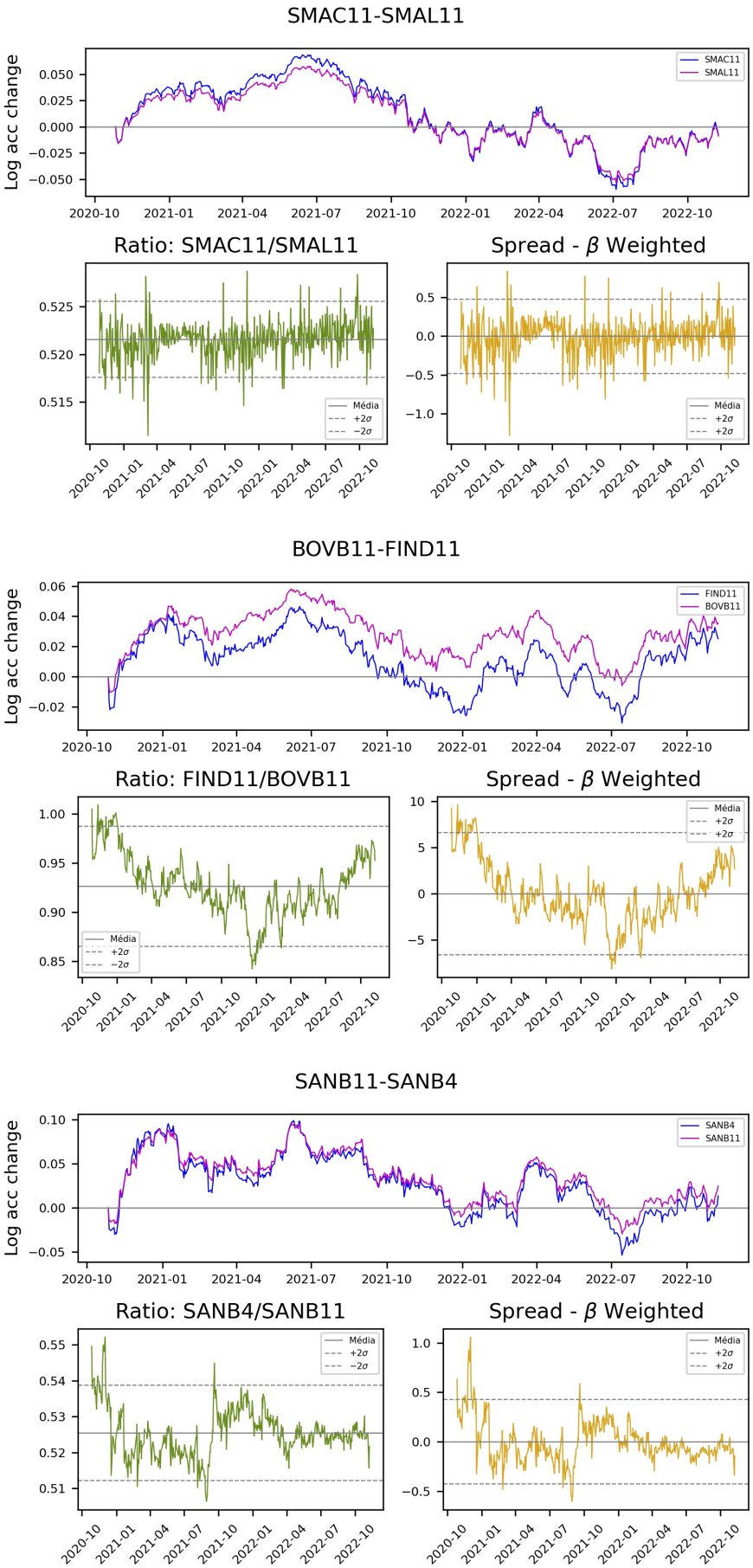


Figura 21: Séries temporais dos pares selecionados



3.3 Otimizadores

3.3.1 Construção e características

Construímos quatro versões diferentes de otimizadores heurísticos. Um baseado no GA e os outros três, no PSO. Descrevemos as principais características de implementação de cada um deles:

GA Algoritmo genético construído conforme seção 2.3.4. A seleção é feita com método de seleção estocástica universal; o *crossover* utiliza método de ponto único aleatoriamente selecionado ao longo do vetor dos genes. Já a mutação é se dá de acordo com a equação 2.26; porém a distribuição utilizada é uma distribuição PERT. Importante assinalar que optamos por aplicar a mutação apenas sobre os descendentes e não sob a totalidade da população[7].

PSO-Simples Modelo de PSO básico, sem nenhum tipo de adaptação dos coeficientes.

PSO-Linear Modificação do PSO onde os coeficientes vão variando linearmente em função da iteração g em que o modelo se encontra, da forma descrita pela equação 2.19. Os intervalos do início até a última iteração $g = G$ foram definidos conforme Tabela 5. Pode-se notar os coeficientes favorecem o comportamento exploratório das partículas no início da otimização. Já mais ao final, a convergência predomina, facilitando o refinamento do resultado.

Tabela 5: Intervalo dos coeficientes no PSO-Linear

Coeficiente	$g = 0$	$g = G$
ω	0.9	0.4
c_1	2.5	1.0
c_2	1.0	2.5

PSO-ESE/ELS Nesta variante, implementamos o algoritmo da forma mais próxima possível ao elaborado por Zhan, Zhi-Hui et al.[8]. O detalhamento do processo de adaptação dos coeficientes (ESE) e do processo de seleção elitista (ELS) foram abordados na seção 2.3.3.3.

[7] O motivo desta escolha não tem relação com a eficiência do algoritmo em si, mas como uma tentativa de reduzir o tempo de processamento; fazendo com que a função objetivo seja chamada para apenas metade da população a cada iteração. Esta escolha pode ter afetado negativamente a precisão do algoritmo genético em comparação com o PSO.

[8] [22]

Com relação à inicialização da população, todos os quatro modelos iniciam com extração da série quasi-aleatória de Sobol[9]. E, no caso do modelos de PSO, o vetor velocidade é inicializado aleatoriamente[10] dentro do espaço de busca.

Com relação ao método para lidar com as restrições do espaço de busca, adotamos o método da *proximidade*. Porque foi o que melhor resultado ofereceu nos testes de desenvolvimento – especialmente na função f_5 – que tem seu mínimo global colocado justamente sobre um dos seus limites. No caso específico dos modelos baseados em PSO, a velocidade da partícula é sempre reiniciada em zero para evitar persistir na evasão do espaço de busca.

Optamos por não impor critérios de parada da otimização. O único parâmetro limitando a evolução do algoritmo é o número máximo de iterações G. Isto é importante para mantermos as condições de comparação semelhantes entre todas as versões do algoritmo e permitir observar sua evolução sem interrupções potencialmente indevidas. Não obstante, monitoramos o custo computacional de cada convergência nos testes a seguir, dando intuição sobre eventuais paradas do algoritmo. Evidente que nas aplicações práticas é desejável a implementação de critérios de parada mais sofisticados[11].

3.3.2 Teste dos otimizadores

Para testar a performance das quatro versões de otimizadores acima, realizamos bateria de testes utilizando como função objetivo cada uma das 14 funções de teste descritas na Tabela 2. Foram rodadas 50 rotinas de otimização independentes, para cada conjunto de parâmetros de otimização. Os resultados apresentados nas tabelas a seguir são a média dos resultados dessas 50 repetições; tanto para o resultado obtido pelo melhor indivíduo de cada repetição (Melhor resultado), quanto pela média de todos indivíduos de cada repetição (Média população). A seguir, discorremos os indicadores apresentados nas tabelas adiante:

Mínimo É a média dos resultados da função objetivo ao final de todas as 50 repetições. O retorno das funções testes foi ajustado para que todas tenham seu mínimo global igual a zero; permitindo assim a comparação direta entre elas.

[9] Vide seção 2.3.5.

[10] Inicialização do vetor velocidade: $\overrightarrow{v}_{0,j} = \frac{1}{2}(\overrightarrow{x}_{random} - \overrightarrow{x}_{0,j})$

[11] Ver comentários finais na seção 4.2

Taxa de sucesso É considerado bem sucedido o teste cujo resultado encontrado pelo algoritmo seja menor que a precisão mínima exigida, arbitrada em 10^{-6}, na média ao longo de 5 iterações[12]. A taxa de sucesso representa quantas vezes foi atingido o mínimo global nas 50 repetições, expresso em percentual.

SFFE Indica a média de quantas vezes a função objetivo precisou ser computada – *Fitness Function Evaluations* (FFE) – para que o algoritmo obtivesse sucesso (S) (com precisão de 10^{-6} na média de no mínimo 5 iterações). É um indicador de custo computacional, especialmente relevante para nossa aplicação que tem uma função objetiva pesada computacionalmente.

TA/kSFFE É índice calculado por $\frac{taxa\ acerto}{SFFE \cdot 10^{-3}}$ e indica uma relação entre a taxa de acertos e o custo computacional associado a esta taxa. Note que só é computado para os testes que encontraram o mínimo global, de modo que o $SFFE \neq 0$. Este indicador pode ser interpretado como um índice de "eficiência" computacional do algoritmo.

3.3.3 Resultados dos testes

Testamos cada um dos algoritmos com os parâmetros sendo a combinação dos seguintes valores: (i) a iteração máxima G foi avaliada com 125, 250, 500 e 1000 iterações e (ii) as populações testadas foram 16, 32, 64, 128 e 256.

A Tabela 6 traz a média dos resultados obtidos em cada uma destas combinações de parâmetros para cada função de teste. E na Tabela 7, todas as funções teste agregadas por otimizador. Analisando esses resultados, podemos pontuar algumas conclusões sobre esta implementação[13] dos algoritmos:

(a) A performance do GA foi substancialmente inferior às três versões do PSO; especialmente nos problemas multimodais não-separáveis.

(b) Especificamente nos dois problemas hiperdimensionais com $d = 5$, a performance do GA contraria o que dissemos no item (a); superando todos as versões do PSO, especialmente na função mais complexa f_{12}. Possivelmente isto se explica pela

[12] Essa condição do mínimo de 5 iterações foi acrescentada porque observamos que o Algortimo Genético por vezes "perde" o melhor indivíduo devido ao mecanismo de Seleção Estocástica Universal, especialmente quando se trata de um gene único na população.

[13] Vale realçar que estas conclusões só tem validade a respeito das implementações desses algoritmos feitas neste trabalho; não sendo possível generalizá-las para as classes de algoritmos.

necessidade de maiores iterações no método PSO, conforme evidência que veremos mais adiante.

(c) O GA apresentou consistentemente[14] a menor dispersão entre o melhor indivíduo e a média da população. Essa melhor convergência pode ser explicada pelo mecanismo de seleção e *crossover*, que faz com que as melhores soluções se "propaguem" entre vários indivíduos da população.

(d) Exceto na f_1 e f_{12} a variante linear do PSO obteve as melhores taxas de acerto dentro das variantes de PSO; conseguindo também o maior taxa de acerto global – 87%. A variante ESE/ELS vem logo em seguida com 84%.

(e) A PSO básico foi superado pelas versões com coeficientes adaptáveis em todas as funçoes teste. As vezes por uma larga margem, caso das funções f_{11} e f_{12}. Isto é uma evidência de que os mecanismos de adaptação trazem significativa melhora à performance dos otimizadores.

(f) Como era esperado, a performance foi excelente nas funções unimodais. Todos os algoritmos alcançaram 100% de acerto, ressalvado a f_7 com o que foi dito no item (b).

(g) Apesar do PSO linear ter tido a melhor taxa de acerto, isso veio a um alto custo computacional – média de 9.9 mil chamadas da função objetivo até o primeiro acerto (SFFE). Neste aspecto, o PSO básico foi o melhor de todos: quando encontrou o mínimo, o fez chamando a função objetivo 2.8 mil vezes em média.

(h) Em termos gerais, pode-se dizer que a PSO-ESE/ELS é a alternativa mais equilibrada; no sentido de que teve a segunda melhor taxa de acertos ao mesmo tempo que registrou a segunda maior eficiência computacional com uma diferença mínima para o primeiro colocado.

[14] A única – e dramática – exceção é a f_2. Uma hipótese para este *oulier* é que a função *cross-in-tray* possui vários mínimos locais próximos do mínimo global tanto em termos do valor do resultado quanto em termos da localização. Isso pode ter feito com que o processo de seleção estocástica universal se tornasse ineficiente ao discernir entre indivíduos com *fitness* muito próximo, povoando continuamente os mínimos locais próximos.

Tabela 6: Resultados dos otimizadores por função teste

			Melhor resultado				Média população			
			Taxa acerto	SFFE	TA/kSFFE	Mínimo	Taxa acerto	SFFE	TA/kSFFE	Mínimo
f_n	**Algoritmo**	**Adaptação**								
f_1	**GA**	**-**	96.0	3939.67	39.59	4.288542e-05	95.0	4179.99	37.66	5.979355e-05
	PSO	**Simples**	100.0	5053.30	41.64	4.127884e-03	100.0	7881.53	30.80	4.127886e-03
		ESE/ELS	100.0	5052.97	38.83	7.685727e-11	81.0	10429.41	19.63	1.066739e-05
		Linear	91.0	16727.63	11.59	7.021830e-07	60.0	34045.74	8.10	1.148339e-03
f_2	**GA**	**-**	84.0	3166.50	26.89	1.450794e-06	1.0	NaN	NaN	5.259661e-03
	PSO	**-**	100.0	1758.44	108.97	-2.037478e-08	92.0	8886.40	40.87	7.029870e-07
		ESE/ELS	100.0	1812.67	101.09	-2.037478e-08	80.0	12054.72	25.49	3.376491e-06
		Linear	100.0	6538.42	29.31	-2.037434e-08	54.0	28462.78	7.51	2.361783e-05
f_3	**GA**	**-**	100.0	606.20	258.49	0.000000e+00	72.0	10038.64	97.85	5.332031e-03
	PSO	**-**	100.0	741.80	240.51	0.000000e+00	100.0	3592.95	77.61	6.250000e-05
		ESE/ELS	100.0	737.36	239.49	0.000000e+00	100.0	5399.41	54.78	2.187500e-05
		Linear	100.0	1152.83	115.36	0.000000e+00	99.0	17322.79	19.07	2.250000e-04
f_4	**GA**	**-**	23.0	4275.93	6.75	4.921436e-02	23.0	4523.63	6.34	7.681788e-02
	PSO	**-**	76.0	2881.20	35.52	1.519911e-02	74.0	8566.28	15.78	1.522994e-02
		ESE/ELS	80.0	3437.77	28.62	1.278358e-02	62.0	11946.42	9.80	1.310132e-02
		Linear	93.0	10139.82	17.33	3.826012e-03	59.0	26714.64	7.48	4.593823e-03

Continua na próxima página

Tabela 6: Resultados dos otimizadores por função teste

			Melhor resultado				Média população			
			Taxa acerto	SFFE	TA/kSFFE	Mínimo	Taxa acerto	SFFE	TA/kSFFE	Mínimo
f_n	**Algoritmo**	**Adaptação**								
f_5	**GA**	**-**	1.0	44625.28	0.11	4.433933e+01	0.0	NaN	NaN	1.451413e+02
	PSO	**Simples**	51.0	3471.25	18.61	5.478762e+01	37.0	12902.22	5.30	5.639322e+01
		ESE/ELS	68.0	3085.88	28.67	2.814813e+01	43.0	15635.36	6.15	3.072690e+01
		Linear	85.0	7795.06	18.53	1.156424e+01	51.0	27039.17	5.79	1.363932e+01
f_6	**GA**	**-**	100.0	1563.32	108.14	3.433170e-12	100.0	1813.00	99.23	3.979908e-12
	PSO	**Simples**	100.0	1807.62	107.48	5.257476e-25	100.0	4090.36	61.31	1.078578e-17
		ESE/ELS	100.0	1764.48	104.65	5.375433e-21	100.0	6263.19	42.16	5.630371e-09
		Linear	100.0	6665.11	29.52	7.841933e-14	96.0	19562.42	15.50	4.230050e-07
f_7	**GA**	**-**	100.0	1572.75	107.24	1.789689e-14	100.0	1820.01	98.45	3.197533e-14
	PSO	**Simples**	87.0	2919.09	42.86	1.317106e-04	87.0	4479.03	34.08	1.317144e-04
		ESE/ELS	89.0	3307.78	35.13	4.529706e-05	89.0	7064.85	22.81	4.673190e-05
		Linear	96.0	11255.31	16.70	1.318413e-06	86.0	20291.52	11.29	3.889881e-06
f_8	**GA**	**-**	7.0	6066.68	9.40	6.634306e-03	7.0	6314.52	8.86	6.809868e-02
	PSO	**Simples**	68.0	3050.85	35.02	2.346740e-03	49.0	13522.39	11.91	2.536779e-03
		ESE/ELS	70.0	3843.46	28.35	2.125001e-03	33.0	20717.26	6.25	2.740402e-03
		Linear	86.0	7325.94	18.57	9.852338e-04	36.0	37300.54	4.04	1.662876e-03

Continua na próxima página

Tabela 6: Resultados dos otimizadores por função teste

			Melhor resultado				Média população			
			Taxa acerto	SFFE	TA/kSFFE	Mínimo	Taxa acerto	SFFE	TA/kSFFE	Mínimo
f_n	**Algoritmo**	**Adaptação**								
f_9	**GA**	**-**	4.0	27422.47	0.35	4.101489e-03	0.0	NaN	NaN	1.032915e+00
	PSO	**Simples**	94.0	2174.98	76.39	1.189009e-01	81.0	10409.29	35.43	1.200630e-01
		ESE/ELS	97.0	2346.95	57.62	1.253712e-02	78.0	10990.16	21.60	1.672865e-02
		Linear	98.0	7047.58	26.72	4.298202e-02	50.0	25532.64	9.93	5.500643e-02
f_{10}	**GA**	**-**	86.0	4618.73	24.63	6.784936e-05	84.0	5216.53	22.45	9.328886e-05
	PSO	**Simples**	100.0	1709.64	113.04	1.153752e-24	100.0	4024.03	62.13	1.239533e-17
		ESE/ELS	100.0	1674.27	109.32	3.743163e-21	100.0	6120.68	43.14	9.155726e-10
		Linear	100.0	6047.94	31.51	1.736327e-14	96.0	19267.66	15.79	5.267797e-07
f_{11}	**GA**	**-**	76.0	2512.92	33.58	2.890039e-01	75.0	2760.48	31.08	5.138457e-01
	PSO	**Simples**	77.0	3040.57	36.21	2.646276e-01	74.0	8203.81	17.21	2.649681e-01
		ESE/ELS	95.0	3743.44	32.79	3.276760e-02	73.0	11923.12	12.03	4.899042e-02
		Linear	98.0	10804.82	18.89	1.689788e-02	62.0	26774.84	8.47	3.097409e-02
f_{12}	**GA**	**-**	75.0	2551.38	33.97	2.931947e-01	75.0	2789.93	31.59	5.241393e-01
	PSO	**Simples**	0.0	7616.00	0.34	5.676193e+00	0.0	13888.00	0.25	5.676867e+00
		ESE/ELS	28.0	30182.46	1.35	1.280727e+00	18.0	35391.28	0.86	1.462307e+00
		Linear	16.0	23671.17	0.82	2.274595e+00	14.0	36143.84	0.72	2.347904e+00

Continua na próxima página

Tabela 6: Resultados dos otimizadores por função teste

			Melhor resultado				Média população			
			Taxa acerto	SFFE	TA/kSFFE	Mínimo	Taxa acerto	SFFE	TA/kSFFE	Mínimo
f_n	**Algoritmo**	**Adaptação**								
f_{13}	**GA**	**-**	44.0	9935.35	4.45	1.324285e-03	44.0	10739.42	4.35	2.778792e-02
	PSO	**Simples**	98.0	1951.08	79.02	8.343604e-05	98.0	6270.92	39.13	9.204035e-05
		ESE/ELS	99.0	2150.42	69.33	8.511566e-06	94.0	9315.91	26.72	4.529518e-05
		Linear	100.0	6332.79	28.12	5.220266e-11	80.0	20889.29	12.34	6.000044e-05
f_{14}	**GA**	**-**	0.0	NaN	NaN	1.139550e+00	0.0	NaN	NaN	3.414779e+00
	PSO	**Simples**	46.0	5621.40	16.16	7.327441e-01	40.0	12252.08	10.04	7.433937e-01
		ESE/ELS	48.0	7519.34	13.48	5.929568e-01	25.0	20403.54	5.74	6.925869e-01
		Linear	55.0	20314.93	5.12	4.225981e-01	17.0	48084.32	2.11	5.787498e-01

Tabela 7: Média dos resultados dos otimizadores

		Melhor resultado				Média população			
		Taxa acerto	SFFE	TA/kSFFE	Mínimo	Taxa acerto	SFFE	TA/kSFFE	Mínimo
Algoritmo	**Adaptação**								
GA	**-**	57.0	5138.91	57.35	3.294462	48.0	4956.76	44.89	10.772172
PSO	**Simples**	79.0	2856.40	72.08	4.400141	74.0	8171.13	33.46	4.515764
	ESE/ELS	84.0	4756.51	64.20	2.148720	70.0	12478.48	22.12	2.354535
	Linear	87.0	9863.39	26.80	1.023295	61.0	26215.59	10.06	1.189977

3.3.4 Análise e seleção dos parâmetros de otimização

Agora, vamos analisar o impacto da escolha dos parâmetros de otimização – população e número de iterações. A Tabela 8 mostra a média das 14 funções teste com a abertura nos parâmetros supracitados. Como era de se esperar, a taxa de acerto cresce com o aumento das iterações e com o aumento populacional de maneira assintótica.

É possível notar que a performance de todos os algoritmos é mais sensível ao tamanho da população do que à quantidade de iterações[15], pelo menos, dentro do intervalo testado. Podemos concluir que nesse intervalo ainda que usássemos os parâmetros máximos (256x1000) nem o modelo PSO básico nem o GA seriam capazes de alcançar 90% de acerto. Com base nessa observação, descartamos esses dois modelos como opções para a aplicação em *pairs trading* que faremos a seguir.

Tabela 8: Taxa de acerto média em todas as funções

(a) GenAlgo

Iterações / População	125	250	500	1000
16	28.1	39.9	47.1	52.1
32	40.4	49.7	56.0	60.1
64	49.4	59.9	62.3	65.4
128	58.7	61.7	64.7	67.9
256	63.7	67.6	70.4	72.6

(b) PSO-Simples

Iterações / População	125	250	500	1000
16	60.9	60.1	65.0	61.9
32	72.9	75.7	75.1	75.9
64	82.0	79.4	80.7	80.7
128	85.3	86.6	87.0	86.3
256	88.6	88.4	89.6	89.3

(c) PSO-ESE/ELS

Iterações / População	125	250	500	1000
16	64.7	71.7	72.3	72.7
32	76.1	81.1	83.0	85.4
64	83.6	86.9	86.6	91.0
128	87.9	89.0	91.9	95.4
256	90.0	92.3	94.7	97.1

(d) PSO-Linear

Iterações / População	125	250	500	1000
16	73.4	79.3	83.0	83.3
32	82.4	87.0	88.3	91.7
64	87.6	88.6	90.4	92.7
128	89.9	90.4	92.7	95.4
256	91.1	93.3	96.0	98.4

Caso o critério para a escolha entre os dois algoritmos restantes fosse puramente a taxa de acerto, claramente o escolhido seria o modelo linear com 256 de população e 1000 iterações, ou até valores maiores que esses. Porém, a aplicação prática requer levar em consideração a eficiência computacional (TA/kSFFE)[16]. Neste critério, o PSO-ESE/ELS

[15] A razão de 1 p.p. de acerto para cada 20 indivíduos ou 128 iterações adicionais. Estimativa obtida por regressão linear com os resultado do PSO-Linear.

[16] O PSO-ESE/ELS é mais eficiente segundo o indicador TA/kSFFE, que mede as chamadas da função objetivo. Porém cabe notar que, devido a maior complexidade do algoritmo, o tempo de processamento

levaria vantagem se o modelo estivesse equipado com um critério de parada; o que não é o caso, conforme discutimos anteriormente.

Diante dessas condições, o critério adotado para escolha foi o algoritmo que tiver a maior taxa de acerto com no máximo 8 mil avaliações da função objetivo[17]. Isto significa escolher a combinação de parâmetros na diagonal superior formada pelas combinações 64x125 e 16x500 que resulte na maior taxa de acerto. Portanto o modelo escolhido é o PSO-linear com população de 64 e 125 iterações. A Tabela 9 traz os resultados com as funções teste obtido especificamente com estes parâmetros de otimização. Vale pontuar que a taxa de acerto desta seleção sobre para 93.3%; se excluídas as funções hiperdimensionais dado que o problema que iremos aplicar não requer tantas dimensões.

real da variante ESE/ELS é maior que o do PSO-Linear, operando sob os mesmos parâmetros. Essa desvantagem se agrava com o aumento dos parâmetros de otimização.

[17] O que significa aproximadamente 9 horas de processamento na máquina utilizada para rodar os algoritmos com a função objetivo do *backtest* para cada par.

Tabela 9: Resultados PSO-Linear – população: 64, iterações: 125

	Melhor resultado				Média população			
	Taxa acerto	SFFE	TA/kSFFE	Mínimo	Taxa acerto	SFFE	TA/kSFFE	Mínimo
f_n								
f_1	100.0	7068.16	14.15	2.067879e-08	0.0	NaN	NaN	8.256683e-04
f_2	100.0	3594.24	27.82	-2.037478e-08	2.0	7872.00	0.25	3.483941e-05
f_3	100.0	935.68	106.87	0.000000e+00	100.0	6210.56	16.10	0.000000e+00
f_4	100.0	5145.60	19.43	5.479386e-12	4.0	7680.00	0.52	6.582672e-04
f_5	84.0	4397.71	19.10	1.040988e+01	8.0	7872.00	1.02	1.260812e+01
f_6	100.0	3613.44	27.67	2.654110e-18	100.0	7089.63	14.11	4.019820e-08
f_7	100.0	5304.32	18.85	1.165285e-12	98.0	7301.45	13.42	1.119235e-07
f_8	74.0	3881.51	19.06	1.791406e-03	0.0	NaN	NaN	2.958754e-03
f_9	100.0	3866.88	25.86	-1.867492e-10	26.0	6935.27	3.75	2.969024e-02
f_{10}	100.0	3488.00	28.67	2.115381e-18	96.0	6905.33	13.90	5.644054e-07
f_{11}	98.0	5216.65	18.79	1.989918e-02	4.0	7872.00	0.51	3.151169e-02
f_{12}	6.0	7061.33	0.85	2.965626e+00	0.0	NaN	NaN	3.042284e+00
f_{13}	100.0	3587.84	27.87	0.000000e+00	82.0	7482.35	10.96	1.167528e-05
f_{14}	64.0	7324.00	8.74	3.647531e-01	0.0	NaN	NaN	6.094070e-01
Média f_n	87.6				37.14			

3.3.5 Comparativo de performance com literatura

Apesar dos resultados obtidos pelo do PSO-Linear nas funções teste aparentarem ser suficientes para o uso desse algoritmo no problema de otimização de *pairs trading*; convém antes compará-lo com os resultados obtidos por referências na área de otimização, com emprego de algoritmos de PSO que representem o estado da arte no campo. Para isto, vamos comparar nossos resultados com aqueles apresentados por Abbas et al.[18].

Abbas et al. desenvolve no artigo referido por este trabalho um modelo de PSO que emprega uma técnica mais avançada de inicialização das partículas. Ele compara esse modelo com outros cinco modelos de PSO; incluindo entre esse alguns que fazem uso de topologia das partículas para ajustar os coeficientes ou que usam estratégias de aprendizagem de máquina em coordenação com o processo de PSO. Na Tabela 10 apresentamos o melhor resultado obtido por Abbas dentre todos os 6 algoritmos na coluna "Mín" e o pior dos 6, na coluna "Min". Na coluna "PSO-Linear" está a média dos resultados obtidos pelo algoritmo desenvolvido para este trabalho em 50 repetições, com os mesmos parâmetros de otimização utilizados por Abbas et al.

Tabela 10: Comparativo com Abbas et al.

f_n*	Dimensões	População	Iterações	PSO-Linear	Mín	Máx
f_1	10	20	40000	3.172600e-04	2.720000e-15	3.850000e-01
f_6	10	20	40000	8.689117e-09	5.550000e-153	9.010000e-10
f_8	10	20	40000	1.904741e-02	2.930000e-03	1.310000e-02
f_{12}	10	20	40000	2.190857e-01	0.000000e+00	9.280000e+00
f_{14}	2	20	40000	3.600000e-01	2.450000e-03	6.020000e-01

** Deve-se atentar que o índice das funções aqui se refere à formulação genérica da função, com as dimensões indicadas acima. No restante do trabalho, utilizamos número de dimensões conforme Tabela 2.*

Das 5 funções teste que foram usadas em simultaneidade por este trabalho e por Abbas et al.; o PSO-Linear entregou em 4 delas resultados dentro do intervalo de performance desses algoritmos mais sofisticados. As duas funções que não conseguimos bater a performance de nenhum das 6 variações de PSO foram f_6 e f_8 por uma diferença de 7.8e-09 e 5.94e-03 respectivamente.

Interessante notar que o PSO-Linear conseguiu performar bem inclusive na f_{14}, que como observou Abbas[19] é uma função especialmente difícil para algortimos que tendem a cair em mínimos locais. Foi justamente a f_{14} que PSO-Linear apresentou seu pior

[18] [26]
[19] [26, p. 289]

resultado entre os problemas de duas dimensões[20].

Outro aspecto importante de observar, é que a performance do PSO-Linear nesses problemas hiperdimensionais, requer um número muito maior de iterações do que estamos utilizando nos testes da seção anterior. Por exemplo a função *Rastrigin* com 5 dimensões (f_{12}) e 125 iterações teve mínimo médio de aproximadamente 3.0. Porém a mesma função com 10 dimensões, muito mais difícil, o resultado foi 0.21; obtido com 40 mil iterações.

3.4 *Backtest*

3.4.1 Função objetivo do problema de *pairs trading*

Até aqui, utilizamos como função objetivo do problema de minimização o conjunto das 14 funções teste apresentadas na seção 2.3.5. Agora que já construímos e testamos o algoritmo que usaremos para otimizar os parâmetros de *trading* da estratégia de pares cointegrados; precisamos apresentar a função objetivo do problema central deste trabalho.

Diferentemente das funções teste, o problema de otimização dos parâmetros de *trading* de pares cointegrados não tem um mínimo global conhecido ou uma formulação analítica. É impossível conhecer o seu comportamento, suas características e eventuais mínimos de forma permanente. Isto acontece, por que o que buscamos é otimizar uma atividade – *trading* – cujo processo fundamental é estocástico e, portanto, seu comportamento só é conhecido no passado.

Isto posto, o melhor que se pode fazer é simular as rotinas de *trading* usando as séries estocásticas do passado para modelar – e otimizar – a rotina de *trading* real. A esse exercício de simulação, chama-se de *backtest*. A estratégia de *trading* simulada nessa sub-rotina de *backtest* foi construída conforme discussão da seção 2.1.3.1.

O *backtest*, porém, não é a função objetivo da otimização, mas sim um algoritmo (sub-rotina) que toma como entrada os parâmetros que estamos buscando otimizar (z, g, p) e retorna as variáveis C de entrada[21] na função objetivo $f(C)$ propriamente dita. A função objetivo f da otimização é o MWRR (3.2) que discutiremos a seguir.

[20] Vide Tabela 9

[21] Ver detalhe na seção 3.4.2.

3.4.2 MWRR

Existem inúmeros candidatos para a função objetivo f. Essa escolha depende do que o usuário almeja maximizar no seu portfólio. A título de exemplo; poder-se-ia querer maximizar o retorno financeiro calculado em moeda ou então maximizar o retorno ajustado pelo risco (volatilidade) medido através do *sharpe ratio* ou ainda a minimizar a correlação dos retornos do portfólio com um dado *benchmark*.

Nossa escolha foi a de buscar maximizar a taxa interna de retorno (TIR) do *trading*, levando em consideração o tempo e o volume financeiro das posições envolvidas. O indicador ideal para isto é a taxa de retorno ponderada pelo dinheiro, ou como é mais conhecida – *Money Weighted Rate of Return*. O MWRR é definido conforme equação (3.2). Por ela, conclui-se que, à taxa de desconto MWRR, o somatório de todos os fluxos de caixa C_n no intervalo de duração do *trade* N, tem valor presente líquido igual a zero. O MWRR é uma função dos fluxos de caixa C obtidos através de *backtesting*, que retorna a taxa de desconto que iguala a zero o somatório dos fluxos à valor presente.

$$\sum_{n=0}^{N} \frac{C_n}{(1+MWRR)^n} = 0 \tag{3.2}$$

O MWRR será a nossa função objetivo f. Simbolizando o algoritmo gerador de fluxos de caixa C (*backtest*) como Ψ e retomando a definição de otimização da equação (2.16) podemos definir o nosso problema na forma da expressão 3.3. Onde z, g, p são os parâmetro de *trading* que queremos otimizar[22] respectivamente: z é o parâmetro para montagem da operação, g é o parâmetro para a reversão e p é o parâmetro para *stop-loss*. Os intervalos de procura, definidos pelas restrições a que está sujeito o problema foram arbitradas de modo a gerar uma espaço de busca grande, porém não permitindo resultado excessivamente distantes das práticas de mercado.

$$\begin{aligned} \min_{z,g,p} \quad & f\left(\Psi(z,g,p,t_0)\right) = -MWRR \\ \text{sujeito a} \quad & \begin{cases} z \in [0.1, 4.0] \\ g, p \in [0.01, 5.0] \end{cases} \end{aligned} \tag{3.3}$$

3.4.3 Conjunto de treino e conjunto de teste

A variável t_0, na expressão (3.3), não faz parte da otimização; mas é uma entrada muito importante da função Ψ (*backtest*). Pois é ela quem representa o intervalo de tempo

[22] Vide seção 2.1.3.1 para maior discussão sobre esses parâmetros.

no qual o *backtest* será calculado; ou seja, t_0 deve representar o intervalo de tempo referente ao conjunto de *treino*. Dessa forma, o algoritmo de otimização tem "conhecimento" apenas de uma parte da série estocástica. É fundamental que os indexadores de tempo t_0 e t_1 das séries estocásticas obedeçam à condição $t_0 \cap t_1 = \emptyset$. Além disso, $t_0 < t_1$, ou seja, o conjunto de treino precisa representar uma porção mais antiga dos dados do que o conjunto de teste. Desta forma, é possível utilizar os parâmetros ótimos retornados pelo otimizador – denotados por * – no conjunto de *teste* t_1 da série e avaliar o resultado que seria obtido com $f\left(\Psi(z^*, g^*, p^*, t_1)\right)$, sem incorrer no viés de *look-ahead*. Em nossa implementação, t_0 se refere ao período entre 1º de abril e 15 de julho de 2022 e t_1 está entre 16 de julho e 26 de outubro também de 2022.

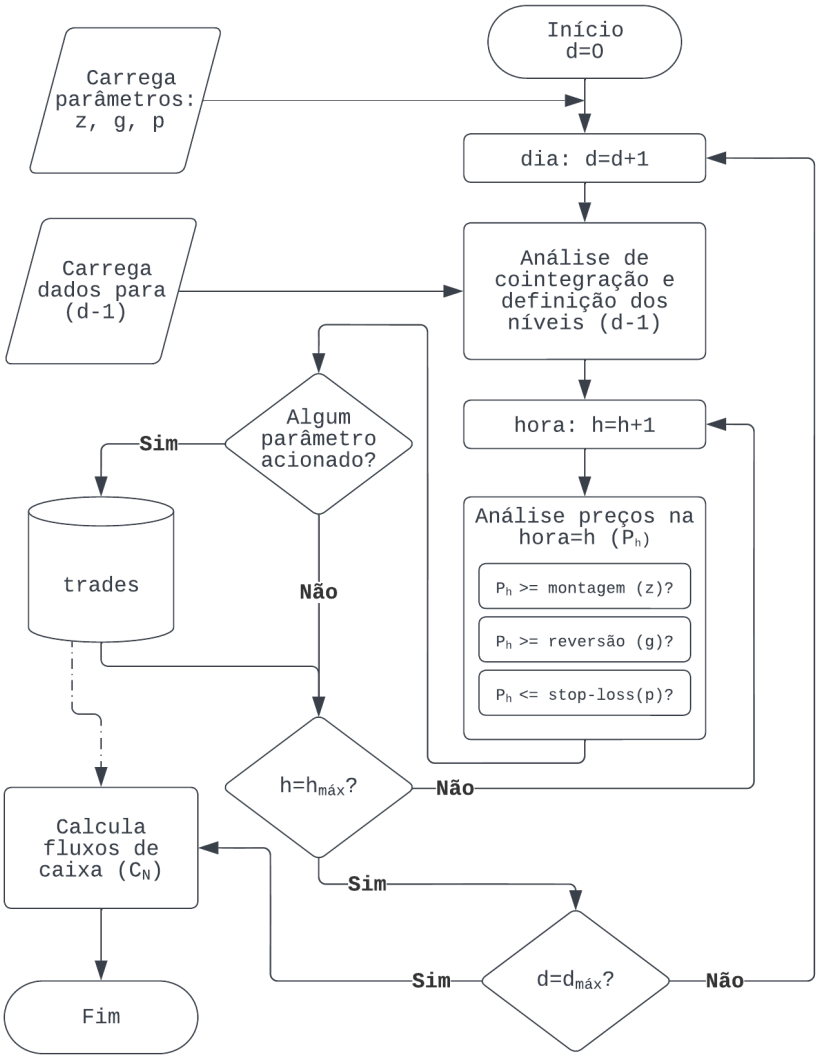
Comparação dos resultados Idealmente, o resultado obtido no conjunto de teste deve ter mesma ordem de grandeza e mesmas características no conjunto de treino. Evidentemente que, em se tratando de processos estocásticos, flutuações irão ocorrer. Definir em que magnitude essas flutuações podem ser consideradas normais é uma decisão que pertence mais à gestão de (e apetite ao) risco do que ao modelo de otimização ou a modelagem do *backtest*.

3.4.4 Diagrama do funcionamento

A Figura 22 mostra o fluxo simplificado do *backtest* implementado. A análise do fluxo já se basta, mas queremos destacar três pontos. Primeiro, que a cada dia (d) uma nova avaliação de cointegração é feita, calculando-se os indicadores utilizados na análise de acionamento dos parâmetros, como por exemplo desvio padrão e volatilidade da série. Esse cálculo é feito com os dados disponíveis até d-1. Feito isso, procede-se à análise dos preços (*ratio*) propriamente ditas com os dados em frequência horária. Na próxima iteração, os dados da análise avançam um dia útil, assim como o dia de simulação também avança mais um dia. Esse método de ir se "movendo" em direção ao futuro com janelas de tempo sincronizadas é chamado de *walk-foward*.

Em segundo lugar, destacamos que o carregamento dos parâmetros de *trading* é feito no início do *backtest* e permanecem constantes ao longo de todo *backtest*. Seria possível – e provavelmente proveitoso – repetir a otimização usando d-1, em paralalelo com a análise de cointegração. Não o fizemos devido ao alto custo computacional que traria. Durante o processo de otimização, cada indivíduo do PSO-linear carrega uma combinação de parâmetros e roda um *backtest*. Isto é feito para toda a população em cada iteração do

Figura 22: Fluxo simplificado do algoritmo de *backtest*



otimizador. Já na fase de testes (onde $t = t_1$), os parâmetros já otimizados são carregados e o algoritmo de *backtest* roda apenas uma vez.

Por fim, para evitar dúvidas; os índices d representados na Figura correspondem aos dias dentro do intervalo definido por t. Por exemplo, para o conjunto de treino, $d \in t_0$; já para o conjunto de testes $d \in t_1$. E d_{max} corresponde ao último dia do conjunto t.

3.4.5 Limitações e problemas do *backtest*

3.4.5.1 Vieses

Convém brevemente mencionar os problemas e dificuldades que existem quando se usa modelos de *backtest* para *ex-ante* a performance de uma dada estratégia de *trading*.

À parte da obviedade de que não há garantias de que o que ocorreu no passado voltará a acontecer no futuro; o emprego do *backtest* reside na expectativa de que os processos de formação dos preços futuros irão refletir os fundamentos econômicos da estratégia – como vimos na seção 2.1.2, consequentemente, certos padrões, devem se reproduzir. Vários cuidados devem ser tomados para evitar algumas armadilhas no processo de identificação desses padrões. Listamos abaixo alguns dos vieses mais comuns juntamente das medidas mitigatórias eventualmente adotadas por nossa implementação:

Sobreajuste Esse é sem dúvida o viés mais perigoso do procedimento de otimização. Uma das principais causas de sobreajuste é a grande quantidade de parâmetros otimizáveis no modelo. Por isto, limitamos à apenas três dimensões, os três parâmetros mais essenciais. Além disso, separamos a base de dados em dois conjuntos, conforme discorremos na seção 3.4.3 com vistas à identificar eventual sobreajuste.

Look-ahead Esse viés ocorre sempre e quando a simulação faz uso de uma informação que é conhecida hoje, mas que estaria indisponível nesse mesmo instante em uma situação real. Eliminamos este problema do modelo ao construí-lo orientado como *walk-foward*, tanto no caso da (i) análise de cointegração, construída sempre com um dia de atraso; (ii) quanto no caso dos parâmetros otimizados, fixados no conjunto de treino e mantido constantes ao longo do *backtest* de teste; e (iii) no tocante à seleção dos pares em si, nos baseamos majoritariamente nos resultados dos testes de cointegração, conforme descrevemos na seção 3.2.3 e não na performance desses pares no *backtest*.

Viés do sobrevivente Consiste em desconsiderar da base de dados aqueles ativos que não estão mais em negociação; seja porque as empresas faliram, foram adquiridas ou fecharam capital. Em nosso caso, não tomamos nenhuma medida de mitigação específica. No entanto, o fato de (i) se processar a análise de cointegração repetidamente antes de se iniciar um trade ao invés de uma única vez e (ii) a curta duração da operação da estratégia *pairs trading*, acabar por minimizar esse viés.

Profundidade do *book* A disponibilidade de volume no *book* de ofertas é um grande desafio para os modelos de *backtest*. Dois erros podem surgir deste problema: ignorar o *price slippage* causado pela operação ou assumir equivocadamente que o valor financeiro da ordem teria sido integralmente executado. A mitigação que adotamos foi no momento da seleção dos pares[23]. No entanto é uma medida tímida e; logo

[23]Vide menção ao *price slippage* na seção 3.2.3.

esse é um viés relevante no nosso modelo e deve ser interpretado como umas das suas limitações.

3.4.5.2 Limitações

Além dos vieses descritos acima a que nosso modelo de *backtest* está sujeito; há também importantes limitações adicionais decorrentes de decisões em sua implementação que aqui as deixamos transparentes:

- Estamos desconsiderando os custos envolvidos nas operações[24]. Embora isso afete os valores absolutos e a frequência das operações descritas aqui; não se trata de uma limitação da otimização em si. Se incluíssemos os custos, o otimizador buscaria igualmente maximizar o retorno sob essas novas condições resultando em parâmetros também ótimos, porém com retornos mais modestos.
- Como a frequência dos preços utilizados é horária; caso uma movimentação de preços condizente com os níveis de montagem, reversão ou de *stop-loss* ocorresse entre dois desses intervalos, ela não seria considerada.
- Não consideramos no retorno da estratégia otimizada o impacto dos custos ou dos retornos associados às garantias financeiras exigidas para a montagem das posições vendidas.
- Não foram considerados os impactos dos tributos incidentes sobre as operações nem sobre os retornos.
- O procedimento de otimização é feito para cada par de forma individual. Portanto, o modelo aqui descrito não otimiza o retorno do portfólio de pares. Relacionado a isso está o fato de que a otimização implementada não influi no dimensionamento dos valores financeiros de cada operação.

3.5 Aplicação do otimizador no *pairs trading*

Agora, finalmente, podemos juntar todas peças construídas ao longo desta Parte 3. Vamos aplicar o PSO-Linear na otimização do retorno calculado pelo MWRR no *backtest* com os 12 pares selecionados por cointegração. Primeiro faremos um resumo esquemático da rotina de otimização. Segundo, mostraremos os resultados da otimização; na sequência

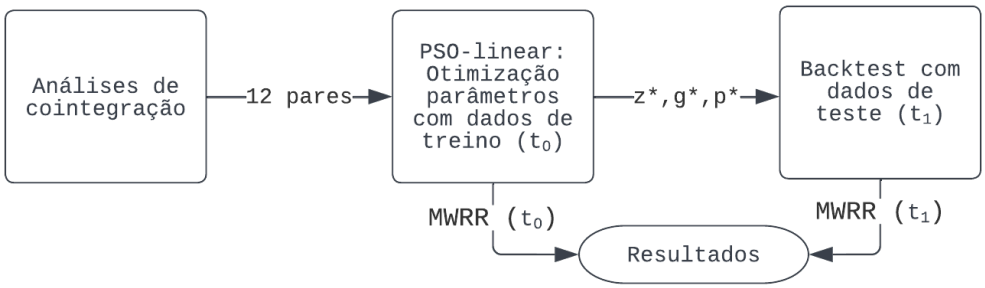
[24] Vide seção 2.1.4 com a listagem dos principais custos.

iremos comparar estes resultados com os *benchmarks* e, ao final, as conclusões obtidas com os experimentos.

3.5.1 Rotina de otimização

Apresenta-se aqui um resumo esquemático de como todas essas "peças" se encaixam nesta rotina de otimização dos parâmetros de *trading*. Começamos com a Fig.23 que mostra a visão global dos experimentos apresentados em toda a Parte 3 deste trabalho. A primeira etapa – da análise de cointegração – foi completamente coberta na seção 3.2. Já as duas últimas são descritas na presente seção e na anterior. A etapa central – de otimização dos parâmetros – exige melhor detalhamento, que pode ser encontrado na Fig.24.

Figura 23: Fluxo geral dos experimentos (Parte 3)

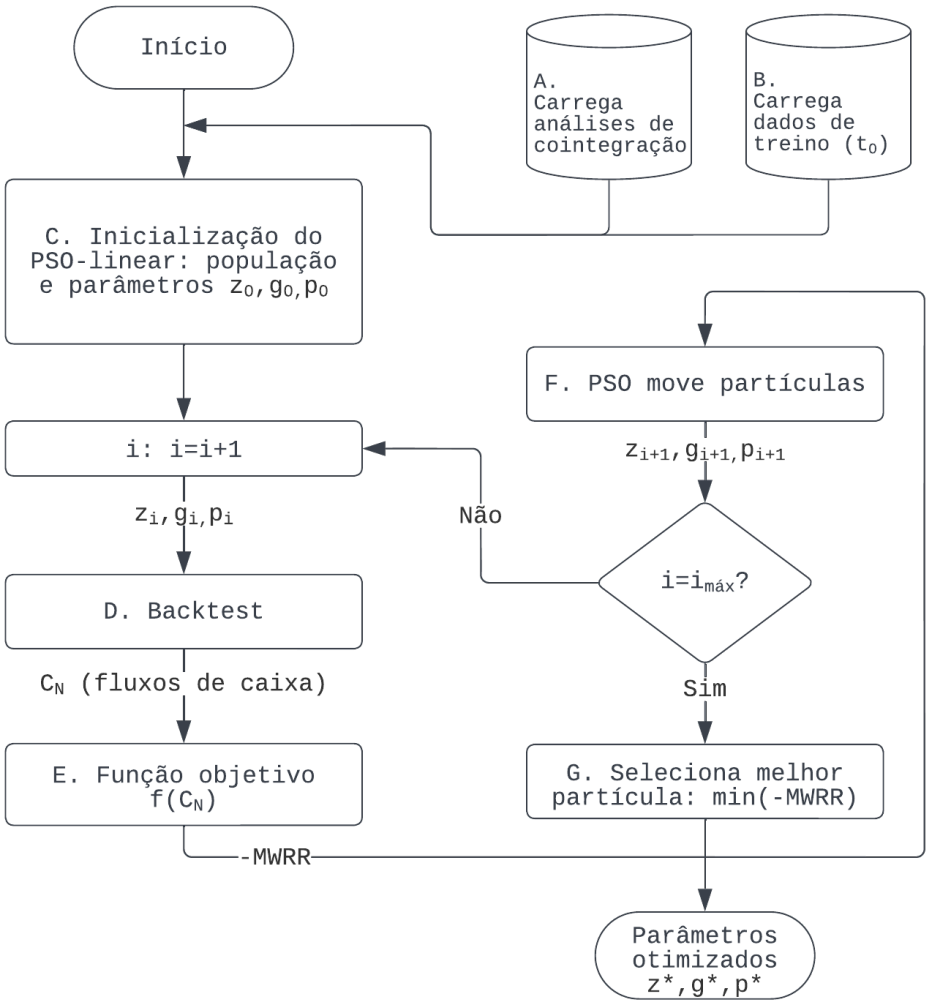


Na Fig.24 a representação dos bancos de dados "A" e "B" indicam a entrada dos dados que foram gerados ou coletados na etapa de análises de cointegração. O processo de otimização propriamente dito começa na etapa "C" onde a população do PSO-Linear é inicializada, conforme já discutimos na seção 3.3.1.

O índice "i" indica o número da iteração do otimizador que, como já vimos, escolhemos $i_{max} = 125$. Já a etapa "D" representa a função Ψ de *backtest*. Note que o algoritmo do *backtest*, detalhado na Fig.22, está representado de forma abstraída pela etapa "D" da Fig.24. Na sequência a função objetivo (3.2) é calculada na etapa "E" a partir dos fluxos de caixa gerados pelo *backtest*.

Já as rotinas do PSO – como por exemplo o cálculo da velocidade, adaptação de coeficientes, nova posição $[z_{i+1}, g_{i+1}, p_{i+1}]$, etc. – estão completamente abstraídas dentro da etapa "F". O resultado de todo este processo são os parâmetros otimizados, descritos na Tabela 11.

Figura 24: Fluxo da otimização dos parâmetros de *pairs trading*



3.5.2 Resultados da otimização

Na Tabela 11 estão os parâmetros de otimização obtidos pelo algoritmo PSO-Linear e o resultado do *backtest* tanto para o conjunto de treino t_0 quanto para o conjunto de teste t_1. A Figura 25 traz gráfico dos resultados no conjunto de teste t_1 obtidos pela carteira dos 12 pares otimizados. Com base nesses dados, elaboramos as seguintes observações.

Conjunto de treino A performance dos pares no conjunto de treino é indício de que o otimizador foi capaz de encontrar as regiões ótimas no espaço de busca. Com uma média de 5.0% de retorno entre os pares e com casos como o de SMAC11-SMALL11 trazendo 10.2%; são resultados bastante positivos. Ainda que não seja possível assegurar-se de que se tratam de mínimos globais; os altos retornos são indício de que o otimizador cumpriu

Tabela 11: Resultado dos pares com parâmetros otimizados

	z^*	g^*	p^*	MWRR t_0	MWRR t_1
BOVA11-BOVB11	1.4332	0.0794	4.1294	0.0750	-0.0084
BOVA11-PIBB11	0.8792	2.4743	0.4000	0.0062	0.0194
BOVB11-FIND11	0.6241	0.1741	0.8617	0.0822	0.0504
CMIG4-TAEE11	2.6942	3.2198	2.9491	0.0000	0.0000
ENGI11-ENGI4	1.2662	2.0921	1.2610	0.0000	0.0318
GGBR3-GOAU3	1.8278	0.0100	1.9611	0.0791	0.0107
GOAU3-GOAU4	1.8591	0.0619	2.7231	0.0886	0.0217
PETR3-PETR4	1.5535	0.4356	2.6655	0.0335	0.0000
SANB11-SANB4	1.8356	2.4650	2.0661	0.0000	0.0000
SAPR11-SAPR4	0.3093	0.0747	3.0434	0.0631	-0.0021
SMAC11-SMAL11	0.6193	0.0356	2.1619	0.1021	0.0713
TAEE3-TAEE4	0.3575	0.1106	0.5710	0.0715	0.0407

seu papel.

Outro indicativo desse bom desempenho no conjunto de treino é a ausência de pares com retornos negativos. Se isso ocorresse, significaria que o otimizador falhou em abster-se de montar posição em um par que não oferece oportunidades de lucro. Exemplo disso são os pares CMIG4-TAEE11 e SANB11-SANB4.

E por fim, após uma repetição da rotina de otimização, não houve variações significativas entre as diferentes rodadas. Diante dessas evidências, concluímos por indução que o PSO-Linear foi capaz de explorar e otimizar satisfatoriamente no conjunto de treino.

Parâmetros de *trading* Os parâmetros otimizados apresentam grande diversidade; com exemplares que, em conjunto, cobrem quase todo o espaço de busca. Houve apenas um caso – g de GGBR3-GOAU3 – onde o parâmetro ótimo está exatamente sobre valor limítrofe imposto pelas restrições.

Alguns casos o z apresentou valores excessivamente baixos, menores que 0.5; o que poderia se reproduzir num excesso de operações em situações reais. É provável que, se incluídos os custos operacionais no *backtest*, o otimizador buscaria regiões mais elevadas para o parâmetro.

Um último ponto que chama a atenção, e que mereceria posterior aprofundamento, é relação risco-retorno formada pelos coeficientes g e p. O que seria intuitivo é que p sempre fosse menor ou igual a g, mantendo relação risco retorno saudável. No entanto, na média, a relação entre g e p é de 0.45; ou seja, para cada unidade de lucro; arrisca-se a pouco mais que duas de prejuízo. Uma hipótese é que, como o otimizador está buscando

maximizar o retorno total, ele acabe por evitar quaisquer situações onde o stop-loss seja acionado; uma vez que isso faria o retorno médio ser sempre menor. É possível que ele esteja "compensando" esse p muito grande com os outros dois parâmetros.

Tomadas em conjunto, as observações acima podem pode ser interpretadas como sugestão de que uma seleção mais cuidadosa e restritiva dos limites do espaço de procura podem tornar a performance da otimização melhor e, eventualmente, dos resultados no conjunto de treino.

Figura 25: Resultados *backtest* com pares otimizados

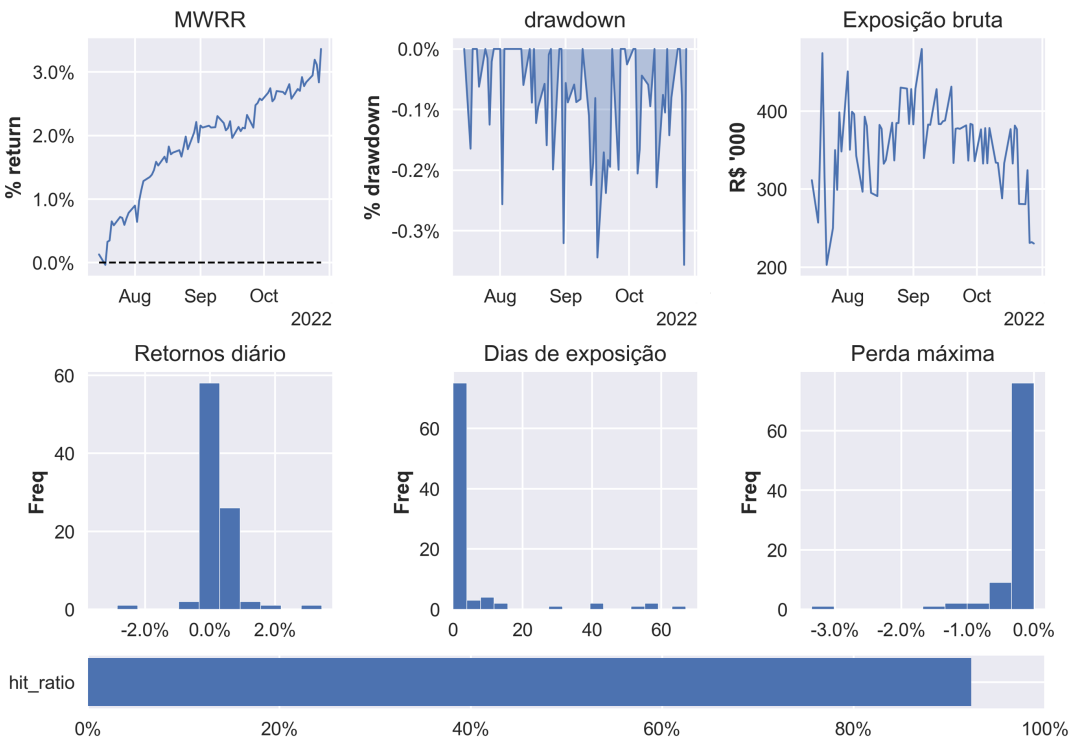


Tabela 12: Resultado no conjunto de testes t_1 da carteira otimizada

MWRR t_1	Vol	Dias de exposição	Hit ratio	Resultado R$
0.03312	0.0257	5.37	94.19%	11120.87

Conjunto de teste e sobreajuste Os resultados obtidos no conjunto de testes, em termos absolutos, não foram ruins. O retorno total da carteira no período foi de 3.3% – equivalentes a 12.45% anualizados, acima do retorno livre de risco do mercado – com uma volatilidade anualizada de 2.57% e taxa de acerto entre os *trades* encerrados de 94.2%. Discutiremos os resultados relativos ao *benchmark* na próxima seção; aqui nos concentraremos na comparação entre os resultados no conjunto de treino e o de teste.

É inescapável a conclusão de que a vários pares sofrem com sobreajuste[25]. Os casos de BOVA11-BOVB11 e SAPR11-SAPR4 são os mais representativos deste problema; apresentando uma dispersão de 8.3 p.p. e 6.5 p.p. entre treino e teste respectivamente. Curiosamente, ENGI11-ENGI4 e BOVA11-PIBB11 contam a história oposta, com o conjunto de teste performando acima do treino; mas numa proporção muito inferior aos dois primeiros. Ao se comparar a carteira otimizada em t_0 com t_1 também se observa uma dispersão relevante de 4.6 p.p.

Sem dúvida que, para se ter mais certeza sobre a gravidade do problema de sobreajuste, seria necessário repetir os experimentos com uma quantidade maior de pares e com janelas de tempo mais longas no *backtest*. Na ausência desses aprofundamentos, prevalece a cautela ao avaliar os resultados apresentados.

Coint score e performance da otimização Embora a intenção da ferramenta que utilizamos para ordenar e selecionar os pares pela "qualidade" da cointegração – o*coint score* – seja apenas isto, uma ferramenta de ordenação; ela apresentou uma capacidade – limitada é verdade, mas não desprezível – de prever a performance do otimizador no conjunto de treino.

Fizemos uma regressão linear entre o MWRR t_0 (dependente) e o *coint score* (independente) dos pares. O resultado foi estatisticamente significante com um p-value de 0.001; estatística F superior ao F-crítico e o um $R^2 = 66.7\%$. O R^2 não é expressivo, mas julgamos de interesse fazer esta observação assim mesmo. Um tópico que poderia ser investigado é estabelecer se o *coint score* ou uma versão aprimorada dele seria capaz de estimar uma relação[26] entre a qualidade da cointegração e o resultado obtido pelo otimizador no conjunto de treino. Isso poderia ser útil na avaliação se um dado otimizador foi capaz de encontrar mínimos eficientemente e apontar casos divergentes onde potencialmente o otimizador não teve boa performance; como por exemplo SANB11-SANB4 – alto em *coint score* porém o MWRR de treino igual a zero.

Fazendo o mesmo exercício para o conjunto de testes, o $R^2 = 38\%$, bem inferior ao que vimos para o conjunto de treino. Sabendo que neste caso está havendo interferência do sobreajuste, pouco pode-se concluir a respeito da relação de *coint score* e performance em t_1.

[25] Vide discussão na seção 3.4.5.

[26] O que provavelmente não seria uma relação linear como fizemos de maneira pouco rigorosa aqui, mas uma dada distribuição das dispersões "aceitáveis".

3.5.3 *Benchmark*

A escolha de um *benchmark* contra o qual se compara uma estratégia de Long&Short é sempre uma escolha bastante arbitrária e, em certa medida, depende dos objetivos de quem faz a comparação. Em nosso caso, queremos analisar se uma carteira com pares cujos parâmetros de *trading* foram otimizados performa melhor que essa mesma carteira com o parâmetros padronizados e idênticos para todos os pares. Para tal, buscamos manter todas as condições idênticas; variando apenas o que estamos interessados em observar – os parâmetros do *trading*.

Tabela 13: MWRR t_1 dos *benchmarks*

	Conservador	Moderado	Agressivo
BOVA11-BOVB11	0.0033	0.0050	-0.0004
BOVA11-PIBB11	0.0000	0.0000	0.0046
BOVB11-FIND11	0.0000	0.0000	0.0155
CMIG4-TAEE11	0.0000	0.0000	-0.0029
ENGI11-ENGI4	0.0000	0.0092	0.0074
GGBR3-GOAU3	0.0000	0.0107	0.0100
GOAU3-GOAU4	0.0000	-0.0005	0.0271
PETR3-PETR4	0.0000	0.0000	0.0204
SANB11-SANB4	0.0000	0.0000	0.0000
SAPR11-SAPR4	0.0000	0.0294	0.0104
SMAC11-SMAL11	0.0181	0.0478	0.0883
TAEE3-TAEE4	0.0000	0.0040	0.0209

Para atingir este objetivo, os *benchmarks* foram construídos utilizando-se os mesmos pares, no mesmo conjunto de teste t_1, com a mesma análise de cointegração informando os níveis do par e através do mesmo algoritmo de *backtest*. Criamos três cenários, que podem ser interpretados como três *benchmarks* distintos. O nível de reversão g e stop-loss p são iguais a 1.0 e são idênticos entre os três cenários; note que eles guardam uma relação risco-retorno de 1:1. O que os difere é o nível do *z-score* em que a posição é *montada*. No cenário "Conservador" a montagem é feita com 2.75 desvios-padrão[27]; no cenário "Moderado", com 2.0 e no "Agressivo", com 1.25. A Tabela 13 mostra o desempenho dos *benchmarks* para cada par e na Tabela 14 os desempenho deles vistos como uma carteira composta pelos 12 pares.

[27]Os desvios disparam a montagem tanto se estiverem acima da média, quanto se estiverem abaixo dela.

Tabela 14: Resultado no conjunto de testes t_1 da carteira *benchmark*

	MWRR t_1	Vol	Dias de exposição	Hit ratio	Resultado R$
Conservador	0.02458	0.0945	0.50	100.00%	734.42
Moderado	0.05146	0.0367	3.54	91.67%	4303.96
Agressivo	0.02964	0.0286	4.43	79.52%	8987.61

***Benchmarks* não considerados** Convém explicar brevemente porque deixamos de lado alguns dos *benchmarks* que costumeiramente se utilizam. Em primeiro lugar, pode-se-ia comparar com o CDI. Porém, essa comparação não é muito útil dado que, numa aplicação real de Long&Short, a garantia depositada junto à B3 pode ser um Título Público, que remunera à taxa Selic. Dessa forma, à performance que reportamos aqui, bastaria adicionar-se o retorno dos ativos em garantia. Dito de outra forma, quando reportamos aqui que o MWRR foi zero, equivale dizer que o retorno foi 100% do CDI.

Outra opção seria um índice de mercado, como por exemplo o iBovespa. Porém, com base no que discorremos na seção 2.1.1 o Long&Short é uma estratégia neutra de mercado e o iBovespa é o próprio mercado; portanto, uma comparação imprópria.

Por fim, poder-se-ia comparar com uma carteira de fundos de investimento que tenham como estratégia principal o *pairs trading*. Neste caso, embora haja a congruência entre as estratégias, não seria possível controlar os demais fatores. Além de que, as limitações do *backtester* poderia dar injusta vantagem ao par otimizado.

3.5.4 Comparação dos resultados com *benchmark*

3.5.4.1 Comparação no nível dos pares

A Tabela 15 mostra a diferença entre o retorno obtido pelos pares otimizados e cada um dos três cenários de *benchmark*. A partir dela, foi elaborada uma segunda tabela: a Tabela 16 mostra a contagem dos pares otimizados que tiveram retorno melhor, pior ou "empatados" em zero.

Nela podemos avaliar que a frequência com que os parâmetros otimizados superaram o *benchmark* foi de 52.8%; pouco mais que o dobro das vezes em que o *benchmark* foi melhor. Os resultados são favoráveis à otimização; porém são desanimadores. Era esperado que a otimização superasse o *benchmark* em todos os casos, ressalvados os eventuais empates; tal como ocorreu no conjunto de treino t_0 – conforme Tabela 17. Lá, os parâmetros

Tabela 15: Diferenças de MWRR t_1 entre otimizados e benckmarks

	Conservador	Moderado	Agressivo
BOVA11-BOVB11	-0.0117	-0.0134	-0.0080
BOVA11-PIBB11	0.0194	0.0194	0.0148
BOVB11-FIND11	0.0504	0.0504	0.0349
CMIG4-TAEE11	0.0000	0.0000	0.0029
ENGI11-ENGI4	0.0318	0.0226	0.0244
GGBR3-GOAU3	0.0107	0.0000	0.0007
GOAU3-GOAU4	0.0217	0.0222	-0.0054
PETR3-PETR4	0.0000	0.0000	-0.0204
SANB11-SANB4	0.0000	0.0000	0.0000
SAPR11-SAPR4	-0.0021	-0.0315	-0.0125
SMAC11-SMAL11	0.0532	0.0235	-0.0170
TAEE3-TAEE4	0.0407	0.0367	0.0198

otimizados venceram em 28 casos, contra apneas 2 do *benchmark*.[28].

Em termos de magnitude das diferenças: das 19 vezes em que a otimização superou o *benchmark*; na média o fez com 1.4 p.p. de vantagem. Das 9 vezes que foi pior, a média foi -0.3 p.p. Outra vez, vemos uma tímida superioridade dos parâmetros otimizados.

Em termos dos cenários, a superioridade dos otimizados foi diminuindo à medida que os cenários ficaram mais agressivos. No cenário "Agressivo" há quase um empate na frequência dos pares que foram melhores. Inclusive, é nesse cenário de *benchmark* que o par com o maior retorno dentre todos foi obtido – SMAC11-SMALL11 com 8.8%.

Tabela 16: Frequência dos resultados em t_1 comparados

Otimização	Conservador	Moderado	Agressivo	Total	% do Total
Melhor	7	6	6	19	52.8%
Pior	2	2	5	9	25.0%
Empate	3	4	1	8	22.2%

Tabela 17: Frequência dos resultados em t_0 comparados

Otimização	Conservador	Moderado	Agressivo	Total	% do Total
Melhor	9	9	10	28	77.8%
Pior	0	1	1	2	5.5%
Empate	3	2	1	6	16.7%

[28]Some-se a evidência que resutlada desta comparação da contagem entre os dois conjuntos às outras já apresentadas que indicam ter havido sobreajuste na otimização.

3.5.4.2 Comparação no nível da carteira

Conclusões ambíguas são obtidas também quando comparamos os resultados agregados em uma carteira formada pelos 12 pares. Quanto ao MWRR, a otimização superou os cenários "Conservador" e "Agressivo"; porém perdeu por uma margem relevante de 1.8 p.p. para o cenário "Moderado". Objetivamente, do ponto de vista do que se buscava otimizar, pode-se concluir que o resultado foi adverso à otimização; tendo ela sido superada pelo *benchmark* "Moderado".

No entando, ao se avaliar os demais indicadores da carteira; alguns pontos à favor da otimização devem ser considerados. A volatilidade de todos os cenários de *benchmark* foi maior que a volatilidade da carteira otimizada – especialmente no cenário Moderado. Quanto ao resultado financeiro simulado[29], os parâmetros otimizados tiveram larga vantagem.

Quanto ao "Hit ratio[30]" o resultado é favorável a otimização; com 94.2% de acerto em 91 trades contra 91.7% em 24 trades do *benchmark* Moderado. Embora o cenário conservador tenha 100% de acerto; apenas 2 operações foram abertas.

Simulação de Monte Carlo Uma hipótese que poderia ser aventada é que, devido a maior volatilidade do *benchmark* Moderado, as médias dos retornos poderiam não ser diferentes estatisticamente. Para testar esta hipótese, rodamos uma simulação de Monte Carlo comparando a carteira otimizada contra o cenário Moderado. Nessa simulação, usamos 5 mil trajetórias e 100 passos para cada carteira. A média de retorno anual da carteira otimizada foi de 13.45% e do *benchmark* Moderado foi 21.8%. Aplicando teste-t de duas caudas a hipótese nula foi rejeitada com $p - value = 0.0$; tanto na versão paramétrica do teste, quanto na não-paramétrica. Conclui-se que a maior volatilidade do *benckmark* Moderado não é suficiente para admitir que a média dos retorno seja indistinta entre eles.

[29]Todas as posições foram abertas com financeiro simulado de R$ 25,000.00; sem um limite para abertura de posições em simultâneo.

[30]Percentual de quantos *trades* foram encerrados com lucro dentre todos os trades encerrados. É um indicador da taxa de acertos da estratégia.

3.5.5 Discussão dos resultados

Analisados em conjunto, os resultados dos experimentos *não permitem* elaborarmos nenhuma conclusão mais assertiva sobre a pergunta central deste trabalho que é avaliar se a otimização dos parâmetros de *trading* melhora a performance da estratégia. Vamos primeiro partir das conclusões que são possíveis e que nos levaram a pensar dessa forma.

Primeiro, dado que o PSO-Linear que construímos foi validado por uma bateria de funções teste e também pela comparação com os sofisticados modelos presentes na literatura especializada sabemos, portanto, que o otimizador funciona. Segundo, some-se a isto todas as evidências apresentadas de que a performance desse otimizador no conjunto de treino excede a performance todos os *benchmarks*; assim como supera a si próprio, quando comparado com o conjunto de teste. A conclusão que se obtém desses resultados é que provavelmente houve sobreajuste do otimizador no conjunto de treino, minando sua capacidade de generalização para conjuntos de dados desconhecidos do modelo. Terceiro, os resultados obtido no conjunto de teste não foram completamente adversos; pelo contrário, trouxeram alguns resultados interessantes que merecem investigação mais aprofundada.

O problema que enfrentamos aqui é um problema comum em *machine learning*: o *dilema* entre variância e viés do modelo. Sem nos aprofundarmos, é possível dizer que o erro total E de um modelo é dado pelo somatório $E = \varepsilon_{vies} + \varepsilon_{variancia} + \xi$. O ε_{vies} é alto quando os modelos "super simplificam", incapazes de capturar a complexidade subjacente ao problema. Já o $\varepsilon_{variancia}$ é alto nos modelos que se especializam em excesso no conjunto de dados de treino "aprendendo" os ruídos ali presentes – em outras palavras, sobreajuste. Por fim, o ξ representa o erro irredutível, ou seja, o erro que é resultante da natureza do problema e sempre existirá, independente da qualidade do modelo.

No fundo, nosso objetivo inicial neste trabalho era saber se o ξ é irremediavelmente grande a ponto de não ser possível aplicar otimizadores na definição dos parâmetros de *trading* e obter bons resultados práticos. Portanto, duas hipóteses são possíveis:

(i) Nosso modelo de otimização sofreu de sobreajuste ($\varepsilon_{variancia}$ grande) e precisa de correções de modo a "aumentar seu viés" e – possivelmente – melhorar sua performance geral.

(ii) Nosso modelo está bem especificado e não sofre de sobreajuste; sendo que a performance que vimos no conjunto de testes é resultado de um ξ que é naturalmente grande como consequência da natureza estocástica do nosso problema.

Assumir que a segunda hipótese é correta encerraria a investigação por concluir que os otimizadores não podem ser usados para os propósitos aqui descritos. No entanto, temos evidências suficientes para não rejeitarmos a primeira hipótese. Na Parte 4 sugerimos alguns tópicos de investigação futura que tem como ponto de partida a hipótese (i).

4 CONSIDERAÇÕES FINAIS

Parece não ser incomum que, na busca por responder uma pergunta, cheguemos ao final da jornada com mais perguntas do que começamos. Este trabalho acrescenta mais um exemplo a essa realidade. Nosso objetivo inicial era avaliar se o uso de otimizadores na configuração dos parâmetros de *trading* poderia melhorar a performance da estratégia. Como vimos, há mais evidências em favor de uma resposta afirmativa à pergunta feita por este trabalho; porém as evidências contrárias são numerosas e fortes o bastante para não serem desprezadas. Essas evidências são um convite a novas investigações que enumeraremos mais adiante nestas considerações finais.

4.1 Contribuições do trabalho

Cointegração e *pairs trading* No tocante aos objetivos educacionais de um trabalho de conclusão de curso; a discussão feita na Parte 2 satisfaz como uma introdução ao ponto de conexão entre as estratégias de *trading* de reversão de média e o conceito de cointegração em séries temporais.

A baixa frequência de cointegração, observada na aplicação do teste de Engle&Granger em um grande universo de ativos na bolsa brasileira, é uma evidência de como são escassas as oportunidades de arbitragem estatística. Também vimos como a proximidade setorial dos papéis é um bom precursor da qualidade e da força da cointegração; corroborando com a teoria sobre a fundamentação econômica do *pairs trading*.

Otimizadores Na discussão sobre os otimizadores, talvez a maior contribuição seja a implementação dos algoritmos[1], especialmente o PSO em suas duas versões com adaptação dos coeficientes – linear e ESE/ELS. A má performance comparativa do algoritmo genético deve ser vista apenas no contexto deste trabalho e não deve ser extrapolada para toda classe de algoritmos genéticos.

[1] Códigos disponíveis nos Anexo B.2 e B.3.

Otimização de *pairs trading* Como dissemos, no tema central do trabalho os resultados não foram conclusivos. Porém algumas conclusões ainda assim são possíveis. Primeiro, diante da complexidade da implementação do algoritmo de otimização e – especialmente – do alto esforço computacional que esse processo requer, certamente essa não é a abordagem com melhor custo-benefício para se iniciar o refinamento de uma estratégia de *pairs trading*. Com isso queremos dizer que investigar diferentes partes do *design* da estratégia do trading – como por exemplo criar novas regras para a montagem e reversão, ou ainda sofisticar a análise de cointegração, aplicação de filtros, etc – pode trazer melhoras de performance a um esforço menor. Os otimizadores parecem ser mais adequados nos estágios finais de algoritmos de *trading* já maduros e avançados.

A segunda conclusão é a de que o dilema entre viés e variância é o grande antagonista na epopeia da otimização de *trading algorítmico*. Qualquer tentativa de otimizar parâmetros cujo processo subjacente seja uma processo estocástico precisa ter mecanismos para identificar e mitigar os efeitos desse problema. Dentre os tópicos para investigação futura que mencionaremos adiante o mais importante é, sem dúvidas, a detida análise das causas do possível sobreajuste e a construção das ferramentas para corrigi-lo.

4.2 Tópicos para investigação futura

Listamos aqui alguns dos assuntos que, partindo das novas perguntas encontradas encontradas neste trabalho, podem motivar futuras investigações e o avanço do conhecimento sobre o tema.

- A investigação e o desenvolvimento de critérios de parada que sejam eficazes; no sentido de interromper a iteração o mais rápido possível sem prejudicar da qualidade da convergência do algoritmo. Aplicações reais dessa otimização precisam ser feitas em tempo hábil para serem aplicadas e não podem consumir recursos computacionais em excesso.

  Uma ideia para este critério poderia utilizar da comparação do resultado obtido no conjunto de teste para determinar um ponto de parada que contribua para minimizar o efeito do sobreajuste, numa espécie de mecanismo de *early-stopping*.

- Repetir o procedimento de otimização utilizado aqui, porém com um *backtest* que cubra um hiato temporal mais longo. A nossa escolha por utilizar dados com frequência horária – com intuito de registrar operações de *daytrade* – ao longo de poucos meses pode ter contribuído para o problema de sobreajuste.

- Elaborar uma modificação no PSO seguindo a hipótese de que o sobreajuste pode ter sido provocado devido à natureza estocástica do problema; ou seja, que a otimização acaba por encontrar um ponto mínimo que é fruto de uma flutuação estocástica e não de um padrão que se repete ao longo do processo.

  Esta modificação deveria capacitar o otimizador a encontrar "regiões" de mínimo e não exatamente o "ponto" de mínimo. Deixamos aqui uma sugestão, como ponto de partida: para cada partícula do PSO, quando chamada a função objetivo, ela deve ser avaliada para o vetor da posição desse ponto, mas também para outros n pontos deslocados desse ponto central. Por exemplo, num problema de três dimensões, a partícula do PSO é inserida no centro de um tetraedro e a função objetivo é calculada para ele e para os quatro vértices desse tetraedro. A média do resultado desses cinco pontos é utilizada pelo PSO para calcular a próxima iteração.

- Testar o resultado da otimização substituindo o MWRR como função objetivo por algum outro indicador que reflita melhor a performance de carteiras ou mesmo rodar a otimização para a carteira como um todo. Vimos que o Hit ratio está muito alto. É possível montar estratégias rentáveis mesmo com taxa de acertos mais próximas à 50%; desde que a relação risco retorno seja ajustada equivalentemente.

REFERÊNCIAS

[1] KAYA, Y.; UYAR, M.; TEKIN, R. A novel crossover operator for genetic algorithms: Ring crossover. *CoRR*, abs/1105.0355, 01 2011.

[2] HASSANAT, A. et al. Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. *Information*, MDPI, v. 10, n. 12, p. 390, 2019.

[3] GLANTZ, M.; KISSELL, R. L. *Multi-asset risk modeling: techniques for a global economy in an electronic and algorithmic trading era*. [S.l.]: Academic Press, 2013.

[4] EHRMAN, D. S. *The handbook of pairs trading: strategies using equities, options, and futures*. [S.l.]: John Wiley & Sons, 2006.

[5] VIDYAMURTHY, G. *Pairs Trading: quantitative methods and analysis*. [S.l.]: John Wiley & Sons, 2004. v. 217.

[6] ENGLE, R. F.; GRANGER, C. W. Co-integration and error correction: representation, estimation, and testing. *Econometrica: journal of the Econometric Society*, p. 251–276, 1987.

[7] PFAFF, B. *Analysis of integrated and cointegrated time series with R*. [S.l.]: Springer Science & Business Media, 2008.

[8] BISGAARD, S.; KULAHCI, M. *Time series analysis and forecasting by example*. [S.l.]: John Wiley & Sons, 2011.

[9] TSAY, R. S. *Analysis of financial time series*. [S.l.]: John wiley & sons, 2005.

[10] DOLADO, J. J.; JENKINSON, T.; SOSVILLA-RIVERO, S. Cointegration and unit roots. *Journal of economic surveys*, Wiley Online Library, v. 4, n. 3, p. 249–273, 1990.

[11] BILGILI, F. Stationarity and cointegration tests: Comparison of engle-granger and johansen methodologies. *Erciyes Üniversitesi İktisadi ve İdari Bilimler Fakültesi Dergisi*, n. 13, p. 131–141, 1998.

[12] HARRIS, R. I. Using cointegration analysis in econometric modelling. Prentice Hall, 1995.

[13] RAYES, A. C. R. W. *Análise de estratégias de pairs tranding através dos métodos de cointegração e correlação aplicados ao mercado acionário brasileiro*. Dissertação (Mestrado) — Faculdades Ibmec, 2012.

[14] HAMILTON, J. D. *Time series analysis*. [S.l.]: Princeton University press, 1994.

[15] KOCHENDERFER, M. J.; WHEELER, T. A. *Algorithms for optimization*. [S.l.]: MIT Press, 2019.

[16] YANG, X. *Introduction to mathematical optimization. from linear programming to metaheuristics*. [S.l.]: Cambridge international science publishing, 2008.

[17] WOLPERT, D. H.; MACREADY, W. G. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, IEEE, v. 1, n. 1, p. 67–82, 1997.

[18] EBERHART, R. C.; SHI, Y.; KENNEDY, J. *Swarm intelligence*. [S.l.]: Elsevier, 2001.

[19] KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: IEEE. *Proceedings of ICNN'95-international conference on neural networks*. [S.l.], 1995. v. 4, p. 1942–1948.

[20] KIRANYAZ, S.; PULKKINEN, J.; GABBOUJ, M. Multi-dimensional particle swarm optimization in dynamic environments. *Expert Systems with Applications*, Elsevier, v. 38, n. 3, p. 2212–2223, 2011.

[21] HELWIG, S. *Particle Swarms for Constrained Optimization*. Dissertação (Mestrado) — Der Technischen Fakultat der Üniversitat Erlangen-N ürnberg, 2010.

[22] ZHAN, Z.-H. et al. Adaptive particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, IEEE, v. 39, n. 6, p. 1362–1381, 2009.

[23] UYAR, S.; ERYIGIT, G.; SARIEL, S. An adaptive mutation scheme in genetic algorithms for fastening the convergence to the optimum. In: *Proceedings of 3rd APIS: Asian Pacific international symposium on information technologies*. [S.l.: s.n.], 2004.

[24] UY, N. Q. et al. Initialising pso with randomised low-discrepancy sequences: the comparative results. In: IEEE. *2007 IEEE Congress on Evolutionary Computation*. [S.l.], 2007. p. 1985–1992.

[25] JAMIL, M.; YANG, X.-S. A literature survey of benchmark functions for global optimization problems. *Int. Journal of Mathematical Modelling and Numerical Optimisation*, v. 4, n. 2, p. 150–194, 2013.

[26] DOR, A. E.; CLERC, M.; SIARRY, P. A multi-swarm pso using charged particles in a partitioned search space for continuous optimization. *Computational Optimization and Applications*, Springer, v. 53, n. 1, p. 271–295, 2012.

ANEXO A – *BACKTESTS*

Detalhamento dos resultados obtidos no conjunto de testes t_1

A.1 Parâmetros otimizados

Par	Montagem	MWRR	Resultado R$	Dias de exp.
BOVA11_BOVB11	20220720_0.9558	0.004153	240.455517	0
BOVA11_BOVB11	20220720_0.9567	0.001299	75.287103	0
BOVA11_BOVB11	20220728_0.9523	0.003265	162.736142	0
BOVA11_BOVB11	20221020_0.9564	0.002397	110.513360	0
BOVA11_PIBB11	20220816_0.5509	0.003456	150.589758	59
BOVB11_BOVA11	20220715_1.0371	0.001501	85.289863	0
BOVB11_BOVA11	20220721_1.0367	0.002481	144.106328	0
BOVB11_BOVA11	20220728_1.0369	-0.028528	-1165.774806	57
BOVB11_FIND11	20220908_1.0564	0.008895	393.803310	43
ENGI11_ENGI4	20220824_5.7225	0.010193	547.283421	53
FIND11_BOVB11	20220715_0.901	0.014846	779.379148	4
FIND11_BOVB11	20220725_0.9078	0.034409	1877.556727	30
GGBR3_GOAU3	20220726_2.0158	0.010742	533.157765	8
GOAU4_GOAU3	20220715_1.0541	0.004440	222.580934	0
GOAU4_GOAU3	20220715_1.0583	0.005040	248.136319	2
GOAU4_GOAU3	20220719_1.053	0.005016	248.610145	1
GOAU4_GOAU3	20220719_1.0581	0.004037	200.693910	0
GOAU4_GOAU3	20220720_1.0544	0.004681	231.434151	42
SAPR11_SAPR4	20220715_4.9469	0.002027	100.621436	13
SAPR11_SAPR4	20220803_4.9714	-0.007827	-389.703518	68
SMAC11_SMAL11	20220809_0.5158	0.005029	244.256506	0
SMAC11_SMAL11	20220922_0.5201	0.002702	122.269606	1
SMAC11_SMAL11	20221003_0.5202	0.003060	139.327104	1
SMAL11_SMAC11	20220718_1.9127	0.000226	12.355681	2
SMAL11_SMAC11	20220720_1.9131	0.002746	151.385038	3
SMAL11_SMAC11	20220726_1.9084	0.000623	32.022643	0
SMAL11_SMAC11	20220726_1.9128	0.003743	202.815417	4
SMAL11_SMAC11	20220801_1.9126	0.000583	31.444411	0
SMAL11_SMAC11	20220802_1.9107	0.005247	294.704131	2
SMAL11_SMAC11	20220804_1.913	0.000637	30.053254	0
SMAL11_SMAC11	20220805_1.9114	0.002096	99.829732	0
SMAL11_SMAC11	20220808_1.9129	0.003240	157.886715	0

Continua

Par	Montagem	MWRR	Resultado R$	Dias de exp.
SMAL11_SMAC11	20220809_1.9119	-0.002715	-120.897488	1
SMAL11_SMAC11	20220810_1.9093	0.000423	18.619826	0
SMAL11_SMAC11	20220810_1.9118	0.001406	61.107042	1
SMAL11_SMAC11	20220811_1.902	-0.001074	-48.475865	1
SMAL11_SMAC11	20220815_1.9113	0.003328	147.739127	1
SMAL11_SMAC11	20220817_1.9048	0.001902	85.659813	0
SMAL11_SMAC11	20220817_1.9133	0.002441	109.748398	0
SMAL11_SMAC11	20220818_1.9099	0.002328	104.087286	0
SMAL11_SMAC11	20220819_1.9116	0.002892	132.032983	1
SMAL11_SMAC11	20220822_1.9118	-0.001026	-46.187449	1
SMAL11_SMAC11	20220824_1.9111	0.003360	148.412784	0
SMAL11_SMAC11	20220825_1.9131	0.001120	49.709255	1
SMAL11_SMAC11	20220826_1.9114	0.001170	51.982220	1
SMAL11_SMAC11	20220829_1.9124	0.003868	170.294328	2
SMAL11_SMAC11	20220831_1.9086	0.001049	45.245070	1
SMAL11_SMAC11	20220902_1.908	0.000410	18.164541	0
SMAL11_SMAC11	20220902_1.9124	0.003265	146.148060	0
SMAL11_SMAC11	20220905_1.9107	0.001255	56.052264	0
SMAL11_SMAC11	20220905_1.913	0.001053	46.312869	2
SMAL11_SMAC11	20220909_1.9128	0.002683	119.920417	0
SMAL11_SMAC11	20220912_1.9052	0.001319	59.551657	0
SMAL11_SMAC11	20220912_1.913	0.003477	154.203683	2
SMAL11_SMAC11	20220915_1.9103	0.000887	39.507655	2
SMAL11_SMAC11	20220919_1.9125	0.000282	12.431325	0
SMAL11_SMAC11	20220920_1.9115	0.002542	112.348921	0
SMAL11_SMAC11	20220921_1.905	0.001504	66.436475	0
SMAL11_SMAC11	20220921_1.9131	0.000556	24.237675	1
SMAL11_SMAC11	20220923_1.9104	0.000946	41.168846	1
SMAL11_SMAC11	20220926_1.9114	0.004039	182.539771	1
SMAL11_SMAC11	20220928_1.9053	0.001403	67.516595	0
SMAL11_SMAC11	20220928_1.9124	0.004413	197.227849	1
SMAL11_SMAC11	20220929_1.912	0.000922	43.459612	0
SMAL11_SMAC11	20220930_1.9129	-0.000724	-34.754310	0
SMAL11_SMAC11	20221003_1.9105	0.000092	4.070805	0
SMAL11_SMAC11	20221005_1.9078	0.000880	39.625821	0
SMAL11_SMAC11	20221005_1.9107	0.000295	13.150965	2
SMAL11_SMAC11	20221007_1.9115	0.000679	31.040749	0
SMAL11_SMAC11	20221010_1.9086	0.000906	41.282812	0
SMAL11_SMAC11	20221011_1.9113	0.002141	95.916297	0
SMAL11_SMAC11	20221014_1.91	0.001699	74.430295	1
SMAL11_SMAC11	20221017_1.9122	0.002356	102.397333	2
SMAL11_SMAC11	20221020_1.9101	0.001975	86.331410	3
SMAL11_SMAC11	20221025_1.9125	-0.000893	-38.882708	11
TAEE3_TAEE4	20220715_0.997	0.002355	118.363592	11
TAEE3_TAEE4	20220808_0.9958	0.002135	109.148825	1
TAEE3_TAEE4	20220812_0.9965	0.003386	171.859564	1
TAEE3_TAEE4	20220816_0.9949	0.002230	110.428155	2
TAEE3_TAEE4	20220819_0.9971	0.002938	146.168581	11
TAEE3_TAEE4	20220928_0.9955	0.003516	176.019223	3
TAEE3_TAEE4	20221004_0.997	0.002361	119.048579	0
TAEE3_TAEE4	20221005_0.997	0.002393	119.303003	2

Continua

Par	Montagem	MWRR	Resultado R$	Dias de exp.
TAEE3_TAEE4	20221010_0.997	0.002735	138.410644	6
TAEE4_TAEE3	20220801_0.9949	0.003378	165.568428	0
TAEE4_TAEE3	20220802_0.9957	0.002228	110.316909	3
TAEE4_TAEE3	20220809_0.9986	0.002470	120.163801	2
TAEE4_TAEE3	20220816_0.9985	0.002960	146.817517	0
TAEE4_TAEE3	20220905_0.9979	0.003955	201.030738	15
TAEE4_TAEE3	20221019_0.9962	0.002041	100.939063	1
TAEE4_TAEE3	20221020_0.9985	0.003177	157.263861	3

A.2 *Benchmark* – Conservador

Par	Montagem	MWRR	Resultado R$	Dias exp.
BOVA11_BOVB11	20220728_0.9523	0.003265	162.736142	0
SMAC11_SMAL11	20220809_0.5158	0.005029	244.256506	0
SMAL11_SMAC11	20220829_1.8968	0.004026	176.534331	1
SMAL11_SMAC11	20220913_1.8959	0.003499	150.897941	1

A.3 *Benchmark* – Moderado

Par	Montagem	MWRR	Resultado R$	Dias exp.
BOVA11_BOVB11	20220728_0.9523	0.003265	162.736142	0
BOVB11_BOVA11	20220720_1.0356	0.001827	105.453972	0
BOVB11_BOVA11	20220728_1.0346	-0.000093	-4.650644	0
ENGI11_ENGI4	20220922_5.66	0.009223	488.455597	1
GGBR3_GOAU3	20220726_2.0158	0.010742	533.157765	8
GOAU4_GOAU3	20220715_1.0541	0.007523	379.355866	2
GOAU4_GOAU3	20220719_1.053	-0.008114	-403.420426	36
SAPR11_SAPR4	20220726_4.8937	0.003662	183.852354	1
SAPR11_SAPR4	20220727_4.8915	0.003299	169.795700	2
SAPR11_SAPR4	20220805_4.8942	0.002986	154.537341	13
SAPR11_SAPR4	20220829_4.8849	0.002754	137.395378	2
SAPR11_SAPR4	20220908_4.8835	0.003323	170.070026	1
SAPR11_SAPR4	20220913_4.8799	0.003233	158.452877	7
SAPR11_SAPR4	20220927_4.8817	0.003572	174.089637	2
SAPR11_SAPR4	20221018_4.877	0.002931	146.450816	2
SMAC11_SMAL11	20220809_0.5158	0.005029	244.256506	0
SMAL11_SMAC11	20220811_1.902	0.000055	2.492719	1
SMAL11_SMAC11	20220829_1.8968	0.004026	176.534331	1
SMAL11_SMAC11	20220830_1.8984	0.007197	310.778454	1
SMAL11_SMAC11	20220913_1.8982	0.007043	306.665148	1
SMAL11_SMAC11	20220921_1.9024	0.003322	144.303598	1
SMAL11_SMAC11	20221010_1.9023	0.004778	213.668053	1
SMAL11_SMAC11	20221014_1.902	0.003461	148.826867	1
TAEE3_TAEE4	20220816_0.987	0.004052	200.697000	1

A.4 *Benchmark* – Agressivo

Par	Montagem	MWRR	Resultado R$	Dias exp.
BOVA11_BOVB11	20220718_0.9575	0.002029	114.888141	1
BOVA11_BOVB11	20220720_0.9558	0.004153	240.455517	0
BOVA11_BOVB11	20220720_0.9567	0.002103	120.529681	1
BOVA11_BOVB11	20220722_0.9571	0.001706	98.329150	3
BOVA11_BOVB11	20220728_0.9523	0.003265	162.736142	0
BOVA11_BOVB11	20221020_0.9564	0.002397	110.513360	0
BOVA11_PIBB11	20221014_0.548	0.004548	190.000801	0
BOVB11_BOVA11	20220715_1.0371	0.002318	132.912862	1
BOVB11_BOVA11	20220721_1.0367	0.002481	144.106328	0
BOVB11_BOVA11	20220728_1.0369	-0.027994	-1143.928284	57
BOVB11_FIND11	20220920_1.0336	0.007392	331.134854	8
BOVB11_FIND11	20221003_1.0363	0.008044	370.910962	4
ENGI11_ENGI4	20220824_5.7225	0.006963	369.766859	2
ENGI11_ENGI4	20220830_5.8165	0.007848	403.386768	1
ENGI11_ENGI4	20220902_5.7877	0.010251	526.756043	1
ENGI11_ENGI4	20220905_5.7884	-0.007949	-416.677796	10
ENGI11_ENGI4	20220920_5.7255	-0.008559	-459.844150	4
ENGI11_ENGI4	20220929_5.8132	-0.013107	-651.392985	10
ENGI11_ENGI4	20221014_5.6897	0.006492	334.925303	0
ENGI11_ENGI4	20221014_5.7187	0.007291	384.755343	1
ENGI11_ENGI4	20221018_5.8254	0.008511	401.107332	1
ENGI11_ENGI4	20221020_5.8112	-0.010985	-510.208050	3
GGBR3_GOAU3	20220726_2.0	0.014766	728.231414	8
GGBR3_GOAU3	20220726_2.0456	-0.008310	-419.978717	0
GGBR3_GOAU3	20220805_2.0509	0.006907	335.619931	1
GGBR3_GOAU3	20220808_2.0749	-0.008753	-431.101109	2
GGBR3_GOAU3	20220810_2.0586	0.007568	376.887890	3
GGBR3_GOAU3	20220816_2.0642	0.006554	336.905731	1
GGBR3_GOAU3	20220818_2.0681	-0.008378	-407.868131	1
GGBR3_GOAU3	20220823_2.0671	-0.011406	-548.918129	39
GGBR3_GOAU3	20221019_2.0254	0.010597	546.577143	4
GOAU4_GOAU3	20220715_1.0541	0.007523	379.355866	2
GOAU4_GOAU3	20220719_1.0581	-0.008882	-437.535463	13
GOAU4_GOAU3	20220808_1.051	0.006471	320.342330	29
GOAU4_GOAU3	20220919_1.067	-0.008352	-415.752105	4
GOAU4_GOAU3	20220923_1.0559	0.005694	283.776829	5
GOAU4_GOAU3	20220930_1.0586	0.006874	342.703180	0
GOAU4_GOAU3	20221014_1.0608	0.010376	523.007377	1
GOAU4_GOAU3	20221019_1.0584	0.007813	387.776129	4
PETR4_PETR3	20220819_0.8971	0.004091	206.260586	2
PETR4_PETR3	20220823_0.8961	0.001272	60.171671	4
PETR4_PETR3	20220830_0.895	0.004201	218.874170	17
PETR4_PETR3	20220929_0.8925	0.006112	319.345196	1
PETR4_PETR3	20221004_0.8915	0.003995	204.570963	7
SAPR11_SAPR4	20220718_4.9333	-0.005118	-254.590901	6
SAPR11_SAPR4	20220726_4.9103	0.002801	141.645312	4
SAPR11_SAPR4	20220802_4.9147	0.004676	239.770279	1
SAPR11_SAPR4	20220804_4.916	-0.004578	-230.104014	4

Continua

Par	Montagem	MWRR	Resultado R$	Dias exp.
SAPR11_SAPR4	20220810_4.9024	0.002890	146.384893	10
SAPR11_SAPR4	20220824_4.9151	-0.006187	-310.532518	4
SAPR11_SAPR4	20220830_4.8842	0.002967	145.076061	1
SAPR11_SAPR4	20220831_4.9124	-0.004751	-231.800847	9
SAPR11_SAPR4	20220915_4.8949	0.004042	208.549332	6
SAPR11_SAPR4	20220926_4.8895	0.002578	127.934098	3
SAPR11_SAPR4	20220929_4.8988	0.004516	228.966505	2
SAPR11_SAPR4	20221006_4.9011	0.002887	144.301458	9
SAPR11_SAPR4	20221020_4.9096	0.004182	209.616471	3
SMAC11_SMAL11	20220808_0.5187	0.002132	110.253416	1
SMAC11_SMAL11	20220816_0.5187	0.005019	227.664963	1
SMAC11_SMAL11	20220824_0.5186	0.003219	144.105827	1
SMAL11_SMAC11	20220718_1.9068	0.006741	360.316209	5
SMAL11_SMAC11	20220726_1.9084	0.012610	686.526631	7
SMAL11_SMAC11	20220811_1.902	0.000055	2.492719	1
SMAL11_SMAC11	20220829_1.8968	0.004026	176.534331	1
SMAL11_SMAC11	20220830_1.8984	0.007197	310.778454	1
SMAL11_SMAC11	20220831_1.9086	0.003835	164.734162	2
SMAL11_SMAC11	20220906_1.9073	0.003769	162.403514	2
SMAL11_SMAC11	20220912_1.9052	0.005630	250.129519	2
SMAL11_SMAC11	20220916_1.9077	-0.000178	-7.886342	2
SMAL11_SMAC11	20220921_1.905	0.003304	144.177931	1
SMAL11_SMAC11	20220923_1.9066	0.003824	165.591169	2
SMAL11_SMAC11	20220928_1.9053	0.007066	317.104242	1
SMAL11_SMAC11	20221004_1.9075	0.003527	157.527966	5
SMAL11_SMAC11	20221014_1.902	0.003461	148.826867	1
TAEE11_CMIG4	20220817_3.2413	-0.013883	-694.990893	5
TAEE11_CMIG4	20220824_3.1541	0.010796	530.915042	2
TAEE3_TAEE4	20220720_0.9909	0.002020	100.645497	1
TAEE3_TAEE4	20220721_0.9917	0.002774	138.195362	1
TAEE3_TAEE4	20220816_0.9899	0.002610	128.808246	1
TAEE3_TAEE4	20220927_0.9918	0.003094	156.833297	1
TAEE3_TAEE4	20220930_0.9894	0.002023	100.547642	1
TAEE4_TAEE3	20220919_0.9927	0.002615	129.141984	1
TAEE4_TAEE3	20220920_0.9928	0.002947	146.595024	1

ANEXO B – CÓDIGOS EM PYTHON

B.1 Cointegração

```python
# main packages
import statsmodels as sm
from statsmodels.tsa.stattools import adfuller

# local packages
#N/A

# cointegration modules
#N/A

class EngleGranger():
    _r2_min = 0.60

    def __init__(self, series_1, series_2, pvalue=0.05):
        self.series_1 = series_1
        self.series_2 = series_2
        self._pvalue_max = pvalue
        self._null_h = True
        self.criteria = {} # criteria to reject the null hyposthesis (True==passed, favours H0 rejection)

    def run(self):
        self.lr_model = self._linear_regression(self.series_1, self.series_2)
        self.residuals = self.lr_model.resid
        self.lr_intercept = self.lr_model.params[0]
        self.lr_beta = self.lr_model.params[1]
        self.adf_results = self._adfuller_on_resid()
        self._tests()

    def _linear_regression(self, series_1, series_2):
        independent_var = sm.tools.tools.add_constant(series_2)
        lr_model = sm.regression.linear_model.OLS(endog=series_1, exog=independent_var)
        lr_model = lr_model.fit(cov_type='HC0', fit_intercept=True) # entender o pq do HC0
        return lr_model

    def _adfuller_on_resid(self):
        return adfuller(self.residuals, regression='c', maxlag=1) # lag=1 accordint to CQF lecture

    def _tests(self):
        self._test_coeffs_pvalues()
```

```
        self._test_lr_r2()
        self._test_adf_pvalue()
        self._test_coint()
        self._get_critical_level()

    def _test_coeffs_pvalues(self):
        self.criteria['lr_coeff_pvalue'] = True
        for p in self.lr_model.pvalues:
            if p >= self._pvalue_max:
                self.criteria['lr_coeff_pvalue'] = False

    def _test_lr_r2(self):
        if self.lr_model.rsquared_adj < self._r2_min:
            self.criteria['lr_r2'] = False
        else:
            self.criteria['lr_r2'] = True

    def _test_adf_pvalue(self):
        self.adf_pvalue = self.adf_results[1]
        if  self.adf_pvalue <= self._pvalue_max:
            self.criteria['adf_pvalue'] = True
        else:
            self.criteria['adf_pvalue'] = False

    def _test_coint(self):
        if all(self.criteria.values()):
            self.criteria['cointegrated'] = True
            self.cointegrated = 1
        else:
            self.criteria['cointegrated'] = False
            self.cointegrated = 0

    def _get_critical_level(self):
        self.adf_critical_level = None
        for level, crit_value in self.adf_results[4].items():
            if self.adf_results[0] < crit_value:
                self.adf_critical_level = level
```

```
# main packages
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import matplotlib.gridspec as gs
import matplotlib.dates as mdates



# local packages
#N/A

# cointegration modules
from cointegration.analysis.englegranjer import EngleGranger
from cointegration.tools.pairs_maker import BuildUniquePairID

class Cointegration_Analysis():
    _pvalue_max = 0.1

    def __init__(self, data, pair_inversion=False, method='engle_granjer'):
        self.method = method
        self.tseries = data['pair']
        self._qry_data = data['qry_data']
        self.results = None
        self.timestamp = self.tseries.index
        self.symb_1 = self.tseries.columns[0]
        self.symb_2 = self.tseries.columns[1]
        self.raw_series_symb_1 = self.tseries[self.symb_1].to_list()
        self.raw_series_symb_2 = self.tseries[self.symb_2].to_list()
        self.base100_symb_1 = self.tseries[self.symb_1] / (self.tseries[self.symb_1][0]/100)
        self.base100_symb_2 = self.tseries[self.symb_2] / (self.tseries[self.symb_2][0]/100)

    def get_results(self):
        if self.results is None:
            self._run_analysis()
            self._get_basic_pair_tseries()
            self.results = self._store_basic_results()
        return self.results

    def _run_analysis(self):
        if self.method == 'engle_granjer':
            self.model = EngleGranger(self.raw_series_symb_1, self.raw_series_symb_2, pvalue=self._pvalue_max)
            self.model.run()
        elif self.method == 'johansen':
            pass

    def _store_basic_results(self):
        res = {}
        res['dep_var'] = self.symb_1
        res['indep_var'] = self.symb_2
        res['pair_id'] = res['dep_var'] +' / '+ res['indep_var']
        res['unique_pair_id'] = BuildUniquePairID(res['indep_var'], res['dep_var'], form='single_pair_from_symbol').build
        res['period_days_real'] = self._get_analysis_period_days()
```

```
        res['period_days'] = self._get_analysis_period_days_adj(res['period_days_real'])
        res['lr_beta'] = round(self.model.lr_beta, 4)
        res['lr_r2'] = round(self.model.lr_model.rsquared_adj, 4)
        res['coint_pvalue'] = round(self.model.adf_pvalue, 4)
        res['cointegrated'] = self.model.cointegrated
        if self.model.adf_critical_level is None:
            res['adf_critical'] = 1
        else:
            res['adf_critical'] = int(self.model.adf_critical_level[:-1])/100
        res['coint_score'] = res['lr_r2'] * (1 - res['coint_pvalue']) * int(self.model.criteria['lr_coeff_pvalue']) * (1-
        res['dep_var_last_price'] = round(self.tseries[self.symb_1][-1], 4)
        res['indep_var_last_price'] = round(self.tseries[self.symb_2][-1], 4)
        res['ratio_last'] = round(self.tseries['ratio'][-1], 4)
        res['ratio_vol'] = self._ratio_vol
        res['ratio_mean'] = round(self.tseries['ratio'].mean(), 4)
        res['ratio_std'] = round(self.tseries['ratio_std'][-1], 4)
        res['ratio_last_zscore'] = round(self.tseries['ratio_zscore'].iloc[-1], 3)
        # res['ratio_mean_plus_2std'] = round(self.tseries['ratio_mean_plus_2std'].iloc[0], 4)
        # res['ratio_mean_minus_2std'] = round(self.tseries['ratio_mean_minus_2std'].iloc[0], 4)
        res['last_update'] = self._last_update_timestp()
        return res

    def _get_basic_pair_tseries(self):
        ''' Create the timeseries with calculations'''
        self.tseries[self.symb_1] = self.raw_series_symb_1
        self.tseries[self.symb_2] = self.raw_series_symb_2
        self.tseries['ratio'] = self.tseries[self.symb_1] / self.tseries[self.symb_2]
        self.tseries['ratio_chg'] = self.tseries['ratio'].pct_change() # check if its not the same
        self.tseries['ratio_mean'] = self.tseries['ratio'].mean()
        self.tseries['ratio_std'] = self.tseries['ratio'].std()
        self.tseries['ratio_vol'] = round(self._get_ratio_vol(), 4)
        self.tseries['ratio_zscore'] = (self.tseries['ratio'] - self.tseries['ratio_mean']) / self.tseries['ratio_std']

    def _get_complement_pair_tseries(self):
        self.tseries['ratio_mean_minus_2std'] = self.tseries['ratio_mean'] - 2*self.tseries['ratio_std']
        self.tseries['ratio_mean_plus_2std'] = self.tseries['ratio_mean'] + 2*self.tseries['ratio_std']

        # deprecate it since moved to signals
        self.tseries['log_dep_var'] = np.log(self.tseries[self.symb_1])
        self.tseries['log_indep_var'] = np.log(self.tseries[self.symb_2])
        self.tseries['log_return_dep_var'] = self.tseries['log_dep_var'].pct_change()
        self.tseries['log_return_indep_var'] = self.tseries['log_indep_var'].pct_change()
        self.tseries['acc_log_return_dep_var'] = self.tseries['log_return_dep_var'].cumsum()
        self.tseries['acc_log_return_dep_var'].fillna(value=0, inplace=True)
        self.tseries['acc_log_return_indep_var'] = self.tseries['log_return_indep_var'].cumsum()
        self.tseries['acc_log_return_indep_var'].fillna(value=0, inplace=True)

        # log of standard prices (used for cointegration analysis ans residuals)
        self.tseries['dep_var_stdp'] = self.base100_symb_1
        self.tseries['indep_var_stdp'] = self.base100_symb_2
        self.tseries['log_dep_var_stdp'] = np.log(self.tseries['dep_var_stdp'])
        self.tseries['log_indep_var_stdp'] = np.log(self.tseries['indep_var_stdp'])
        self.tseries['log_stdratio'] = self.tseries['log_dep_var_stdp'] / self.tseries['log_indep_var_stdp']
        self.tseries['log_stdratio_chg'] = self.tseries['log_stdratio'].pct_change() # checi if any  calc need real ratio
```

```
    # series pondered by beta
    self.tseries['spread_beta_hedge'] =  self.tseries[self.symb_1] - self.tseries[self.symb_2] * self.model.lr_beta -
    self.tseries['spread_std'] =  self.tseries['spread_beta_hedge'].std()
    self.tseries['spread_mean'] =  self.tseries['spread_beta_hedge'].mean()

def _get_analysis_period_days(self):
    return math.ceil(len(self.timestamp) / (self._get_timestamps_interval()/252))

def _get_analysis_period_days_adj(self, real_p):
    array = np.array([[1260, 756, 504, 252, 126, 63, 21, 5], [1260, 756, 504, 252, 126, 63, 21, 5]])
    array[1,:] = np.abs(real_p - array[0])
    index = np.where(array[1, :]==min(array[1, :]))[0][0]
    return array[0,index]

def _get_timestamps_interval(self):
    '''Divide 1 year on negotiation seconds by the negotiation seconds of the given interval'''
    timeframe = pd.DataFrame(self.timestamp, columns=['timestamp'])
    timeframe['gap'] = timeframe.timestamp.diff()
    most_common_interval = timeframe.groupby('gap').count()['timestamp'].idxmax()
    timeframe_secs = (60*60*9*252) / (most_common_interval.days*9*60*60+(most_common_interval.total_seconds()-((60*60
    return timeframe_secs

def _get_ratio_vol(self):
    unit_ratio_vol = self.tseries['ratio_chg'].std()
    self._ratio_vol = unit_ratio_vol * self._get_timestamps_interval() **0.5
    return self._ratio_vol

def _last_update_timestp(self):
    '''Manages cases where live=False and returns the correct last timestamp update'''
    if self._qry_data['live_qry_time'] is None:
        return str(self.tseries.index[-1])
    else:
        return str(self._qry_data['live_qry_time'])

def plot(self):
    DATEFORMAT = mdates.DateFormatter('%Y-%m')
    plt.rcParams['lines.linewidth'] = 0.65
    plt.rcParams['xtick.labelsize'] = 7
    plt.rcParams['ytick.labelsize'] = 7
    self._get_complement_pair_tseries()
    tseries = self.tseries
    # plt.rcParams.update({'font.size': 6})
    # fig, ((ax1), (ax2, ax4)) = plt.subplots(nrows=2, ncols=2, figsize=(11.7, 8.3), dpi=350, constrained_layout=True)
    # fig.suptitle(tseries.columns[0] + ' / ' + tseries.columns[1], fontsize=18)

    fig = plt.figure(dpi=300, constrained_layout=True)
    GS = gs.GridSpec(nrows=2, ncols=2, figure=fig,
                     width_ratios = [0.5, 0.5],
                     height_ratios = [0.45, 0.55])
    ax1 = fig.add_subplot(GS[0, 0:2])
    ax2 = fig.add_subplot(GS[1 , 0])
    ax3 = fig.add_subplot(GS[1 , 1])
```

```
# ax1 = prices
ax1.plot(tseries['acc_log_return_dep_var'], c='b', label=tseries.columns[0])
ax1.plot(tseries['acc_log_return_indep_var'], c='m', label=tseries.columns[1])
ax1.set_ylabel('Log acc change')
ax1.axhline(y=0, c='grey')
# ax1.set_title('Log returns')
ax1.legend(prop={'size': 5})

# ax 2 = ratio
ax2.plot(tseries['ratio'], c='olivedrab')
ax2.axhline(y=tseries['ratio_mean'].iloc[0], c='grey', linestyle='-', label='Média')
ax2.axhline(y=tseries['ratio_mean_plus_2std'].iloc[0], c='grey', linestyle='--', label=r'$+2\sigma$')
ax2.axhline(y=tseries['ratio_mean_minus_2std'].iloc[0], c='grey',linestyle='--', label=r'$-2\sigma$')
ax2.set_title('Ratio: '+ tseries.columns[0] + '/' + tseries.columns[1])
ax2.legend(prop={'size': 5})
ax2.set_xticklabels(self.timestamp, rotation=45)
ax2.xaxis_date()
ax2.xaxis.set_major_formatter(DATEFORMAT)

# ax 3 = pondered beta
ax3.plot(self.timestamp, tseries['spread_beta_hedge'], c='goldenrod')
ax3.axhline(y=0, linestyle='-', c='grey', label='Média')
ax3.axhline(y=2*tseries['spread_std'].iloc[0], c='grey', linestyle='--', label=r'$+2\sigma$')
ax3.axhline(y=-2*tseries['spread_std'].iloc[0], c='grey', linestyle='--', label=r'$+2\sigma$')
ax3.set_title(r'Spread - $ \beta $ Weighted')
ax3.legend(prop={'size': 5})
ax3.set_xticklabels(self.timestamp, rotation=45)
ax3.xaxis_date()
ax3.xaxis.set_major_formatter(DATEFORMAT)

plt.suptitle(self.results['unique_pair_id'], fontsize=12)

return fig
```

B.2 Algoritmo Genético

```
# main packages
import numpy as np

# local packages
# N/A

# cointegration modules
from cointegration.optimizer.geneticalgo.specie import Specie
from cointegration.optimizer.geneticalgo.params import ParamsInterpreter
from cointegration.optimizer.geneticalgo.survival import Survival
from cointegration.optimizer.geneticalgo.crossover import Crossover
from cointegration.optimizer.geneticalgo.mutation import Mutation, AdaptiveMutation
from cointegration.optimizer.pso.initiator import ParticlesInitiator


class GenAlgoOptimizer():

    def __init__(self, population=None, max_iter=None, fitness_func=None, params=None, adaptation=None, **kwargs):
        self._max_iter = max_iter
        self._pop_size = self._assert_pop_size(population)
        self._fit_func = fitness_func
        self.fitness_history = {'best': [], 'average': []}
        self._pinterpreter = ParamsInterpreter(**params)
        self._instance_mutation(adaptation)
        self._population_initialization()

    def optimize(self):
        for iter_ in range(self._max_iter):
            self._run_fitness(iter_)
            self.survivors = Survival(method='best_half').select(self.population)
            offspring = Crossover(self.survivors, method='one_point').mate()
            self.offspring = self._mutation.mutate(offspring.copy())
            self._reinstanciate_population()
            self._update_adaptive_mutation_params(iter_, offspring)

    def _assert_pop_size(self, pop_size):
        if pop_size % 4 != 0:
            raise ValueError('[GenAlgo Optimizer]: Population size must be in multiples of 4')
        else:
            return pop_size

    def _population_initialization(self):
        initiator = ParticlesInitiator(params_interp=self._pinterpreter, particles=self._pop_size)
        initiator.calc_initial_position()
        genomes = initiator._initial_position_vector
        self.population = []
        for i in range(self._pop_size):
            genome = [genomes[0][i], genomes[1][i]]
            individual = Specie()
            individual.genome = genome
            self.population.append(individual)
```

```
def _instance_mutation(self, adaptation):
    self._adaptive_mutation = AdaptiveMutation(method='rank', max_gen=self._max_iter, adapt=adaptation, pop_size=self
    self._adaptive_mutation_params = self._adaptive_mutation.get_params(generation=0)
    self._mutation = Mutation(pinterpreter=self._pinterpreter)
    self._mutation.update_adaptive(self._adaptive_mutation_params)

def _run_fitness(self, iter_):
    self.pop_fitness = self._run_real_fitness(iter_)
    self._run_std_fitness(iter_, self.pop_fitness)
    self.fitness_history['average'].append(np.mean(self.pop_fitness))
    self.fitness_history['best'].append(min(self.pop_fitness))

def _run_real_fitness(self, iter_):
    for individual in self.population:
        result = self._fit_func.run(individual.genome)
        individual.fitness = result
    pop_fitness = []
    for individual in self.population:
        pop_fitness.append(individual.fitness)
    return pop_fitness

def _run_std_fitness(self, iter_, pop_fitness):
    self._std_result = self._calc_std_fitness(pop_fitness)
    for i, individual in enumerate(self.population):
        individual.std_fitness = self._std_result[i]

def _calc_std_fitness(self, real_fitness):
    max_ = max(real_fitness)
    min_ = min(real_fitness)
    std_results = []
    if (max_ - min_) == 0:
        return np.linspace(0, 1, num=len(real_fitness))
    else:
        for res in real_fitness:
            z = (max_-res) / (max_ - min_)
            std_results.append(z)
        return std_results

def _update_adaptive_mutation_params(self, iter_, offspring):
    self._adaptive_mutation_params = self._adaptive_mutation.get_params(iter_, avg_fit=self.pop_fitness)
    self._mutation.update_adaptive(self._adaptive_mutation_params)

def _reinstanciate_population(self):
    self.population = self.survivors.copy()
    self.population.extend(self.offspring)
```

```
# cointegration modules
#N/A

# main packages
import numpy as np
import random

# local packages
#N/A

class Survival():

    def __init__(self, method='sus'):
        methods = {'best_individual': self._absolute_best_individual, # select the best individual among pupulation
                   'best_half': self._absolute_best_half,
                   'roulette_wheel': self._roulette_wheel,
                   'sus': self._stochastic_universal_sampling,
                   }
        self._method = methods[method]

    def select(self, population):
        self._pop = population
        self._pop_size = len(population)
        self._std_fitness = [individual.std_fitness for individual in self._pop]
        survivors_index = self._method()
        survivors = []
        for index in survivors_index:
            survivors.append(self._pop[index])
        return survivors

    def _roulette_wheel(self):
        wheel = self._build_wheel()
        pointers = [random.random() for i in self._pop_size]
        indexed = np.array([np.arange(self._pop_size), self._std_fitness, wheel])
        prior_selection = self._select_in_wheel(indexed, pointers)
        selection = self._enforce_unique_survivors(prior_selection, indexed)
        return self._order_selection(selection, indexed)

    def _stochastic_universal_sampling(self):
        wheel = self._build_wheel()
        pointers = np.linspace(0.01, 0.99, num=int(self._pop_size/2))
        indexed = np.array([np.arange(self._pop_size), self._std_fitness, wheel])
        prior_selection = self._select_in_wheel(indexed, pointers)
        selection = self._enforce_unique_survivors(prior_selection, indexed)
        return self._order_selection(selection, indexed)

    def _build_wheel(self):
        cum_prob = []
        sum_ = sum(self._std_fitness)
        for res in self._std_fitness:
            probability = res / sum_
            try:
                cum_prob.append(probability + cum_prob[-1])
            except IndexError:
```

```
            cum_prob.append(probability)
    return cum_prob

def _select_in_wheel(self, indexed, pointers):
    selected = []
    i = 0
    for point in pointers:
        for col_ind, wheel_mark in zip(indexed[0, i:], indexed[2, i:]):
            if wheel_mark >= point:
                selected.append(int(col_ind))
                i = int(col_ind)
                break
    return selected

def _enforce_unique_survivors(self, prior_selection, indexed):
    '''When wheel techniques selecte repeated survivors, this methodes re-select based odn best performer
    to reduce excessive selective pressure, if no repetitions, than pior wheel selection returned'''
    offspring_len = len(prior_selection)
    unique_prior = len(set(prior_selection))
    if offspring_len == unique_prior:
        return prior_selection
    else:
        selected = list(set(prior_selection))
        ordered_best = indexed[0,indexed[1].argsort()]
        while True:
            for i in range(ordered_best.shape[0]):
                candidate = int(ordered_best[-i-1])
                if not candidate in selected:
                    selected.append(candidate)
                    break
            if len(selected) == len(prior_selection):
                break
        return selected

def _order_selection(self, selection, indexed):
    ordered = []
    ordered_indexed = list(indexed[:, np.argsort(indexed[1,:])][0].astype(int))
    for i in range(len(ordered_indexed)):
        candidate = ordered_indexed[-i-1]
        if candidate in selection:
            ordered.append(candidate)
    return ordered

def _absolute_best_half(self):
    indexed = np.array([np.arange(self._pop_size), self._std_fitness])
    best = indexed[-1:, ].argsort()
    return best[-1, -int(self._pop_size/2):]

def _absolute_best_individual(self):
    indexed = np.array([np.arange(self._pop_size), self._std_fitness])
    best = indexed[-1:, ].argsort()
    return best[-1, -1]
```

```
# main packages
import numpy as np
import random

# local packages
#N/A

# cointegration modules
#N/A


class Mutation():

    def __init__(self, pinterpreter=None):
        self._pinterpreter = pinterpreter

    def update_adaptive(self, adapt_par):
        self._list_child_params = adapt_par

    def mutate(self, offspring):
        for i, child in enumerate(offspring):
            self._set_individual_adaptive_params(i)
            for g, gene in enumerate(child.genome):
                if self._skip_mutation():
                    continue
                gene_name = self._pinterpreter.get_name(g)
                type_ = self._pinterpreter.parameters[gene_name]['types']
                if type_ == list:
                    offspring[i].genome[g] = self._mutate_list(gene_name)
                elif type_ == float:
                    offspring[i].genome[g] = self._mutate_float(gene, gene_name)
                elif type_ == int:
                    offspring[i].genome[g] = int(self._mutate_float(gene, gene_name))
                elif type_ == bool:
                    offspring[i].genome[g] = self._mutate_bool(gene)
        return offspring

    def _set_individual_adaptive_params(self, child_index):
        dict_ = self._list_child_params[child_index]
        self._randomize_mutation = dict_['randomize_mutation']
        self._rate = dict_['rate']
        self._lambda = dict_['lamb']

    def _skip_mutation(self):
        if self._randomize_mutation:
            return bool(np.random.binomial(1, 1-self._rate))
        else:
            return False

    def _mutate_list(self, gene_name):
        return np.random.choice(self._pinterpreter.parameters[gene_name]['group'])

    def _mutate_float(self, gene, gene_name):
        while True:
```

```
            var = self._pert_dist(lamb=self._lambda)
            new_value = gene * (1+ var)
            new_value = self._enforce_boundaries(gene_name, new_value)
            if len(self._pinterpreter.parameters[gene_name]['invalid'])==0:
                return new_value
            elif not new_value in self._pinterpreter.parameters[gene_name]['invalid']:
                return new_value

    def _enforce_boundaries(self, gene_name, new_value):
        '''If value is outside limits, return the limit value (shrink method) '''
        if new_value > self._pinterpreter.parameters[gene_name]['ub']:
            return self._pinterpreter.parameters[gene_name]['ub']
        elif new_value < self._pinterpreter.parameters[gene_name]['lb']:
            return self._pinterpreter.parameters[gene_name]['lb']
        else:
            return new_value

    def _pert_dist(self, low=-1, mode=0, high=1, lamb=4):
        if mode is None:
            mode = abs
        r = high - low
        alpha = 1 + lamb * (mode - low) / r
        beta = 1 + lamb * (high - mode) / r
        return low + random.betavariate(alpha, beta) * r

    def _mutate_bool(self, gene):
        if bool(np.random.binomial(1, self._rate)):
            return not gene
        else:
            return gene



class AdaptiveMutation():
    _fixed = {'rate': 0.15, 'lamb': 4, 'randomize_mutation': True}

    def __init__(self, max_gen=None, adapt=True, method='linear', pop_size=None):
        self._max_gen = max_gen
        self._offspring_size = int(pop_size / 2)
        self._adapt = adapt
        self._load_adapt_method(method)

    def _load_adapt_method(self, method):
        methods = {'linear': self._linear, 'phased': self._phased, 'rank': self._rank}
        self._method = methods[method]

    def _linear(self, generation=int, **kwargs):
        rate_boundaries = [0.01, 0.5]
        lambda_boundaries = [0, 50]
        randomize = True
        factor = generation / self._max_gen
        rate = factor * (rate_boundaries[0] - rate_boundaries[1]) + rate_boundaries[1]
        lamb = factor * (lambda_boundaries[1] - lambda_boundaries[0]) + lambda_boundaries[0]
        return {'rate': rate, 'lamb': lamb, 'randomize_mutation': randomize}
```

```
def _rank(self, generation=int, avg_fit=None, **kwargs):
    rate_boundaries = [0.01, 0.5]
    lambda_boundaries = [0, 50]
    if avg_fit is None:
        return self._fixed
    else:
        sigma = np.std(avg_fit)
        params = []
        print(sigma)
        for i in range(self._offspring_size):
            # if avg_fit == float('inf'):
                # return {'rate': rate_boundaries[1], 'lamb': lambda_boundaries[1], 'randomize_mutation': True}
            # else:
            par = self._fixed
            params.append(par)
        return params

def _phased(self, generation=int, **kwargs):
    self._calc_era(generation)
    rate = self._rates[self._current_era]
    randomize = self._randomize[self._current_era]
    lamb = self._calc_lamb(generation)
    return {'rate': rate, 'lamb': lamb, 'randomize_mutation': randomize}

def _calc_evolution_eras(self):
    # expansion / consolidation / specialization / eugeny
    self._eras_duration = [0.15, 0.35, 0.35, 0.15]  # must add up to 1
    self._eras_duration = [0.3, 0.25, 0.30, 0.15]  # must add up to 1
    self._lambda_min = [0, 4, 10, 0]
    self._lambda_max = [0, 4, 100, 0]
    self._rates =  [1, 0.25, 0.35, 0] # note that last phase has no mutation
    self._randomize = [True, True, True, True] # All true since were controling randomization through rate
    self._gen_boundaries = []
    for interval in self._eras_duration:
        try:
            self._gen_boundaries.append(int(round(self._max_gen * interval + self._gen_boundaries[-1], 0)))
        except IndexError:
            self._gen_boundaries.append(int(round(self._max_gen * interval, 0)))

def get_params(self, generation, **kwargs):
    if self._adapt:
        params = self._method(generation, **kwargs)
    else:
        params = self._fixed
    ans = []
    if isinstance(params, dict):
        for i in range(int(self._offspring_size)):
            ans.append(params)
    else:
        ans = params
    return ans

def _calc_era(self, generation):
```

```
        previous = 0
        for i, upper_boundary in enumerate(self._gen_boundaries):
            if previous <= generation and generation <= upper_boundary:
                self._current_era = i
                break

    def _calc_lamb(self, generation):
        lower = self._lambda_min[self._current_era]
        upper = self._lambda_max[self._current_era]
        if self._current_era == 0:
            interval = self._gen_boundaries[self._current_era]
            lamb_line = np.linspace(lower, upper, num=interval+1)
            return lamb_line[generation]
        else:
            interval = self._gen_boundaries[self._current_era] - self._gen_boundaries[self._current_era-1]
            lamb_line = np.linspace(lower, upper, num=interval+1)
            return lamb_line[generation - self._gen_boundaries[self._current_era-1]]
```

```python
# main packages
import numpy as np

# local packages
#N/A

# cointegration modules
from cointegration.optimizer.geneticalgo.specie import Specie


class Crossover():

    def __init__(self, survivors=None, method='one_point'):
        self._method = method
        self._parents_genome_pairs = self._parent_random_selection(survivors)
        # self.parent_2 = parent_2.genome.copy()

    def _crossover_methods(self, *args):
        methods = {'one_point': self._one_point,}
        return methods[self._method](*args)

    def mate(self):
        childs = []
        for genomes in self._parents_genome_pairs:
            childs_genome = self._crossover_methods(genomes[0], genomes[1])
            child_1 = Specie(childs_genome[0])
            child_2 = Specie(childs_genome[1])
            childs.extend([child_1, child_2])
        return childs

    def _parent_random_selection(self, survivors_list):
        survivors = survivors_list.copy()
        parents_pairs_genome = []
        for i in range(int(len(survivors_list)/2)):
            rand_1 = np.random.randint(0, len(survivors))
            parent_1 = survivors.pop(rand_1)
            genome_1 = parent_1.genome.copy()
            rand_2 = np.random.randint(0, len(survivors))
            parent_2 = survivors.pop(rand_2)
            genome_2 = parent_2.genome.copy()
            parents_pairs_genome.append([genome_1, genome_2])
        return parents_pairs_genome

    def _one_point(self, parent_gen_1, parent_gen_2):
        c1, c2 = [], []
        section_point = np.random.randint(1, len(parent_gen_1))
        c1.extend(parent_gen_1[:section_point])
        c1.extend(parent_gen_2[section_point:])
        c2.extend(parent_gen_2[:section_point])
        c2.extend(parent_gen_1[section_point:])
        return c1, c2
```

B.3 PSO

```
# main packages
import numpy as np

# local packages
# N/A

# cointegration modules
from cointegration.optimizer.pso.particle import Particle
from cointegration.optimizer.pso.initiator import ParticlesInitiator
from cointegration.optimizer.pso.params import ParamsInterpreter
from cointegration.optimizer.pso.ese import AdaptiveESE
from cointegration.optimizer.pso.els import AdaptiveELS
from cointegration.optimizer.pso.linear import AdaptiveLinear

class PSOOptimizer():

    def __init__(self, max_iter=100, particles=50, fitness_func=None, params=None, adaptation=None, time_weighted=False,
        ''' fitness function must be object already instanciated, called with methos named 'run' This run must receive as
        arguments teh arguments with the parameters to be optimized. Other custom arguments can be passed as kwargs' '''
        self._max_iter = max_iter
        self._particles_qty = particles
        self._fit_func = fitness_func
        self._adapt_method = adaptation
        self._time_weighted = time_weighted
        self._pinterpreter = ParamsInterpreter(**params)
        self._particles_initialization()
        self._instance_adaptation(adaptation)
        self._gbest_fitness = np.inf
        self._gbest = None

    def _particles_initialization(self):
        inital_coefs = self._initialize_adaptation_coefs()
        initiator = ParticlesInitiator(params_interp=self._pinterpreter, particles=self._particles_qty)
        initiator.get_initial_pos_velo()
        self._swarm = []
        for i in range(self._particles_qty):
            part = Particle(id_=i, params_interp=self._pinterpreter)
            part.set_initial_pos(initiator.get_initial_pos_particle(i))
            part.set_initial_velocity(initiator.get_initial_velocity_particle(i))
            part.update_opt_coefs(inital_coefs)
            self._swarm.append(part)

    def _instance_adaptation(self, adaptation):
        if adaptation is None:
            self._adapt = False
        else:
            self._adapt = True
            if adaptation =='dynamic':
                self._evo_state_history = {}
                self._ese = AdaptiveESE(swarm=self._swarm, dimensions=self._pinterpreter.dimensions, max_iter=self._max_i
                self._els = AdaptiveELS(swarm=self._swarm, params_interp=self._pinterpreter, max_iter=self._max_iter)
            elif adaptation == 'linear':
```

```
            self._linear = AdaptiveLinear(swarm=self._swarm, max_iter=self._max_iter)

    def _initialize_adaptation_coefs(self):
        if self._adapt_method is None:
            coef = {'inertia_w': 0.5, 'personal_coef': 1.5,'social_coef': 1.5}
        elif self._adapt_method =='dynamic':
            coef = AdaptiveESE.get_initial()
        elif self._adapt_method == 'linear':
            coef = AdaptiveLinear.get_initial()
        return coef

    def optimize(self, **fit_func_kwargs):
        for iter_ in range(self._max_iter):
            self._move_particles(iter_=iter_, **fit_func_kwargs)
            self._update_gbest()
            self._register_hist_convergence(iter_)
            if self._adapt:
                self._adapter_manager(iter_, **fit_func_kwargs)

    def _move_particles(self, iter_=None, **fit_func_kwargs):
        for particle in self._swarm:
            if iter_ != 0:
                particle.new_position()
            position_in_original_param_format = self._pinterpreter.convert_array_todict(particle.get_position())
            result = self._fit_func.run(position_in_original_param_format, **fit_func_kwargs)
            particle.update_best_personal(result)
            if result < self._gbest_fitness:
                self._gbest_fitness = result
                self._gbest = particle.get_position()

    def _adapter_manager(self, iter_, **fit_func_kwargs):
        if self._adapt_method =='dynamic':
            self._ese_adapter(iter_)
            self._els_adapter(iter_, **fit_func_kwargs)
        elif self._adapt_method == 'linear':
            self._linear_adapt(iter_)

    def _ese_adapter(self, iter_):
        current_coefs = self._swarm[0].opt_coefs.copy()
        opt_coefs = self._ese.adapt(iter_=iter_,
                                    gbest_fitness=self._gbest_fitness,
                                    coefs=current_coefs)
        self._evo_state = self._ese.evo_state
        self._evo_state_history[iter_] = self._ese.evo_state_index
        for particle in self._swarm:
            particle.update_opt_coefs(coefs=opt_coefs)

    def _els_adapter(self, iter_, **fit_func_kwargs):
        if self._evo_state == 'convergence':
            new_position = self._els.get_gbest_new_position(iter_=iter_)
            position_in_original_param_format = self._pinterpreter.convert_array_todict(new_position)
            new_fitness = self._fit_func.run(position_in_original_param_format, **fit_func_kwargs)
            if new_fitness < self._gbest_fitness:
                self._gbest_fitness = new_fitness
```

```
            self._gbest = new_position
            self._update_gbest()

    def _linear_adapt(self, iter_=None):
        opt_coefs = self._linear.adapt(iter_)
        for particle in self._swarm:
            particle.update_opt_coefs(coefs=opt_coefs)

    def _phased_adapt(self, iter_=None):
        opt_coefs = self._phased.adapt(iter_)
        for particle in self._swarm:
            particle.update_opt_coefs(coefs=opt_coefs)

    def _update_gbest(self):
        # print('\nres: ', self._gbest_fitness)
        # print(self._gbest)

        for particle in self._swarm:
            particle.update_best_social(self._gbest)

    def _register_hist_convergence(self, iteration):
        if iteration == 0:
            self.fitness_history = {'best': [], 'average': []}
        else:
            self.fitness_history['best'].append(self._gbest_fitness)
            sum_ = 0
            for particle in self._swarm:
                if particle.pbest_fitness != np.inf:
                    sum_ += particle.pbest_fitness
            self.fitness_history['average'].append(sum_/len(self._swarm))

    def get_result(self, output='array'):
        if output == 'array':
            return self._gbest_fitness, self._gbest
        elif output == 'dict':
            return self._gbest_fitness, self._pinterpreter.convert_array_todict(self._gbest)
```

```
# main packages
import numpy as np
from scipy.stats.qmc import Sobol

# local packages
# N/A

# cointegration modules
# N/A

class ParticlesInitiator():

    def __init__(self, params_interp=None, particles=None):
        self._pinterp = params_interp
        self._particles_qty = particles
        self._identify_numeric_bounded_variables()

    def _identify_numeric_bounded_variables(self):
        self._valid_float, self._valid_int, self._other_param = [], [], []
        for name, params in  self._pinterp.parameters.items():
            has_lb = not self._pinterp.parameters[name]['lb'] is None
            has_ub = not self._pinterp.parameters[name]['ub'] is None
            if has_lb and has_ub:
                if params['types'] == float:
                    self._valid_float.append(name)
                elif params['types'] == int:
                    self._valid_int.append(name)
            else:
                self._other_param.append(name)

    def get_initial_pos_velo(self):
        self.calc_initial_position()
        self.calc_initial_velocities()

    # # # POSITION
    def calc_initial_position(self):
        init_position = []
        n_bounded = self._draw_sobol_for_numeric_bounded()
        other_random = self._draw_random_for_unbounded()
        for i in range(self._pinterp.dimensions):
            try:
                init_position.append(n_bounded[self._pinterp.get_name(i)])
            except KeyError:
                init_position.append(other_random[self._pinterp.get_name(i)])
        self._initial_position_vector = init_position

    def _draw_sobol_for_numeric_bounded(self):
        num_params = {}
        sobol_matrix = self._build_sobol_matrix()
        for i, name in enumerate(self._numeric_bounded_params):
            num_params[name] = self._rescale_sobol(sobol_matrix, name, i)
        return num_params

    def _build_sobol_matrix(self):
```

```
        self._numeric_bounded_params = self._valid_float
        self._numeric_bounded_params.extend(self._valid_int)
        numeric_bounded_dimension = len(self._numeric_bounded_params)
        sobol_gen = Sobol(d=numeric_bounded_dimension)
        return sobol_gen.random(self._particles_qty)

    def _rescale_sobol(self, matrix, name, i):
        self._lb = self._pinterp.parameters[name]['lb']
        self._ub = self._pinterp.parameters[name]['ub']
        vec = matrix[:, i]
        rescaled = list(map(self._min_max, vec))
        return np.array(rescaled)

    def _min_max(self, x):
        return self._lb + ( (x - 0) * (self._ub - self._lb) / 1)

    def _draw_random_for_unbounded(self):
        other = {}
        for name in self._other_param:
            other[name] = self._random_initializer(name)
        return other

    def _random_initializer(self, name):
        values = []
        for i in range(self._particles_qty):
            values.append(self._pinterp.new_random()[name])
        return np.array(values)

    def get_initial_pos_particle(self, particle_index):
        particle_pos = []
        for d in range(self._pinterp.dimensions):
            particle_pos.append(self._initial_position_vector[d][particle_index])
        return np.array(particle_pos)

    # # # VELOCITY
    def calc_initial_velocities(self):
        init_velo = []
        random_pos = self._draw_random()
        velocities = self._half_diff(random_pos)
        for i in range(self._pinterp.dimensions):
            init_velo.append(velocities[self._pinterp.get_name(i)])
        self._initial_velocity = init_velo

    def _draw_random(self):
        random_pos = {}
        for name in self._pinterp.param_names:
            random_pos[name] = self._random_initializer(name)
        return random_pos

    def _half_diff(self, random_pos):
        velo = {}
        for name in random_pos.keys():
            col = self._pinterp.get_index(name)
            # velo.append(1/2* (random_vec[d] - self._initial_position_vector[d]))
```

```
            velo[name] = 1/2 * (random_pos[name] - self._initial_position_vector[col])
        return velo

    def get_initial_velocity_particle(self, particle_index):
        particle_pos = []
        for d in range(self._pinterp.dimensions):
            particle_pos.append(self._initial_velocity[d][particle_index])
        return np.array(particle_pos)
```

```
# main packages
import numpy as np

# local packages
# N/A

# cointegration modules
from cointegration.optimizer.pso.repair import Repairer

class Particle():

    def __init__(self, id_=None, params_interp=None):
        self._id = id_
        self.pbest_fitness = np.inf
        self._dimensions = params_interp.dimensions
        ### Alternatives for constrain
        # self._constrain_repair = Repairer(position_method='random_shrink', velocity_method='zero', params_interp=params_
        # self._constrain_repair = Repairer(position_method='reflect', velocity_method='zero', params_interp=params_interp
        # self._constrain_repair = Repairer(position_method='random_method', velocity_method='zero', params_interp=params_
        self._constrain_repair = Repairer(position_method='shrink', velocity_method='zero', params_interp=params_interp)

    def update_opt_coefs(self, coefs):
        self.opt_coefs = {}
        self.opt_coefs['inertia_w'] = coefs['inertia_w']
        self.opt_coefs['personal_coef'] = coefs['personal_coef']
        self.opt_coefs['social_coef'] = coefs['social_coef']

    def set_initial_pos(self, init_pos):
        self.pos = init_pos
        self.pbest = init_pos
        self.gbest = init_pos

    def set_initial_velocity(self, init_velo):
        self._curr_velocity = init_velo

    def _new_velocity(self):
        inertia = self.opt_coefs['inertia_w'] * self._curr_velocity
        cognitive = self.opt_coefs['personal_coef'] * np.random.uniform(0,1) * (self.pbest - self.pos)
        social = self.opt_coefs['social_coef'] * np.random.uniform(0,1) * (self.gbest - self.pos)
        self._curr_velocity = inertia + cognitive + social
        return self._curr_velocity

    def new_position(self):
        new_pos = self.pos + self._new_velocity()
        if self._constrain_repair.evaluate(new_pos):
            self.pos = new_pos
        else:
            self.pos, self._curr_velocity = self._constrain_repair.repair(new_pos, self._curr_velocity)
        return self.pos

    def update_best_personal(self, new_fitness):
        self.current_fitness = new_fitness
        if new_fitness < self.pbest_fitness:
            self.pbest_fitness = new_fitness
```

```
            self.pbest = self.get_position()

    def update_best_social(self, gbest):
        self.gbest = np.array(gbest)

    def get_position(self):
        return self.pos
```

```
# main packages
import numpy as np

# local packages
# N/A

# cointegration modules
# N/A

class AdaptiveELS():
    _elitist_learnr_max = 1
    _elitist_learnr_min = 0.01

    def __init__(self, swarm=None, dimensions=None, params_interp=None, max_iter=None):
        self._swarm = swarm
        self._max_iter = max_iter
        self._pinterpreter = params_interp
        self._dimensions = params_interp.dimensions

    def get_gbest_new_position(self, iter_=None):
        gbest_position = self._swarm[0].gbest.copy()
        new_value, dimension = self._nudge_one_dimension(gbest_position, iter_)
        gbest_position[dimension] = self._enforce_boundaries(new_value, dimension)
        return gbest_position

    def _get_worst_positioned(self):
        worst_fitness = -np.inf
        worst_index = None
        for i, particle in enumerate(self._swarm):
            if particle.current_fitness > worst_fitness:
                worst_index = i
                worst_fitness = particle.current_fitness
        return worst_index

    def _nudge_one_dimension(self, position, iter_):
        dim = np.random.randint(0, self._dimensions)
        lb = self._pinterpreter.lb(dim)
        ub = self._pinterpreter.ub(dim)
        elitist_rate = self._calc_elitist_learning_rate(iter_)
        perturbation = np.random.normal(loc=0, scale=elitist_rate)
        return position[dim] + (ub-lb) * perturbation, dim

    def _calc_elitist_learning_rate(self, iter_):
        epoch = iter_ / self._max_iter
        return self._elitist_learnr_max - epoch * (self._elitist_learnr_max-self._elitist_learnr_min)

    def _enforce_boundaries(self, new_value, dimension):
        if new_value < self._pinterpreter.lb(dimension):
            new_value = self._pinterpreter.lb(dimension)
        elif new_value > self._pinterpreter.ub(dimension):
            new_value = self._pinterpreter.ub(dimension)
        return new_value

# main packages
```

```
import numpy as np

# local packages
# N/A

# cointegration modules
# N/A

class EvolutionaryState():
    _exploration_max = 0.4
    _convergence_min = 0

    def __init__(self, swarm=None, dimensions=None, time_weighted=False, max_iter=None):
        self._dimensions = dimensions
        self._swarm = swarm
        self._max_iter = max_iter
        self._time_weighted = time_weighted

    def get_state(self, gbest_fitness, iter_):
        self._gbest_fitness = gbest_fitness
        self.factor = self._get_factor()
        if self._time_weighted:
            self.factor = self._time_weighted_factor(iter_)
        return self._fuzzy_classifier()

    def _get_factor(self):
        distances = self._calc_distance()
        best_index = self._get_best_positioned()
        dg = distances[best_index]
        dmax = max(distances)
        dmin = min(distances)
        if (dmax - dmin) == 0:
            return 0
        else:
            return (dg - dmin) / (dmax - dmin)

    def _time_weighted_factor(self, iter_):
        phase = iter_ / self._max_iter
        hyper = (np.tanh(8*phase-2.5)+1) /2
        time_dep_factor = self._linear(iter_)
        weighted_factor = time_dep_factor * (1 - hyper) + self.factor * hyper
        return weighted_factor

    def _linear(self, iter_):
        phase = 1 - iter_ / self._max_iter
        return self._convergence_min + (self._exploration_max - self._convergence_min) * phase

    def _calc_distance(self):
        # improve speed with cuda
        self.distances = []
        for i_particle in self._swarm:
            i_part_distances = 0
            for j_particle in self._swarm:
                if id(i_particle) == id(j_particle):
```

```
                continue
            else:
                for k in range(self._dimensions):
                    i_k_pos = i_particle.get_position()[k]
                    j_k_pos = j_particle.get_position()[k]
                    i_part_distances += (i_k_pos - j_k_pos)**2
        d_i = (1 / (len(self._swarm) -1)) * np.sqrt(i_part_distances)
        self.distances.append(d_i)
    return self.distances

def _get_best_positioned(self):
    best_fitness = np.inf
    best_index = None
    for i, particle in enumerate(self._swarm):
        if particle.current_fitness < best_fitness:
            best_index = i
            best_fitness = particle.current_fitness
    return best_index

def _fuzzy_classifier(self):
    classes = {'exploration': 1, 'exploitation':2, 'convergence': 3, 'jumping-out': 4}
    index_classes = {1: 'exploration', 2: 'exploitation', 3: 'convergence' , 4: 'jumping-out'}
    mu_func = {'exploration': self._mu_exploration, 'exploitation': self._mu_exploitation, 'convergence': self._mu_co
    num, den = 0, 0
    for phase, value in classes.items():
        num += mu_func[phase](self.factor) * classes[phase]
        den += mu_func[phase](self.factor)
    self.evo_state_index = int(num/den)
    return index_classes[self.evo_state_index]

def _mu_convergence(self, factor):
    if factor <= 0.1:
        return 1
    elif factor <= 0.3:
        return -5 * factor + 1.5
    else:
        return 0

def _mu_exploitation(self, factor):
    if factor <= 0.2:
        return 0
    elif factor <= 0.3:
        return 10 * factor - 2
    elif factor <= 0.4:
        return 1
    elif factor <= 0.6:
        return -5 * factor +3
    else:
        return 0

def _mu_exploration(self, factor):
    if factor <= 0.4:
        return 0
    elif factor <= 0.6:
```

```
            return 5 * factor - 2
        elif factor <= 0.7:
            return 1
        elif factor <= 0.8:
            return -10 * factor + 8
        else:
            return 0

    def _mu_jumping(self, factor):
        if factor <= 0.7:
            return 0
        elif factor <= 0.9:
            return 5 * factor - 3.5
        else:
            return 1


class AccelerationCoefs():
    _coefs_initialization = 2
    _inertia_initialization = 0.9
    _min_value_acc = 1.5
    _low = 0.01
    _mid = 0.05
    _upp = 0.1
    _chg_size = {'jumping-out': {'personal_coef': [-_mid, -_upp], 'social_coef': [_mid, _upp]},
                 'exploration': {'personal_coef': [_mid, _upp], 'social_coef': [-_mid, -_upp]},
                 'exploitation': {'personal_coef': [_low, _mid], 'social_coef': [-_low, -_mid]},
                 'convergence': {'personal_coef': [_low, _mid], 'social_coef': [_low, _mid]}}

    @classmethod
    def get_initial(cls):
        initial = {'inertia_w': cls._inertia_initialization,
                   'personal_coef': cls._coefs_initialization,
                   'social_coef': cls._coefs_initialization}
        return initial

    @classmethod
    def adjust_coefs(cls, evo_state=None, evo_factor=None, older_coefs=None):
        new_coef = {}
        new_coef['inertia_w'] = cls._weight_adapter(evo_factor)
        new_coef['personal_coef'] = cls._coef_change(evo_state=evo_state, coef='personal_coef') + older_coefs['personal_c
        new_coef['social_coef'] = cls._coef_change(evo_state=evo_state, coef='social_coef') + older_coefs['social_coef']
        new_coef = cls._enforce_coefs_bounds(new_coef)
        return new_coef

    @classmethod
    def _enforce_coefs_bounds(cls, new_coef):
        if new_coef['personal_coef'] < cls._min_value_acc:
            new_coef['personal_coef'] = cls._min_value_acc
        if new_coef['social_coef'] < cls._min_value_acc:
            new_coef['social_coef'] = cls._min_value_acc
        if (new_coef['personal_coef']+new_coef['social_coef']) > 4.0:
            new_coef = cls._renormalize(new_coef)
        return new_coef
```

```
    @classmethod
    def _coef_change(cls, evo_state=None, coef=None):
        lower = cls._chg_size[evo_state][coef][0]
        upper = cls._chg_size[evo_state][coef][1]
        return np.random.uniform(low=lower, high=upper)

    @classmethod
    def _renormalize(cls, new_coef):
        new_coef['personal_coef'] = 4* (new_coef['personal_coef'] / (new_coef['personal_coef'] + new_coef['social_coef']))
        new_coef['social_coef'] = 4* (new_coef['social_coef'] / (new_coef['personal_coef'] + new_coef['social_coef']))
        return new_coef

    @classmethod
    def _weight_adapter(cls, evo_factor):
        return 1 / (1 + 1.5 * np.e**(-2.6 * evo_factor) )


class MixedAdaptation():
    _exploration_max = 0.65

    @classmethod
    def adjust_factor(cls, factor, iter_, max_iter):
        time_dep_factor = cls._exploration_max * (1 - iter_ / max_iter)
        weighted_factor = (time_dep_factor + factor)/2
        return weighted_factor


class AdaptiveESE():

    def __init__(self, swarm=None, dimensions=None, max_iter=None, adapt=True, time_weighted=False):
        self._swarm = swarm
        self._dimensions = dimensions
        self._adapt = adapt
        self._max_iter = max_iter
        self._time_weighted = time_weighted
        self.coef_history = {}

    @classmethod
    def get_initial(cls):
        return AccelerationCoefs.get_initial()

    def adapt(self, iter_=None, gbest_fitness=None, coefs=None):
        if self._adapt:
            evo = EvolutionaryState(swarm=self._swarm, dimensions=self._dimensions, max_iter=self._max_iter, time_weighte
            self.evo_state = evo.get_state(gbest_fitness, iter_)
            self.evo_factor = evo.factor
            self.evo_state_index = evo.evo_state_index
            self.coef_history[iter_] = AccelerationCoefs.adjust_coefs(evo_state=self.evo_state, evo_factor=evo.factor, ol
            return self.coef_history[iter_]
        else:
            return coefs
```

```python
# main packages
import numpy as np
import random

# local packages
# N/A

# cointegration modules

class ParamsInterpreter():
    '''Parameter interpreter is an objcted to extract and deal with the optmization
    parameters. It must receive a dict the parameters to be optmized using the
    following structure:
    params = {
        types = {<param_name> : python built-in type},
            --> informs what is the type of the param; Can be int, float, list or boolean
        >>>IF type is float or integer:
            ub = {<param_name> : value},
                --> informs the UPPER BOUNDARY.
            lb = {<param_name> : value},
                --> informs the LOWER BOUNDARY.
            step = {<param_name> : value}, [optional]
                --> informs the step of allowed change in the value.
            invalid =  {<param_name> : [value]}, [optional]
                --> informs a list, with a set of values that are NOT allowed to exist;
        >>>IF type is set:
            goup =  {<param_name> : [value]},
                --> informs a list, with a set of values that are allowed to exist; only possible with list type
        >>>IF type is boolean:
            No additional info must be passed.
        IMPORTANT: parameter names (keys) must be identical across different dictionaries
        }'''

    def __init__(self, **kwargs):
        '''Can pass a single dictionary, or many kwords as desired'''
        self._register_params(**kwargs)

    def _register_params(self, types=None, ub=None, lb=None, step=None, invalid=None, group=None, boolean=None):
        self.parameters = {}
        for name in types.keys():
            self.parameters[name] = {'types': None, 'ub': None, 'lb': None, 'step': None, 'invalid': [], 'boolean': None}
        self._add_parameter(types, 'types')
        self._add_parameter(ub, 'ub')
        self._add_parameter(lb, 'lb')
        self._add_parameter(step, 'step')
        self._add_parameter(invalid, 'invalid')
        self._add_parameter(group, 'group')
        self._build_ref()
        self.param_names = list(self.parameters.keys())
        self.dimensions = len(self.parameters)

    def _add_parameter(self, dict_, data):
        if dict_ is None:
            return None
```

```
        for name, value in dict_.items():
            self.parameters[name][data] = value

    def _build_ref(self):
        self.params_id = {'index':{}, 'name': {}}
        for i, name in enumerate(self.parameters.keys()):
            self.params_id['index'][i] = name
            self.params_id['name'][name] = i

    def _get_ref(self, value, asname=False):
        if isinstance(value, str):
            if asname:
                return value
            else:
                return self.get_index(value)
        elif isinstance(value, int):
            return self.get_name(value)

    def get_name(self, index):
        return self.params_id['index'][index]

    def get_index(self, name):
        return self.params_id['name'][name]

    def convert_array_todict(self, params):
        pos = {}
        for i, key in self.params_id['index'].items():
            pos[key] = params[i]
        return pos

    def convert_dict_toarray(self, params):
        pos = []
        for i, (key, value) in enumerate(params.items()):
            pos.append(value)
        return np.array(pos)

    # # # RANDOM PARAMETER VALUE GENERATION
    def new_random(self, output='dict'):
        '''Generates and entry of random values for the parameters, obeying restrictions'''
        new = {}
        for name in self.param_names:
            new[name] = {}
            if self.parameters[name]['types']==int:
                new[name] = self._random_int(self.parameters[name])
            elif self.parameters[name]['types']==float:
                new[name] = self._random_float(self.parameters[name])
            elif self.parameters[name]['types']==bool:
                new[name] = self._random_bool(self.parameters[name])
            elif self.parameters[name]['types']==list:
                new[name] = self._random_list(self.parameters[name])
        if output == 'dict':
            return new
        elif output == 'array':
            return self.convert_dict_toarray(new)
```

```
@staticmethod
def _random_int(param):
    if param['step'] is None:
        param['step'] = 1
    if len(param['invalid'])==0:
        num = random.randrange(start=param['lb'], stop=param['ub'], step=param['step'])
    else:
        while True:
            num = random.randrange(start=param['lb'], stop=param['ub'], step=param['step'])
            if not num in param['invalid']:
                break
    return num

@staticmethod
def _random_float(param):
    if len(param['invalid'])==0:
        num = random.uniform(param['lb'], param['ub'])
    else:
        while True:
            num = random.uniform(param['lb'], param['ub'])
            if not num in param['invalid']:
                break
    return num

@staticmethod
def _random_bool(param):
    num = random.randint(0, 1)
    return bool(num)

@staticmethod
def _random_list(param):
    index = random.randint(0, len(param['group'])-1)
    return param['group'][index]

# # # GETTERS AND SETTERS
def ub(self, value):
    name = self._get_ref(value, asname=True)
    try:
        ans = self.parameters[name]['ub']
        if ans is None:
            ans = np.inf
    except KeyError:
        ans = np.inf
    return ans

def lb(self, value):
    name = self._get_ref(value, asname=True)
    try:
        ans = self.parameters[name]['lb']
        if ans is None:
            ans = -np.inf
    except KeyError:
        ans = -np.inf
```

```
        return ans

    def step(self, value):
        print('Not implemented')

    def invalid(self, value):
        print('Not implemented')

    def group(self, value):
        name = self._get_ref(value, asname=True)
        ans = self.parameters[name]['group']
        return ans



# main packages
# N/A
import numpy as np

# local packages
# N/A

# cointegration modules
# N/A

class BaseRepair():

    def evaluate(self, position):
        self.index_outside_search_space = []
        for i, param in enumerate(position):
            ub = param <= self._pinterp.ub(i)
            lb = param >= self._pinterp.lb(i)
            if all([lb, ub]):
               continue
            else:
                self.index_outside_search_space.append(i)
        return not bool(self.index_outside_search_space)


class PositionRepair(BaseRepair):

    def _load_pos_methods(self, method):
        pos_methods = {'reflect': self._reflect, 'shrink': self._shrink, 'random_shrink': self._random_shrink, 'random_me
        self._pos_method = pos_methods[method]

    def _repair_pos(self, position):
        while True:
            repaired_pos = self._pos_method(position)
            if self.evaluate(repaired_pos):
                break
        return repaired_pos

    def _reflect(self, position):
        for i in self.index_outside_search_space:
```

```
            if position[i] > self._pinterp.ub(i):
                position[i] = 2 * self._pinterp.ub(i) - position[i]
            elif position[i] < self._pinterp.lb(i):
                position[i] = 2 * self._pinterp.lb(i) - position[i]
            else:
                print('WARNGIN DEV: CASO NAO TRATADO')
        return position

    def _shrink(self, position):
        for i in self.index_outside_search_space:
            if position[i] > self._pinterp.ub(i):
                position[i] = self._pinterp.ub(i)
            elif position[i] < self._pinterp.lb(i):
                position[i] = self._pinterp.lb(i)
            else:
                print('WARNGIN DEV: CASO NAO TRATADO')
        return position

    def _random_shrink(self, position):
        for i in self.index_outside_search_space:
            if position[i] > self._pinterp.ub(i):
                position[i] = self._pinterp.ub(i) * (1- np.random.uniform(0, 0.2))
            elif position[i] < self._pinterp.lb(i):
                position[i] = self._pinterp.lb(i) * (1- np.random.uniform(0, 0.2))
            else:
                print('WARNGIN DEV: CASO NAO TRATADO')
        return position

    def _random_method(self, position):
        methods = {0: self._shrink, 1: self._random_shrink}
        rand_method = np.random.randint(0, 2)
        return methods[rand_method](position)


class VelocityRepair(BaseRepair):

    def _load_velo_methods(self, method):
        velo_methods = {'zero': self._zero, 'half_invert': self._half_invert}
        self._velo_method = velo_methods[method]

    def _repair_velo(self, velocity):
        repaired_velo = self._velo_method(velocity)
        return repaired_velo

    def _zero(self, velocity):
        for i in self.index_outside_search_space:
            velocity[i] = 0
        return velocity

    def _half_invert(self, velocity):
        for i in self.index_outside_search_space:
            velocity[i] = velocity[i] * 1/2
        return velocity
```

```
class Repairer(PositionRepair, VelocityRepair):

    def __init__(self, position_method=None, velocity_method=None, params_interp=None):
        self._pinterp = params_interp
        self._load_methods(position_method, velocity_method)

    def _load_methods(self, position_method, velocity_method):
        self._load_pos_methods(position_method)
        self._load_velo_methods(velocity_method)

    def repair(self, position, velocity):
        rep_velocity = self._repair_velo(velocity) # must come before adjusting position
        rep_position = self._repair_pos(position)
        return rep_position, rep_velocity
```