

econcomplex

Documentação Técnica — Versão 1.0.1

Biblioteca Python para Indicadores de
Complexidade Econômica e Ciência Regional

Elton Freitas

Dependências: `numpy` ≥ 1.21 , `pandas` ≥ 1.3 , `scipy` ≥ 1.9
Compatível com Python 3.9+ (NumPy 1.x e 2.x)

Junho de 2026

Sumário

1	Visão Geral	3
1.1	Instalação	3
1.2	Estrutura de Módulos	4
1.3	Mapa da API: Portas de Entrada, Implementações Avançadas e Aliases	4
2	Organização dos Dados	5
2.1	Formato Padrão: Tabela Longa (<i>long format</i>)	5
2.2	Regras de Qualidade dos Dados	5
2.3	Nomes das Colunas — Dicionário <code>cols</code>	6
2.4	Formatos de Arquivo Suportados	6
2.5	Exemplos de Bases Compatíveis	7
3	Como Executar os Scripts	7
3.1	Pré-requisitos	7
3.2	Execução Direta via Terminal	8
3.3	Execução em Jupyter Notebook / JupyterLab	8
3.4	Execução no VS Code / PyCharm / Spyder	9
3.5	Script Mínimo de Execução	9
3.6	Execução com Dados da RAIS / CNAE (caso de uso real)	10
3.7	Execução em Painel Multitemporal	11
4	Indicadores do Núcleo (<code>core</code>)	11
4.1	Vantagem Comparativa Revelada (RCA)	11
4.2	Diversidade e Ubiquidade	12
5	Complexidade Econômica (<code>complexity</code>)	12
5.1	Método do Vetor Próprio (Hidalgo & Hausmann, 2009)	12
5.2	Método de Reflexões (Hidalgo & Hausmann, 2009)	12
5.3	Fitness-Complexity (Tacchella et al., 2012)	12
5.4	ECI Subnacional	13
6	Product Space e Relatedness (<code>relatedness</code>)	13
6.1	Proximidade (ϕ)	13
6.2	Densidade e Distância	13
6.3	Co-ocorrência e Índices de Relatedness	13
6.4	Cross-Space (relatedness entre classificações distintas)	13
7	Especialização Regional (<code>specialization</code>)	14
8	Desigualdade e Concentração (<code>inequality</code>)	14
9	Produtividade (<code>productivity</code>)	14
10	Complexidade Baseada em Patentes (<code>patents</code>)	14
11	Dinâmica Temporal (<code>dynamics</code>)	14
12	Complexity Outlook (<code>outlook</code>)	15

13 Otimização de Complexidade (<code>optimization</code>)	15
14 Difusão Estratégica (<code>optimization.diffusion</code>)	15
15 Pipeline Unificado (<code>compute_complexity</code>)	16
15.1 Assinatura da Função	16
15.2 Parâmetros	17
16 Como Interpretar os Indicadores	17
17 Exemplos Completos	18
17.1 Exemplo 1 — Análise Regional de Empregos	18
17.2 Exemplo 2 — Exportações com Dados COMEX	19
17.3 Exemplo 3 — Painel Temporal RAIS (2010–2022)	19
17.4 Exemplo 4 — Alvos de Diversificação (Otimização de ECI)	20
18 Referência da API	21
18.1 <code>core</code> — Core	21
18.2 <code>complexity</code> — Complexidade	22
18.3 <code>relatedness</code> — Relatedness e Product Space	22
18.4 <code>specialization</code> — Especialização Regional	23
18.5 <code>inequality</code> — Desigualdade e Concentração	23
18.6 <code>productivity</code> — Produtividade	24
18.7 <code>patents</code> — Complexidade de Patentes	24
18.8 <code>dynamics</code> — Dinâmica Temporal	24
18.9 <code>outlook</code> — Complexity Outlook	25
18.10 <code>optimization</code> — Otimização de ECI e Difusão Estratégica	25
18.11 <code>pipeline</code> — Pipeline Unificado	26
19 Referências Bibliográficas	26
20 Licença e Citação	27

Visão Geral

econcomplex é uma biblioteca Python que consolida as melhores implementações de indicadores de Complexidade Econômica e de Ciência Regional, reunindo funcionalidades de quatro pacotes de referência na literatura:

- **EconGeo** (Balland, 2017) — pacote R com foco em indicadores geográficos e de especialização regional
- **economiccomplexity** (Vargas Sepúlveda, 2020) — pacote R com backend C++ (Armadillo) para indicadores de complexidade
- **py-econcomplexity** (The Growth Lab at Harvard, 2020) — pipeline completo em Python com suporte a séries temporais
- **py-economic-complexity** — implementação modular em Python com suporte a Polars e indicadores de cross-space

Instalação

```
1 # Do PyPI (recomendado)
2 pip install econcomplex
3
4 # Versao de desenvolvimento mais recente, direto do GitHub
5 pip install git+https://github.com/eltonfreitas/econcomplex.git
6
7 # Ou, para desenvolvimento local, a partir da raiz da biblioteca
8 pip install -e .
9
10 # Verificar a instalacao
11 python3 -c "import econcomplex as ec; print(ec.__version__)"
```

Listing 1: Instalação via pip

Estrutura de Módulos

Módulo	Conteúdo
core	RCA, RPOP, Mcp, diversidade, ubiquidade, trim_core, make_sample_data, utilitários
complexity	ECI/PCI — porta de entrada única eci_pci(method=...) (eigenvector, reflections, fitness), ECI subnacional
relatedness	Proximidade (discreta, contínua, cosseno, correlação), densidade, distância, co-ocorrência, cross-space
specialization	LQ, Hachman, Krugman, coef. especialização, similaridade
inequality	Gini, Gini locacional, Hoover-Gini, HHI, entropia Shannon
productivity	PRODY, EXPY, PGI, PEII
patents	Ease of Recombination, Modular Complexity
dynamics	Taxas de crescimento, entrada/saída (APIs de matriz e painel)
outlook	COI, COG (Complexity Outlook)
optimization	Otimização de ECI (Stojkoski & Hidalgo 2026), metas de crescimento, difusão estratégica (Alshamsi et al. 2018)
pipeline	Função compute_complexity() — pipeline unificado

Mapa da API: Portas de Entrada, Implementações Avançadas e Aliases

A API pública tem três camadas. As **portas de entrada** são o que a maioria dos usuários precisa; as **implementações avançadas** são as funções específicas de cada método, para as quais as portas delegam (expostas para uso em pesquisa); os **aliases** são nomes curtos alternativos ligados à mesma função (sem código nem cálculo duplicado).

Porta de entrada	Delega para (avançadas)
eci_pci(method=...)	eci_pci_eigenvector, method_of_reflections, fitness_complexity (variantes: mor_region, mor_activities)
proximity(continuous=..., continuous_method=...)	continuous_proximity; cosine_proximity, correlation_proximity
compute_complexity()	pipeline completo por período sobre eci_pci, proximity, density, COI/COG

Alias	Função canônica (mesmo objeto)
density, relatedness	relatedness_density
hhi	herfindahl
coi / cog	complexity_outlook_index / _gain
pgi / peii	product_gini_index / product_emissions_index
spec_coefficient	specialization_coefficient
cross_space_proximity	cross_proximity
location_quotient	rca (fórmula idêntica)

Organização dos Dados

Esta seção descreve o formato de entrada esperado pela biblioteca, garantindo que os dados estejam corretamente estruturados antes da execução de qualquer indicador.

Formato Padrão: Tabela Longa (*long format*)

A biblioteca **econcomplex** trabalha exclusivamente com **tabelas no formato longo** (*tidy data*), onde cada linha representa **uma combinação única** de região × atividade (e período, quando houver):

regiao	atividade	valor	ano (opcional)
SP	cnae_10	12 345	2022
SP	cnae_25	6 789	2022
RJ	cnae_10	9 012	2022
RJ	cnae_25	3 456	2022
SP	cnae_10	13 200	2023
⋮	⋮	⋮	⋮

Atenção

A biblioteca **não** aceita matrizes pivotadas (regiões nas linhas, atividades nas colunas). Use `pivot_to_matrix()` apenas para funções de baixo nível que exijam a matriz explicitamente.

Regras de Qualidade dos Dados

1. **Sem linhas duplicadas:** cada par (*regiao*, *atividade*) deve aparecer exatamente uma vez por período.
2. **Valores não-negativos:** a coluna de valor deve conter apenas números ≥ 0 .
3. **Sem NaN:** remova ou impute valores ausentes antes de passar o DataFrame.
4. **Granularidade consistente:** todas as regiões devem usar o mesmo nível geográfico (ex.: todos municípios *ou* todos estados).

5. **Atividades homogêneas:** use uma única classificação por análise (ex.: CNAE 2 dígitos ou CNAE 4 dígitos, não misturados).

Nomes das Colunas — Dicionário *cols*

O pipeline usa um dicionário *cols* para mapear os nomes reais das colunas do seu DataFrame:

Chave	Tipo	Descrição
"loc"	obrigatória	Coluna de região / localização
"act"	obrigatória	Coluna de atividade / produto / tecnologia
"val"	obrigatória	Coluna de valor numérico (empregos, exportações ...)
"time"	opcional	Coluna de período (ano, trimestre ...)

```

1 import pandas as pd
2 import econcomplex as ec
3
4 df = pd.read_csv("meus_dados.csv")
5
6 # Verificar estrutura
7 print(df.dtypes)
8 print(df.shape)
9 print(df.isnull().sum())          # checar NaN
10 print(df.duplicated().sum())      # checar duplicatas
11
12 # Limpeza basica
13 df = df.dropna(subset=["regiao", "cnae", "empregos"])
14 df = df[df["empregos"] >= 0]      # remover valores negativos
15 df = df.drop_duplicates(subset=["regiao", "cnae", "ano"])
16
17 # Rodar o pipeline
18 resultado = ec.compute_complexity(
19     df,
20     cols={"loc": "regiao", "act": "cnae",
21           "val": "empregos", "time": "ano"},
22 )

```

Listing 2: Verificação e preparação do DataFrame antes do pipeline

Formatos de Arquivo Suportados

Formato	Leitura pandas	Observação
.csv	pd.read_csv()	Mais comum; use <code>sep=";"</code> para CSVs BR
.parquet	pd.read_parquet()	Ideal para bases grandes (RAIS, COMEX)
.xlsx	pd.read_excel()	Requer <code>openpyxl</code> ; evitar >100 000 linhas
.feather	pd.read_feather()	Alta performance; requer <code>pyarrow</code>
.dta	pd.read_stata()	Arquivos Stata; preserva labels

```

1 import pandas as pd
2
3 # CSV com separador ponto-e-virgula (padrao brasileiro)

```

```
4 df = pd.read_csv("rais.csv", sep=";", encoding="latin-1",
5                  usecols=["municipio", "cnae2d", "ano", "vinculos"],
6                  dtype={"municipio": str, "cnae2d": str})
7
8 # Parquet (preferencial para RAIS/COMEX)
9 df = pd.read_parquet("comex_ncm.parquet",
10                     columns=["co_mun", "co_ncm", "ano", "vl_fob"])
11
12 # Excel
13 df = pd.read_excel("dados.xlsx", sheet_name="Sheet1",
14                   usecols=["uf", "setor", "ano", "empregos"])
```

Listing 3: Leitura de diferentes formatos de arquivo

Exemplos de Bases Compatíveis

RAIS/MTE

Empregos formais por município × CNAE (2 ou 4 dígitos) × ano. Coluna de valor: vínculos ativos em 31/12.

COMEX/MDIC

Exportações por município × NCM × ano. Coluna de valor: VL_FOB (valor em US\$).

INPI — Patentes

Pedidos de patente por Unidade da Federação × classificação IPC × ano.

Atlas da Complexidade (Harvard)

Exportações por país × produto (HS) × ano. Disponível em <https://atlas.cid.harvard.edu/>.

Dados simulados (testes)

Use `ec.make_sample_data(n_locs=20, n_acts=50, seed=42)` para gerar um DataFrame sintético.

Como Executar os Scripts

Esta seção descreve todas as formas de executar a biblioteca **econcomplex**, desde a execução direta de scripts Python até o uso em ambientes interativos como Jupyter e IDEs.

Pré-requisitos

Atenção

Certifique-se de que as dependências estão instaladas antes de executar qualquer script.

```
1 # Instalar a biblioteca (dependencias vem automaticamente:
2 # numpy>=1.21, pandas>=1.3, scipy>=1.9)
3 pip install econcomplex
4
5 # Ou em modo de desenvolvimento
6 pip install -e /caminho/para/econcomplex
7
8 # Verificar instalacao
9 python3 -c "import econcomplex as ec; print(ec.__version__)"
```

Listing 4: Instalação das dependências

Execução Direta via Terminal

A forma mais simples de usar a biblioteca é executar um script `.py` diretamente no terminal:

```
1 # Navegue ate a pasta da biblioteca
2 cd /caminho/para/econcomplex
3
4 # Executar o script de exemplo completo (tour por todos os indicadores)
5 python3 examples/basic_usage.py
6
7 # Executar um script proprio
8 python3 meu_script.py
```

Listing 5: Execução de scripts no terminal

Caso a biblioteca **não** esteja instalada via `pip`, adicione o caminho manualmente no início de cada script:

```
1 import sys
2 sys.path.insert(0, "/caminho/para/econcomplex")
3
4 import econcomplex as ec
5 print(ec.__version__) # 1.0.1
```

Listing 6: Adicionar caminho ao sys.path (sem pip install)

Execução em Jupyter Notebook / JupyterLab

```
1 cd /caminho/para/meu_projeto
2 jupyter notebook
3 # ou
4 jupyter lab
```

Listing 7: Abrir Jupyter a partir da pasta do seu projeto

Com a biblioteca instalada via `pip`, a primeira célula é apenas os imports (use o fallback de `sys.path` somente se não a tiver instalado):

```
1 # Celula 1 -- imports
2 import econcomplex as ec
3 import numpy as np
4 import pandas as pd
5
6 print("econcomplex", ec.__version__, "carregado com sucesso.")
```

Listing 8: Célula de configuração no Jupyter

```
1 # Celula 2 -- carregar dados
2 df = pd.read_csv("meus_dados.csv") # colunas: regioao, setor, empregos
3
4 # Celula 3 -- calcular complexidade
5 resultado = ec.compute_complexity(
6     df,
```

```

7     cols={"loc": "regiao", "act": "setor", "val": "empregos"},
8     method="eigenvector",
9     compute_coi_cog=True,
10 )
11 resultado.head()
12
13 # Celula 4 -- visualizar ECI por regiao
14 eci = (resultado[["regiao", "eci"]]
15         .drop_duplicates()
16         .set_index("regiao")
17         .sort_values("eci", ascending=False))
18 print(eci)

```

Listing 9: Células subsequentes — uso normal

Execução no VS Code / PyCharm / Spyder

Em qualquer IDE, configure o **interpretador Python** e o **diretório de trabalho** apontando para a pasta de scripts.

```

1 // .vscode/settings.json
2 {
3     "python.defaultInterpreterPath":
4         "/Library/Frameworks/Python.framework/Versions/3.12/bin/python3",
5     "terminal.integrated.cwd": "${workspaceFolder}"
6 }

```

Listing 10: Configurar caminho no VS Code (settings.json)

No **Spyder**, vá em *Tools* → *PYTHONPATH Manager* e adicione o diretório `econcomplex/`.

Script Mínimo de Execução

Template pronto para uso com qualquer base em formato CSV:

```

1 """
2 Template de uso da biblioteca econcomplex.
3 Adapte os nomes das colunas e o caminho do arquivo CSV.
4 """
5 import pandas as pd
6 import econcomplex as ec
7
8 # 1. Carregar dados
9 df = pd.read_csv("dados.csv")
10 print("Colunas:", df.columns.tolist())
11 print("Shape:", df.shape)
12
13 # 2. Calcular todos os indicadores
14 resultado = ec.compute_complexity(
15     df,
16     cols={
17         "loc": "nome_coluna_regiao",      # ex: "municipio", "uf"
18         "act": "nome_coluna_atividade",   # ex: "cnae", "produto"
19         "val": "nome_coluna_valor",       # ex: "empregos", "exportacoes"
20         "time": "nome_coluna_ano",        # opcional
21     },
22     method="eigenvector",                # 'eigenvector' | 'reflections' | 'fitness'

```

```

23     threshold=1.0,
24     compute_coi_cog=True,
25 )
26
27 # 3. Exportar resultados
28 resultado.to_csv("resultados_complexidade.csv", index=False)
29 print("Resultado salvo em resultados_complexidade.csv")
30 print("Colunas geradas:", sorted(resultado.columns.tolist()))
31
32 # 4. Indicadores individuais (opcional)
33 mat = ec.pivot_to_matrix(df,
34                             "nome_coluna_regiao",
35                             "nome_coluna_atividade",
36                             "nome_coluna_valor")
37
38 eci, pci = ec.eci_pci(mat)
39 hachman = ec.hachman_index(mat)
40 krugman = ec.krugman_index(mat)
41 entropia = ec.shannon_entropy(mat)
42
43 saida = pd.DataFrame({
44     "eci": eci, "hachman": hachman,
45     "krugman": krugman, "entropia": entropia,
46 })
47 print(saida.round(3))

```

Listing 11: Template mínimo — copiar e adaptar

Execução com Dados da RAIS / CNAE (caso de uso real)

```

1  import pandas as pd
2  import econcomplex as ec
3
4  # Carregar RAIS agregada
5  # Esperado: municipio, cnae2d, ano, vinculos
6  rais = pd.read_parquet("rais_municipio_cnae.parquet")
7
8  # Filtrar um unico ano para analise
9  rais_2022 = rais[rais["ano"] == 2022].copy()
10
11 # Pipeline
12 resultado = ec.compute_complexity(
13     rais_2022,
14     cols={"loc": "municipio", "act": "cnae2d", "val": "vinculos"},
15     method="eigenvector",
16     compute_coi_cog=False, # desligar COG para bases grandes
17 )
18
19 # Top 20 municipios mais complexos
20 top20 = (resultado[["municipio", "eci"]]
21         .drop_duplicates()
22         .sort_values("eci", ascending=False)
23         .head(20))
24 print(top20.to_string(index=False))
25
26 resultado.to_csv("complexidade_municipios_2022.csv", index=False)

```

Listing 12: Uso com dados RAIS — municípios x CNAE

Execução em Pannel Multitemporal

Para analisar a evolução dos indicadores ao longo do tempo, basta incluir a coluna temporal no dicionário `cols`:

```

1 # rais com coluna "ano" contendo 2010, 2015, 2020
2 resultado_painel = ec.compute_complexity(
3     rais,
4     cols={
5         "loc": "municipio",
6         "act": "cnae2d",
7         "val": "vinculos",
8         "time": "ano",          # ativa o loop temporal
9     },
10    method="reflections",
11    compute_coi_cog=False,
12 )
13
14 # O resultado ja contem todos os anos
15 print(resultado_painel["ano"].value_counts().sort_index())
16
17 # Evolucao do ECI mediano por ano
18 eci_evolucao = (resultado_painel[["ano", "municipio", "eci"]]
19                 .drop_duplicates()
20                 .groupby("ano")["eci"]
21                 .median()
22                 .round(4))
23 print(eci_evolucao)
```

Listing 13: Execução em painel — múltiplos anos

Nota

O pipeline itera automaticamente sobre cada período, garantindo que a matriz de proximidade e os vetores de complexidade sejam recalculados de forma independente para cada ano.

Indicadores do Núcleo (core)

Vantagem Comparativa Revelada (RCA)

A RCA de Balassa é o principal filtro de especialização:

$$RCA_{il} = \frac{x_{il} / \sum_l x_{il}}{\sum_i x_{il} / \sum_{il} x_{il}}$$

```

1 import econcomplex as ec
2 import pandas as pd
3
4 df = pd.read_csv("dados.csv")
5 mat = ec.pivot_to_matrix(df, "regiao", "setor", "empregos")
6
7 rca      = ec.rca(mat)                # valores continuos
8 mcp      = ec.mcp(mat, threshold=1)  # 1 se RCA >= 1, 0 caso contrario
9 rca_rpop = ec.rpop(mat, pop)         # RCA de Puga & Turrini (requer
    vetor de populacao)
```

Listing 14: RCA contínuo e binário

Diversidade e Ubiquidade

```

1 div = ec.diversity(mat)      # numero de setores com RCA >= 1 por regiao
2 ubi = ec.ubiquity(mat)      # numero de regioes com RCA >= 1 por setor

```

Listing 15: Diversidade e ubiquidade

Complexidade Econômica (complexity)

Método do Vetor Próprio (Hidalgo & Hausmann, 2009)

$$ECI = \frac{\vec{v}_2 - \mu(\vec{v}_2)}{\sigma(\vec{v}_2)}, \quad PCI = \frac{\vec{u}_2 - \mu(\vec{u}_2)}{\sigma(\vec{u}_2)}$$

onde $\vec{v}_2$ é o segundo autovetor esquerdo da matriz $\tilde{M}$:

$$\tilde{M}_{ij} = \frac{M_{ij}}{k_{c,0} \cdot k_{p,0}}$$

```

1 eci, pci = ec.eci_pci(mat, method="eigenvector")
2 print(eci.head())      # Series indexada por regiao
3 print(pci.head())      # Series indexada por produto/setor
4
5 # Pre-processamento: unidades com diversidade/ubiquidade zero sao
6 # removidas automaticamente e retornadas como NaN. Desative com
7 # trim=False, ou restrinja ao nucleo conectado com dmin=2, umin=2.
8 core = ec.trim_core(mat, dmin=2, umin=2)      # tambem disponivel avulsa

```

Listing 16: ECI e PCI pelo método do vetor próprio

Método de Reflexões (Hidalgo & Hausmann, 2009)

```

1 eci_r, pci_r = ec.eci_pci(mat, method="reflections", iterations=20)
2 # tol=1e-10 interrompe as iteracoes quando as reflexoes padronizadas
3 # estabilizam; os sinais sao orientados como no metodo do autovetor
4 # (ECI correlaciona positivamente com diversidade, PCI negativamente
5 # com ubiquidade).

```

Listing 17: Método de Reflexões

Fitness-Complexity (Tacchella et al., 2012)

$$F_c^{(n+1)} = \sum_p M_{cp} Q_p^{(n)}, \quad Q_p^{(n+1)} = \frac{1}{\sum_c M_{cp} / F_c^{(n)}}$$

```

1 fit, comp = ec.eci_pci(mat, method="fitness", iterations=20)
2 # iterations e um teto (20, como no pacote R economiçcomplexity);
3 # o laco para na convergencia (tol) e avisa se houver instabilidade
4 # real. log_fitness=True retorna a escala log recomendada por
5 # Cristelli et al. (2015):
6 fit_log, comp_log = ec.fitness_complexity(mat, log_fitness=True)

```

Listing 18: Fitness-Complexity

ECI Subnacional

```
1 # pci_nacional: PCI calculado com dados nacionais/internacionais
2 eci_sub = ec.subnational_eci(mat_sub, pci_nacional)
```

Listing 19: ECI subnacional com PCI externo

Product Space e Relatedness (relatedness)

Proximidade (ϕ)

$$\phi_{ij} = \min(P(RCA_i \geq 1 \mid RCA_j \geq 1), P(RCA_j \geq 1 \mid RCA_i \geq 1))$$

```
1 phi_d = ec.proximity(mat)["product"] # baseada em Mcp
    binario
2 phi_c = ec.proximity(mat, continuous=True)["product"] # baseada em RCA
    contínuo
```

Listing 20: Proximidade discreta e contínua

Densidade e Distância

$$\omega_{ij} = \frac{\sum_k M_{kj} \phi_{ij}}{\sum_k \phi_{ij}}, \quad d_{ij} = 1 - \omega_{ij}$$

```
1 density = ec.density(mat, phi_d) # DataFrame loc x act
2 distance = ec.distance(mat, phi_d) # DataFrame loc x act
```

Listing 21: Densidade e distância

Co-ocorrência e Índices de Relatedness

```
1 cooc = ec.co_occurrence(mat)
2 phi_cos = ec.cosine_proximity(mat) # baseada em cosseno
3 phi_cor = ec.correlation_proximity(mat)
4 rca_rel = ec.relatedness(mat, phi_d) # densidades de relatedness
```

Listing 22: Co-ocorrência e índices de relatedness

Cross-Space (relatedness entre classificações distintas)

```
1 # phi_cross: proximidade entre setores e tecnologias (matrizes
    diferentes)
2 phi_cross = ec.cross_space_proximity(mat_setor, mat_tech)
3 density_c = ec.cross_relatedness(mat_setor, phi_cross)
```

Listing 23: Proximidade e relatedness cross-space

Especialização Regional (specialization)

```

1 lq      = ec.location_quotient(mat)      # equivalente ao RCA
2 hachman = ec.hachman_index(mat)          # 0 (diversificado) a 1
   (especializado)
3 krugman = ec.krugman_index(mat)          # diferenca estrutural entre
   regioes
4 coef_esp = ec.spec_coefficient(mat)      # coeficiente de especializacao
5 sim_exp  = ec.export_similarity(mat)     # similaridade de estrutura
   produtiva

```

Listing 24: Índices de especialização

Desigualdade e Concentração (inequality)

```

1 gini_r   = ec.gini(mat)                  # Gini da distribuicao do valor
2 gini_loc = ec.locational_gini(mat)       # Gini locacional (por setor)
3 hoover   = ec.hoover_gini(mat)           # Indice de Hoover
4 hhi      = ec.hhi(mat)                   # Herfindahl-Hirschman
5 entropia = ec.shannon_entropy(mat)       # entropia de Shannon (por regioao)

```

Listing 25: Indicadores de desigualdade

Produtividade (productivity)

```

1 prody = ec.prody(mat, gdp_per_capita)    # renda associada ao produto
2 expy  = ec.expy(mat, gdp_per_capita)     # sofisticacao da cesta
   exportadora
3 pgi   = ec.pgi(mat, gini_vec)             # Product Gini Index (vetor de
   Gini regional)
4 peii  = ec.peii(mat, emissions)          # Product Emissions Intensity
   Index

```

Listing 26: Produtividade e desigualdade dos produtos

Complexidade Baseada em Patentes (patents)

```

1 ease_rec = ec.ease_of_recombination(mat_pat)
2 mod_comp = ec.modular_complexity(mat_pat)

```

Listing 27: Indicadores de patentes (input: matriz patente x tecnologia)

Dinâmica Temporal (dynamics)

```

1 growth = ec.growth_rates(df, "regiao", "setor", "empregos", "ano")
2 entries = ec.entry_tracking(df, "regiao", "setor", "empregos", "ano")
3 exits   = ec.exit_tracking(df, "regiao", "setor", "empregos", "ano")

```

Listing 28: Crescimento e rastreamento de entrada/saída

Complexity Outlook (outlook)

O **COI** mede o potencial latente de diversificação; o **COG** mede o ganho esperado de complexidade:

$$COI_c = \sum_p (1 - M_{cp}) \omega_{cp} Q_p$$

```
1 coi = ec.coi(mat, pci, phi=phi_d)
2 cog = ec.cog(mat, pci, phi=phi_d)
```

Listing 29: COI e COG

Otimização de Complexidade (optimization)

Camada de otimização orientada a metas para diversificação estratégica (Stojkoski & Hidalgo, 2026, *Research Policy* 55, 105454). Um modelo de previsão com steppingstone é calibrado em um painel histórico, o esforço W_{cp} (o RCA adicional necessário para entrar em uma atividade) é obtido em forma fechada, e um programa inteiro 0–1 seleciona o portfólio de menor esforço que atinge um ECI alvo.

```
1 # 1. Calibrar o modelo steppingstone em um painel (t, t+5, t+10)
2 model = ec.calibrate_steppingstone(df, "regiao", "setor", "empregos",
3                                   "ano", horizon=10, steppingstone=5)
4
5 # 2. Esforço e previsão sem politica para o ultimo ano
6 W = ec.effort_matrix(mat, model)          # RCA adicional por
7     candidato
8
9 fc = ec.forecast_specialization(mat, model) # rca, mcp, pci, eci
10
11 # 3. Portfólio de menor esforço que eleva o ECI de cada local em 0.1
12 portfolio = ec.eci_optimization(mat, model, delta_eci=0.1)
13
14 # 4. Opcional: meta de crescimento - converte meta (3.5%/ano)
15 #     em ECI alvo e otimiza para ele
16 gm = ec.calibrate_growth_model(painel, "regiao", "ano", "pibpc", "eci")
17 eci_alvo = ec.eci_target_for_growth(gm, 0.035, pibpc_atual)
18 portfolio = ec.eci_optimization(mat, model, target_eci=eci_alvo)
```

Listing 30: Fluxo da Otimização de ECI

Difusão Estratégica (optimization.diffusion)

Modelo de contágio complexo de diversificação na rede de atividades relacionadas (Alshamsi, Pinheiro & Hidalgo, 2018, *Nature Communications* 9, 1328): a probabilidade de entrar em uma atividade é $p_i = B$ (fração de relacionadas presentes) $^\alpha$. A estratégia gulosa de sempre entrar na atividade mais relacionada é subótima; o tempo mínimo de diversificação exige mirar hubs em uma janela estreita e relativamente cedo.

```
1 adj = ec.proximity_network(mat, phi_threshold=0.55) # product space
2 fit = ec.calibrate_contagion(df, "regiao", "setor", "empregos", "ano",
3                             adjacency=adj)          # B e alpha
4
```



```

5 active0 = ec.mcp(mat).loc["minha_regiao"] # portfolio atual
6 seq = ec.diversification_strategy(adj, active0, B=fit["B"],
7                                   alpha=fit["alpha"], strategy="greedy")
8 tabela = ec.compare_strategies(adj, active0, B=fit["B"],
9                                   alpha=fit["alpha"])
10 otimo = ec.optimize_sequence(adj, active0, B=fit["B"],
11                               alpha=fit["alpha"])

```

Listing 31: Fluxo de difusão estratégica

Pipeline Unificado (compute_complexity)

Assinatura da Função

```

1 def compute_complexity(
2     data: pd.DataFrame,
3     cols: dict, # {"loc": str, "act": str, "val": str,
4                 "time": str}
5     *,
6     method: str = "eigenvector", # 'eigenvector' / 'reflections' /
7                                   'fitness'
8     presence_test: str = "rca", # 'rcá' / 'rpop' / 'both' / 'manual'
9     threshold: float = 1.0, # limiar de binarizacao
10    iterations: int = 20, # para reflections/fitness
11    proximity_type: str = "discrete", # 'discreté' / 'continuous'
12    proximity_method: str = "max", # 'max' / 'sqrt' / 'min'
13    pop: pd.DataFrame | None = None, # populacao (para 'rpop'/'both')
14    compute_coi_cog: bool = True,
15    trim: bool = True, # poda unidades degeneradas
16    dmin: int = 1, umin: int = 1, # nucleo (use 2,2 se esparso)
17    time_col: str | None = None,
18 ) -> pd.DataFrame:
19     """
20     Retorna o DataFrame original enriquecido com as colunas:
21     rca, mcp, diversity, ubiquity, eci, pci,
22     density, distance, [coi, cog]
23     """

```

Listing 32: Assinatura da função pipeline

Nota

`compute_complexity` é uma fina **camada de orquestração**: chama as mesmas funções públicas que você chamaria na mão (`rca`, `mcp`, `diversity`, `ubiquity`, `eci_pci`, `proximity`, `relatedness_density`, `distance`, `coi/cog`) e remonta o resultado em formato longo, iterando por período em painéis. É a porta de entrada para o caso comum; para controle fino, use as funções de matriz diretamente.

Parâmetros

Parâmetro	Tipo	Descrição
<code>data</code>	<code>DataFrame</code>	Tabela no formato longo
<code>cols</code>	<code>dict</code>	Mapeamento de colunas (ver Seção 2.3)
<code>method</code>	<code>str</code>	Método de complexidade (porta de entrada única)
<code>presence_test</code>	<code>str</code>	Como binarizar o Mcp (RCA, RPOP, ambos, manual)
<code>threshold</code>	<code>float</code>	Limiar de binarização (padrão: 1.0)
<code>iterations</code>	<code>int</code>	Iterações para métodos iterativos (padrão: 20)
<code>proximity_type</code>	<code>str</code>	Proximidade discreta (co-ocorrência) ou contínua
<code>proximity_method</code>	<code>str</code>	Normalização da proximidade
<code>pop</code>	<code>DataFrame</code>	População, obrigatória para RPOP
<code>compute_coi_cog</code>	<code>bool</code>	Calcular COI e COG (mais lento)
<code>trim</code>	<code>bool</code>	Poda unidades degeneradas (padrão: True)
<code>dmin, umin</code>	<code>int</code>	Limiares do núcleo repassados a <code>eci_pci</code> ; use 2 para dados esparsos

Nota

Unidades degeneradas (diversidade ou ubiquidade zero) são removidas automaticamente na etapa de complexidade e retornadas como NaN (ver `trim_core`). Para dados subnacionais esparsos, passe `dmin=2`, `umin=2` para restringir ao núcleo conectado.

Como Interpretar os Indicadores

Guia rápido de leitura dos principais resultados. Todos os índices de complexidade são **medidas relativas**: compare apenas dentro da mesma matriz (mesmo período e mesma classificação de atividades).

Indicador	Faixa típica	Leitura
ECI / PCI	z-score (média 0, dp 1)	Maior = local / atividade mais complexo. NaN = unidade podada por diversidade/ubiquidade zero.
Fitness / Complexity	> 0 (ou escala log)	Análogo não linear do ECI/PCI; o ranking é o que importa.
RCA / LQ	≥ 0 ; 1 = neutro	> 1 : local especializado na atividade; base da matriz binária M_{cp} .
Proximidade ϕ	$[0, 1]$	Similaridade tipo-probabilidade entre atividades; maior = especialização conjunta mais fácil.
Densidade ω	0–100 %	Fração das atividades relacionadas a c que o local já possui; maior = entrada mais fácil.
Relatedness relativa $\tilde{\omega}$	z-score	> 0 : atividade mais relacionada que a média do option set do local.
Distância	$[0, 1]$	$1 - \omega/100$; maior = entrada mais difícil.
COI	sem limite	Maior = mais atividades complexas ao alcance (potencial de diversificação).
COG	≥ 0 por atividade	Ganho de COI se o local desenvolver a atividade; identifica apostas estratégicas.
Gini locacional	$[0, 0,5]$	Maior = atividade geograficamente mais concentrada.
HHI (normalizado)	$[0, 1]$	Maior = portfólio do local mais concentrado. A entropia lê ao contrário.
Hachman	$[0, 1]$	1 = local espelha a estrutura agregada.
Krugman	$[0, 2]$	Maior = estrutura mais diferente do agregado.
PRODY / EXPY	unidades do vetor de renda	Nível de renda associado à atividade / à cesta do local.
Esforço W_{cp}	RCA adicional	Menor = entrada mais barata; ≤ 0 indica que o modelo já prevê a entrada.
Contágio B, α	$B \in (0, 1], \alpha \approx 1$	$\alpha > 1$: atividades relacionadas se reforçam (contágio complexo).

Atenção

Insumos como PIB per capita (PRODY/EXPY), população (RPOP), vetor de Gini regional (PGI) e emissões (PEII) são **dados externos**: a biblioteca não os deriva da matriz, e variáveis proxy produzem indicadores sem significado econômico.

Exemplos Completos

Exemplo 1 — Análise Regional de Empregos

```

1 import pandas as pd
2 import numpy as np
3 import econcomplex as ec
4
```

```

5 # Gerar dados sinteticos (ou carregar CSV real)
6 df = ec.make_sample_data(n_locs=50, n_acts=30, seed=42)
7 # Colunas: loc, act, val
8
9 # Pipeline completo
10 resultado = ec.compute_complexity(
11     df,
12     cols={"loc": "loc", "act": "act", "val": "val"},
13     method="eigenvector",
14     compute_coi_cog=True,
15 )
16
17 # Resumo dos indicadores gerados
18 print(resultado.columns.tolist())
19 # ['loc', 'act', 'val', 'rcá', 'mcp', 'diversity', 'ubiquity',
20 #  'éci', 'pci', 'density', 'distancé', 'coí', 'cog']
21
22 # Top 10 regioes por ECI
23 top_eci = (resultado[["loc", "eci"]]
24             .drop_duplicates()
25             .sort_values("eci", ascending=False)
26             .head(10))
27 print(top_eci)
28
29 # Salvar
30 resultado.to_csv("resultado_regional.csv", index=False)

```

Listing 33: Análise completa de emprego regional

Exemplo 2 — Exportações com Dados COMEX

```

1 import pandas as pd
2 import econcomplex as ec
3
4 # Carregar base COMEX agregada por municipio x NCM x ano
5 comex = pd.read_parquet("comex_mun_ncm.parquet")
6 comex_2023 = comex[comex["ano"] == 2023].copy()
7
8 # Limpeza
9 comex_2023 = comex_2023.dropna(subset=["co_mun", "co_ncm", "vl_fob"])
10 comex_2023 = comex_2023[comex_2023["vl_fob"] > 0]
11
12 # Pipeline
13 resultado = ec.compute_complexity(
14     comex_2023,
15     cols={"loc": "co_mun", "act": "co_ncm", "val": "vl_fob"},
16     method="eigenvector",
17     compute_coi_cog=True,
18 )
19
20 # Salvar
21 resultado.to_csv("complexidade_exportacoes_2023.csv", index=False)
22 print("ECI medio:", resultado["eci"].dropna().mean().round(4))

```

Listing 34: Análise de exportacoes com dados COMEX/MDIC

Exemplo 3 — Painel Temporal RAIS (2010–2022)

```

1 import pandas as pd
2 import econcomplex as ec
3
4 rais = pd.read_parquet("rais_municipio_cnae_2010_2022.parquet")
5
6 # Pipeline temporal -- itera sobre cada ano automaticamente
7 painel = ec.compute_complexity(
8     rais,
9     cols={
10         "loc": "municipio",
11         "act": "cnae2d",
12         "val": "vinculos",
13         "time": "ano",
14     },
15     method="eigenvector",
16     compute_coi_cog=False,
17 )
18
19 # Evolucao do ECI mediano
20 eci_trend = (painel[["ano", "municipio", "eci"]]
21               .drop_duplicates()
22               .groupby("ano")["eci"]
23               .median()
24               .round(4))
25 print(eci_trend)
26
27 painel.to_csv("painel_complexidade_2010_2022.csv", index=False)

```

Listing 35: Pipeline com dados em painel — múltiplos períodos

Exemplo 4 — Alvos de Diversificação (Otimização de ECI)

Uso passo a passo da camada de otimização com um painel multi-anual (exige ao menos os períodos t , $t + \tau$ e $t + \Delta t$):

```

1 import pandas as pd
2 import econcomplex as ec
3
4 # 1. Painel com ao menos t, t+5, t+10 (ex.: 2005, 2010, 2015)
5 rais = pd.read_parquet("rais_municipio_cnae_2005_2015.parquet")
6
7 # 2. Calibrar o modelo de previsao com steppingstone
8 model = ec.calibrate_steppingstone(
9     rais, "municipio", "cnae2d", "vinculos", "ano",
10     horizon=10, steppingstone=5,
11 )
12
13 # 3. Matriz do ultimo ano e portfolios de menor esforco
14 mat = ec.pivot_to_matrix(rais[rais["ano"] == 2015],
15                           "municipio", "cnae2d", "vinculos")
16 portfolio = ec.eci_optimization(mat, model, delta_eci=0.1)
17 print(portfolio.head()) # location, activity, effort, pci, alvos
18
19 # 4. Opcional: meta de crescimento (exige painel de PIB pc e ECI)
20 gm = ec.calibrate_growth_model(macro, "municipio", "ano",
21                                "pibpc", "eci", horizon=10)
22 eci_alvo = ec.eci_target_for_growth(gm, 0.035, pibpc_atual)

```

```

23 portfolio = ec.eci_optimization(mat, model, target_eci=eci_alvo)
24
25 # 5. Opcional: timing da diversificacao (difusao estrategica)
26 adj = ec.proximity_network(mat, phi_threshold=0.55)
27 fit = ec.calibrate_contagion(rais, "municipio", "cnae2d",
28                             "vinculos", "ano", adjacency=adj)
29 active0 = ec.mcp(mat).loc["meu_municipio"]
30 otimo = ec.optimize_sequence(adj, active0,
31                             B=fit["B"], alpha=fit["alpha"])
32 print(otimo["sequence"][:10], otimo["improvement"])

```

Listing 36: Otimização de ECI de ponta a ponta

Referência da API

Inventário completo das 87 funções públicas, agrupadas por módulo. As assinaturas mostram os parâmetros e seus valores padrão; consulte os docstrings no código (`help(ec.funcao)`) para a documentação completa de cada parâmetro. Esta seção é gerada automaticamente a partir do pacote instalado por `docs/generate_api_reference.py`.

core — Core

Função	Descrição
<code>rca(mat, binary=False, threshold=1.0)</code>	Vantagem Comparativa Revelada (índice de Balassa).
<code>rpop(mat, pop, binary=False, threshold=1.0)</code>	RCA normalizada por população (RPOP).
<code>mcp(mat, presence_test='rca', rca_threshold=1.0, rpop_threshold=1.0, pop=None)</code>	Matriz binária de presença Mcp (RCA, RPOP, ambas ou manual).
<code>diversity(mat, use_rca=True, threshold=1.0)</code>	Diversidade: número de atividades com RCA acima do limiar, por região.
<code>ubiquity(mat, use_rca=True, threshold=1.0)</code>	Ubiquidade: número de regiões com RCA acima do limiar, por atividade.
<code>normalized_ubiquity(mat, threshold=1.0)</code>	Ubiquidade normalizada pela participação no valor total.
<code>pivot_to_matrix(df, index, columns, values, fill_value=0.0)</code>	Converte DataFrame longo em matriz larga (pivot).
<code>melt_matrix(mat, index_name='location', columns_name='activity', values_name='value')</code>	Converte matriz larga em DataFrame longo.
<code>binarize(mat, threshold=1.0)</code>	Binariza a matriz no limiar dado.
<code>normalize_zscore(vec)</code>	Padroniza um vetor (z-score).
<code>normalize_01(vec)</code>	Normaliza um vetor para [0, 1] (min-max).
<code>make_sample_data(n_locs=50, n_acts=30, seed=42, loc_col='loc', act_col='act', val_col='val')</code>	Gera dados sintéticos longos com estrutura aninhada (exemplos e testes).
<code>trim_core(mat, dmin=1, umin=1, use_rca=True, threshold=1.0, max_iter=100)</code>	Poda iterativa ao núcleo (dmin, umin), removendo unidades degeneradas.

complexity — Complexidade

Função	Descrição
<code>eci_pci(mat, use_rca=True, threshold=1.0, method='eigenvector', iterations=None, extremality=1.0, tol=1e-10, log_fitness=False, trim=True, dmin=1, umin=1)</code>	Porta de entrada única do ECI/PCI (autovetor, reflexões ou fitness), com poda automática.
<code>eci_pci_eigenvector(mat, use_rca=True, threshold=1.0)</code>	Implementação do método do autovetor (Hidalgo & Hausmann 2009).
<code>method_of_reflections(mat, use_rca=True, threshold=1.0, iterations=20, return_both=True, tol=1e-10)</code>	Método das Reflexões iterativo para ECI/PCI.
<code>mor_regions(mat, use_rca=True, threshold=1.0, steps=20)</code>	Reflexões apenas do lado das regiões, no passo escolhido.
<code>mor_activities(mat, use_rca=True, threshold=1.0, steps=19)</code>	Reflexões apenas do lado das atividades.
<code>fitness_complexity(mat, use_rca=True, threshold=1.0, iterations=20, extremality=1.0, tol=1e-10, log_fitness=False)</code>	Algoritmo Fitness–Complexity (Tacchella et al. 2012).
<code>subnational_eci(mat, pci_external, use_rca=True, threshold=1.0, standardize=True)</code>	ECI subnacional projetado com PCI externo.

relatedness — Relatedness e Product Space

Função	Descrição
<code>proximity(mat, use_rca=True, threshold=1.0, method='max', compute='both', continuous=False, continuous_method='correlation')</code>	Matrizes de proximidade entre produtos e/ou regiões (discreta ou contínua).
<code>continuous_proximity(rca_mat, method='correlation')</code>	Proximidade contínua a partir do RCA (correlação ou cosseno).
<code>cosine_proximity(mat, use_rca=True)</code>	Atalho: proximidade contínua por cosseno.
<code>correlation_proximity(mat, use_rca=True)</code>	Atalho: proximidade contínua por correlação.
<code>relatedness_density(mat, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Densidade de relatedness por par (região, atividade), em percentual.
<code>density(mat, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Alias de <code>relatedness_density</code> .
<code>relatedness(mat, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Alias de <code>relatedness_density</code> .
<code>distance(mat, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Distância: $1 - \text{densidade}/100$.
<code>relatedness_density_internal(mat, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Densidade restrita às atividades já presentes.

Função	Descrição
<code>relatedness_density_external(mat, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Densidade restrita às atividades ausentes.
<code>relative_relatedness(mat, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Relatedness relativa: z-score no option set (Pinheiro et al. 2022).
<code>co_occurrence(mat, diagonal=False)</code>	Matriz de coocorrência entre atividades.
<code>relatedness_index(mat, method='cosine', diagonal=False)</code>	Índice de relatedness por coocorrência (probabilidade, associação, cosseno, Jaccard).
<code>z_score_novelty(incidence)</code>	Z-score de novidade (atipicidade da coocorrência).
<code>cross_proximity(mat_a, mat_b, use_rca=True, threshold=1.0)</code>	Proximidade entre dois espaços de atividades distintos.
<code>cross_relatedness(mat_a, x_phi, use_rca=True, threshold=1.0)</code>	Densidade de relatedness cross-space (regiões x espaço B).
<code>cross_space_proximity(mat_a, mat_b, use_rca=True, threshold=1.0)</code>	Alias de <code>cross_proximity</code> .

specialization — Especialização Regional

Função	Descrição
<code>location_quotient(mat, binary=False, threshold=1.0)</code>	Quociente Locacional (idêntico ao RCA).
<code>location_quotient_avg(mat)</code>	LQ médio ponderado por região (coeficiente de Hoover).
<code>hachman_index(mat)</code>	Índice de Hachman (similaridade à estrutura nacional).
<code>specialization_coefficient(mat)</code>	Coefficiente de especialização de Hoover.
<code>spec_coefficient(mat)</code>	Alias de <code>specialization_coefficient</code> .
<code>krugman_index(mat)</code>	Índice de especialização de Krugman.
<code>export_similarity(mat, use_rca=True, epsilon=0.1, log=True)</code>	Similaridade de pauta entre regiões (Bahar et al. 2014).

inequality — Desigualdade e Concentração

Função	Descrição
<code>gini(x)</code>	Coefficiente de Gini padrão (vetor ou colunas de DataFrame).
<code>locational_gini(mat)</code>	Gini locacional de Krugman, por atividade.
<code>hoover_gini(mat, pop=None)</code>	Gini com eixo populacional (curva de Hoover).
<code>herfindahl(mat, normalize=True)</code>	Índice Herfindahl–Hirschman por região.

Função	Descrição
<code>hhi(mat, normalize=True)</code>	Alias de <code>herfindahl</code> .
<code>shannon_entropy(mat, base=2.0)</code>	Entropia de Shannon por região (diversificação).
<code>hoover_index(mat, pop=None)</code>	Índice de Hoover (Robin Hood) por atividade.

productivity — Produtividade

Função	Descrição
<code>prody(mat, gdp, use_rca=True)</code>	PRODY: nível de renda associado a cada atividade.
<code>expy(mat, gdp, use_rca=True)</code>	EXPY: renda implícita da cesta de cada região.
<code>product_gini_index(mat, gini_vec, threshold=1.0)</code>	PGI: desigualdade embutida em cada produto.
<code>pgi(mat, gini_vec, threshold=1.0)</code>	Alias de <code>product_gini_index</code> .
<code>product_emissions_index(mat, emissions, threshold=1.0)</code>	PEII: intensidade de emissões embutida em cada produto.
<code>peii(mat, emissions, threshold=1.0)</code>	Alias de <code>product_emissions_index</code> .

patents — Complexidade de Patentes

Função	Descrição
<code>ease_of_recombination(incidence)</code>	Facilidade de recombinação (EOR) por tecnologia.
<code>modular_complexity(incidence, eor=None)</code>	Complexidade modular de cada patente.
<code>modular_complexity_avg(incidence, eor=None)</code>	Complexidade modular média por tecnologia.

dynamics — Dinâmica Temporal

Função	Descrição
<code>growth_rate(mat1, mat2, axis=0, pct=True)</code>	Taxa de crescimento agregada entre dois períodos.
<code>growth_matrix(mat1, mat2, pct=True)</code>	Matriz de crescimento célula a célula.
<code>growth_rates(df, loc, act, val, time, axis=0, pct=True)</code>	Crescimento em painel longo, entre períodos consecutivos.
<code>entry(mats, use_rca=True, threshold=1.0)</code>	Entradas: transições 0 para 1 na especialização.
<code>exit(mats, use_rca=True, threshold=1.0)</code>	Saídas: transições 1 para 0 na especialização.

Função	Descrição
<code>entry_exit_summary(mats, use_rca=True, threshold=1.0)</code>	Resumo de entradas e saídas por par (região, atividade).
<code>entry_tracking(df, loc, act, val, time, use_rca=True, threshold=1.0)</code>	Entradas em formato longo (painel).
<code>exit_tracking(df, loc, act, val, time, use_rca=True, threshold=1.0)</code>	Saídas em formato longo (painel).

outlook — Complexity Outlook

Função	Descrição
<code>complexity_outlook_index(mat, pci, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	COI: potencial de diversificação rumo a atividades complexas.
<code>complexity_outlook_gain(mat, pci, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	COG: ganho de COI ao desenvolver cada atividade.
<code>coi(mat, pci, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Alias de <code>complexity_outlook_index</code> .
<code>cog(mat, pci, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Alias de <code>complexity_outlook_gain</code> .

optimization — Otimização de ECI e Difusão Estratégica

Função	Descrição
<code>calibrate_steppingstone(df, loc, act, val, time, horizon=10, steppingstone=5, threshold=1.0, proximity_method='max')</code>	Calibra o modelo forward com steppingstone (OLS de entrada/saída).
<code>effort_matrix(mat, model)</code>	Esforço W_{cp} : RCA adicional necessário para entrar em cada atividade.
<code>forecast_specialization(mat, model)</code>	Projeção sem política ($W=0$): RCA, M_{cp} , PCI e ECI futuros.
<code>eci_optimization(mat, model, delta_eci=0.1, target_eci=None, locations=None, solver='milp')</code>	Programa 0–1: portfólio de menor esforço para atingir o ECI alvo.
<code>calibrate_growth_model(df, loc, time, gdppc, eci, horizon=10)</code>	Regressão de crescimento em painel (ECI, convergência, interação).
<code>expected_growth(model, eci, gdppc_now, period=None)</code>	Crescimento anual esperado dado o ECI e o PIB per capita.
<code>eci_target_for_growth(model, growth_target, gdppc_now, period=None)</code>	Inverte a regressão: ECI compatível com a meta de crescimento.
<code>proximity_network(mat, phi=None, phi_threshold=0.55, method='max')</code>	Rede binária de atividades relacionadas (ϕ acima do limiar).
<code>activation_probabilities(adjacency, active, B=1.0, alpha=1.0)</code>	Probabilidade de ativação $p = Bx^\alpha$ por atividade.

Função	Descrição
<code>calibrate_contagion(df, loc, act, val, time, adjacency=None, phi_threshold=0.55, threshold=1.0, presence_test='rca', n_bins=20)</code>	Calibra B e alfa do contágio a partir das entradas observadas.
<code>diversification_strategy(adjacency, active0, B=1.0, alpha=1.0, strategy='greedy', seed=None)</code>	Sequência de alvos por estratégia heurística, com tempos esperados.
<code>expected_diversification_time(adjacency, active0, sequence, B=1.0, alpha=1.0)</code>	Tempo total esperado de uma sequência fixa de alvos.
<code>compare_strategies(adjacency, active0, B=1.0, alpha=1.0, n_random=10, seed=0)</code>	Compara o tempo total das cinco estratégias heurísticas.
<code>optimize_sequence(adjacency, active0, B=1.0, alpha=1.0, n_iter=2000, seed=0)</code>	Recozimento simulado sobre sequências; nunca pior que a gulosa.

pipeline — Pipeline Unificado

Função	Descrição
<code>compute_complexity(data, cols, *, method='eigenvector', presence_test='rca', threshold=1.0, iterations=20, proximity_type='discrete', proximity_method='max', pop=None, compute_coi_cog=True, trim=True, dmin=1, umin=1, time_col=None)</code>	Pipeline completo: todos os indicadores a partir do DataFrame longo (suporta painel).

Referências Bibliográficas

Balassa (1965)

Balassa, B. (1965). Trade liberalisation and “revealed” comparative advantage. *The Manchester School*, 33(2), 99–123.

Hidalgo & Hausmann (2009)

Hidalgo, C.A., & Hausmann, R. (2009). The building blocks of economic complexity. *PNAS*, 106(26), 10570–10575.

Balland (2017)

Balland, P.-A. (2017). *EconGeo: Computing Key Indicators of the Geography of Innovation*. R package.

Vargas Sepúlveda (2020)

Vargas Sepúlveda, M. (2020). *economiccomplexity: Computational Methods for Economic Complexity*. R package.

The Growth Lab at Harvard (2020)

The Growth Lab at Harvard University (2020). *py-econcomplexity: Economic Complexity in Python*. GitHub.

Tacchella et al. (2012)

Tacchella, A., Cristelli, M., Caldarelli, G., Gabrielli, A., Pietronero, L. (2012). A New Metrics for Countries' Fitness and Products' Complexity. *Scientific Reports*, 2, 723.

Hidalgo et al. (2007)

Hidalgo, C.A., Klinger, B., Barabási, A.-L., Hausmann, R. (2007). The product space conditions the development of nations. *Science*, 317(5837), 482–487.

Alshamsi et al. (2018)

Alshamsi, A., Pinheiro, F.L., Hidalgo, C.A. (2018). Optimal diversification strategies in the networks of related products and of related research areas. *Nature Communications*, 9, 1328.

Pinheiro et al. (2022)

Pinheiro, F.L., Hartmann, D., Boschma, R., Hidalgo, C.A. (2022). The time and frequency of unrelated diversification. *Research Policy*, 51(8), 104323.

Stojkoski & Hidalgo (2026)

Stojkoski, V., Hidalgo, C.A. (2026). Optimizing economic complexity. *Research Policy*, 55, 105454.

Licença e Citação

Esta biblioteca é distribuída sob a licença MIT. Para uso em trabalhos acadêmicos, cite a biblioteca (<https://github.com/eltonfreitas/econcomplex>) junto com os artigos originais dos indicadores utilizados (ver Referências).

```
1 import econcomplex as ec
2 print(ec.__version__)    # 1.0.1
```

Listing 37: Verificar versão da biblioteca