

# econcomplex

Technical Documentation — Version 1.0.0

---

Python Library for Economic Complexity  
and Regional Science Indicators

*Elton Freitas*

Dependencies: `numpy`  $\geq 1.21$ , `pandas`  $\geq 1.3$ , `scipy`  $\geq 1.9$   
Compatible with Python 3.9+ (NumPy 1.x and 2.x)

June 2026

## Contents

---

<b>1</b>	<b>Overview</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Module Structure . . . . .	4
1.3	API Map: Entry Points, Advanced Implementations, and Aliases . . . . .	4
<b>2</b>	<b>Data Organisation</b>	<b>5</b>
2.1	Standard Format: Long Table ( <i>long format</i> ) . . . . .	5
2.2	Data Quality Rules . . . . .	5
2.3	Column Names — <code>cols</code> Dictionary . . . . .	6
2.4	Supported File Formats . . . . .	6
2.5	Examples of Compatible Datasets . . . . .	7
<b>3</b>	<b>How to Run the Scripts</b>	<b>7</b>
3.1	Prerequisites . . . . .	7
3.2	Direct Execution via Terminal . . . . .	8
3.3	Running in Jupyter Notebook / JupyterLab . . . . .	8
3.4	Running in VS Code / PyCharm / Spyder . . . . .	9
3.5	Minimal Execution Script . . . . .	9
3.6	Running with RAIS / CNAE Data (real use case) . . . . .	10
3.7	Running a Multi-Year Panel . . . . .	10
<b>4</b>	<b>Core Indicators (<code>core</code>)</b>	<b>11</b>
4.1	Revealed Comparative Advantage (RCA) . . . . .	11
4.2	Diversity and Ubiquity . . . . .	12
<b>5</b>	<b>Economic Complexity (<code>complexity</code>)</b>	<b>12</b>
5.1	Eigenvector Method (Hidalgo & Hausmann, 2009) . . . . .	12
5.2	Method of Reflections (Hidalgo & Hausmann, 2009) . . . . .	12
5.3	Fitness-Complexity (Tacchella et al., 2012) . . . . .	12
5.4	Subnational ECI . . . . .	13
<b>6</b>	<b>Product Space and Relatedness (<code>relatedness</code>)</b>	<b>13</b>
6.1	Proximity ( $\phi$ ) . . . . .	13
6.2	Density and Distance . . . . .	13
6.3	Co-occurrence and Relatedness Indices . . . . .	13
6.4	Cross-Space (relatedness between distinct classifications) . . . . .	13
<b>7</b>	<b>Regional Specialization (<code>specialization</code>)</b>	<b>14</b>
<b>8</b>	<b>Inequality and Concentration (<code>inequality</code>)</b>	<b>14</b>
<b>9</b>	<b>Productivity (<code>productivity</code>)</b>	<b>14</b>
<b>10</b>	<b>Patent-Based Complexity (<code>patents</code>)</b>	<b>14</b>
<b>11</b>	<b>Temporal Dynamics (<code>dynamics</code>)</b>	<b>14</b>
<b>12</b>	<b>Complexity Outlook (<code>outlook</code>)</b>	<b>15</b>

<b>13 ECI Optimization (<code>optimization</code>)</b>	<b>15</b>
<b>14 Strategic Diffusion (<code>optimization.diffusion</code>)</b>	<b>15</b>
<b>15 Unified Pipeline (<code>compute_complexity</code>)</b>	<b>16</b>
15.1 Function Signature . . . . .	16
15.2 Parameters . . . . .	17
<b>16 Interpreting the Indicators</b>	<b>17</b>
<b>17 Complete Examples</b>	<b>18</b>
17.1 Example 1 — Regional Employment Analysis . . . . .	18
17.2 Example 2 — Exports with COMEX Data . . . . .	19
17.3 Example 3 — RAIS Time Panel (2010–2022) . . . . .	20
17.4 Example 4 — Diversification Targets (ECI Optimization) . . . . .	20
<b>18 API Reference</b>	<b>21</b>
18.1 <code>core</code> — Core . . . . .	21
18.2 <code>complexity</code> — Complexity . . . . .	22
18.3 <code>relatedness</code> — Relatedness and Product Space . . . . .	22
18.4 <code>specialization</code> — Regional Specialization . . . . .	23
18.5 <code>inequality</code> — Inequality and Concentration . . . . .	24
18.6 <code>productivity</code> — Productivity . . . . .	24
18.7 <code>patents</code> — Patent-Based Complexity . . . . .	24
18.8 <code>dynamics</code> — Temporal Dynamics . . . . .	24
18.9 <code>outlook</code> — Complexity Outlook . . . . .	25
18.10 <code>optimization</code> — ECI Optimization and Strategic Diffusion . . . . .	25
18.11 <code>pipeline</code> — Unified Pipeline . . . . .	26
<b>19 References</b>	<b>27</b>
<b>20 License and Citation</b>	<b>28</b>

## Overview

**econcomplex** is a Python library that consolidates the best implementations of Economic Complexity and Regional Science indicators, combining functionalities from four reference packages in the literature:

- **EconGeo** (Balland, 2017) — R package focused on geographic and regional specialization indicators
- **economiccomplexity** (Vargas Sepúlveda, 2020) — R package with C++ backend (Armadillo) for complexity indicators
- **py-econcomplexity** (The Growth Lab at Harvard, 2020) — complete Python pipeline with time series support
- **py-economic-complexity** — modular Python implementation with Polars support and cross-space indicators

## Installation

```
1 # From PyPI (recommended)
2 pip install econcomplex
3
4 # Latest development version, directly from GitHub
5 pip install git+https://github.com/eltonfreitas/econcomplex.git
6
7 # Or, for local development, from the library root directory
8 pip install -e .
9
10 # Verify the installation
11 python3 -c "import econcomplex as ec; print(ec.__version__)"
```

Listing 1: Installation via pip

## Module Structure

Module	Contents
core	RCA, RPOP, Mcp, diversity, ubiquity, <code>trim_core</code> , <code>make_sample_data</code> , utilities
complexity	ECI/PCI — single entry point <code>eci_pci(method=...)</code> (eigenvector, reflections, fitness), subnational ECI
relatedness	Proximity (discrete, continuous, cosine, correlation), density, distance, co-occurrence, cross-space
specialization	LQ, Hachman, Krugman, specialization coef., similarity
inequality	Gini, locational Gini, Hoover-Gini, HHI, Shannon entropy
productivity	PRODY, EXPY, PGI, PEII
patents	Ease of Recombination, Modular Complexity
dynamics	Growth rates, entry/exit tracking (matrix and panel APIs)
outlook	COI, COG (Complexity Outlook)
optimization	ECI Optimization (Stojkoski & Hidalgo 2026), growth targeting, strategic diffusion (Alshamsi et al. 2018)
pipeline	<code>compute_complexity()</code> function — unified pipeline

## API Map: Entry Points, Advanced Implementations, and Aliases

The public API has three layers. **Entry points** are what most users need; **advanced implementations** are the method-specific functions the entry points delegate to (exposed for research use); **aliases** are alternative short names bound to the very same function (no duplicated code or computation).

Entry point	Delegates to (advanced)
<code>eci_pci(method=...)</code>	<code>eci_pci_eigenvector</code> , <code>method_of_reflections</code> , <code>fitness_complexity</code> (variants: <code>mor_region</code> , <code>mor_activities</code> )
<code>proximity(continuous=..., continuous_method=...)</code>	<code>continuous_proximity</code> ; <code>cosine_proximity</code> , <code>correlation_proximity</code> , <code>density_proximity</code> , <code>distance_proximity</code> , <code>coi_proximity</code> , <code>cog_proximity</code>
<code>compute_complexity()</code>	full per-period pipeline over <code>eci_pci</code> , <code>proximity</code> , <code>density</code> , <code>COI/COG</code>

Alias	Canonical function (same object)
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 (identical formula)

## Data Organisation

This section describes the input format expected by the library, ensuring that data is correctly structured before running any indicator.

### Standard Format: Long Table (*long format*)

The **econcomplex** library works exclusively with **tables in long format** (*tidy data*), where each row represents a **unique combination** of region  $\times$  activity (and period, when applicable):

region	activity	value	year (optional)
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
⋮	⋮	⋮	⋮

#### Warning

The library does **not** accept pivoted matrices (regions in rows, activities in columns). Use `pivot_to_matrix()` only for low-level functions that explicitly require the matrix.

### Data Quality Rules

1. **No duplicate rows:** each (*region*, *activity*) pair must appear exactly once per period.
2. **Non-negative values:** the value column must contain only numbers  $\geq 0$ .
3. **No NaN:** remove or impute missing values before passing the DataFrame.
4. **Consistent granularity:** all regions must use the same geographic level (e.g., all municipalities *or* all states).
5. **Homogeneous activities:** use a single classification per analysis (e.g., CNAE 2-digit *or* CNAE 4-digit, not mixed).

## Column Names — cols Dictionary

The pipeline uses a `cols` dictionary to map the actual column names of your DataFrame:

Key	Type	Description
"loc"	required	Region / location column
"act"	required	Activity / product / technology column
"val"	required	Numeric value column (jobs, exports ...)
"time"	optional	Period column (year, quarter ...)

```

1 import pandas as pd
2 import econcomplex as ec
3
4 df = pd.read_csv("my_data.csv")
5
6 # Check structure
7 print(df.dtypes)
8 print(df.shape)
9 print(df.isnull().sum())          # check for NaN
10 print(df.duplicated().sum())      # check for duplicates
11
12 # Basic cleaning
13 df = df.dropna(subset=["region", "cnae", "jobs"])
14 df = df[df["jobs"] >= 0]          # remove negative values
15 df = df.drop_duplicates(subset=["region", "cnae", "year"])
16
17 # Run the pipeline
18 result = ec.compute_complexity(
19     df,
20     cols={"loc": "region", "act": "cnae",
21           "val": "jobs", "time": "year"},
22 )

```

Listing 2: Verifying and preparing the DataFrame before the pipeline

## Supported File Formats

Format	pandas reader	Notes
.csv	pd.read_csv()	Most common; use <code>sep=";"</code> for Brazilian CSVs
.parquet	pd.read_parquet()	Ideal for large datasets (RAIS, COMEX)
.xlsx	pd.read_excel()	Requires <code>openpyxl</code> ; avoid >100 000 rows
.feather	pd.read_feather()	High performance; requires <code>pyarrow</code>
.dta	pd.read_stata()	Stata files; preserves labels

```

1 import pandas as pd
2
3 # CSV with semicolon separator (Brazilian standard)
4 df = pd.read_csv("rais.csv", sep=";", encoding="latin-1",
5                 usecols=["municipality", "cnae2d", "year", "jobs"],
6                 dtype={"municipality": str, "cnae2d": str})
7

```

```

8 # Parquet (preferred for RAIS/COMEX)
9 df = pd.read_parquet("comex_ncm.parquet",
10                     columns=["co_mun", "co_ncm", "year", "vl_fob"])
11
12 # Excel
13 df = pd.read_excel("data.xlsx", sheet_name="Sheet1",
14                   usecols=["state", "sector", "year", "jobs"])

```

Listing 3: Reading different file formats

## Examples of Compatible Datasets

### RAIS/MTE

Formal employment by municipality  $\times$  CNAE (2 or 4 digits)  $\times$  year. Value column: active employment bonds on 31/12.

### COMEX/MDIC

Exports by municipality  $\times$  NCM  $\times$  year. Value column: VL\_FOB (value in USD).

### INPI — Patents

Patent applications by State of the Federation  $\times$  IPC classification  $\times$  year.

### Atlas of Economic Complexity (Harvard)

Exports by country  $\times$  product (HS)  $\times$  year. Available at <https://atlas.cid.harvard.edu/>.

### Simulated data (testing)

Use `ec.make_sample_data(n_locs=20, nActs=50, seed=42)` to generate a synthetic DataFrame.

## How to Run the Scripts

This section describes all the ways to run the **econcomplex** library, from direct execution of Python scripts to use in interactive environments such as Jupyter and IDEs.

### Prerequisites

#### Warning

Make sure the dependencies are installed before running any script.

```

1 # Install the library (dependencies come automatically:
2 # numpy>=1.21, pandas>=1.3, scipy>=1.9)
3 pip install econcomplex
4
5 # Or in development mode
6 pip install -e /path/to/econcomplex
7
8 # Verify installation
9 python3 -c "import econcomplex as ec; print(ec.__version__)"

```

Listing 4: Installing the dependencies



## Direct Execution via Terminal

The simplest way to use the library is to run a `.py` script directly in the terminal:

```
1 # Navigate to the library folder
2 cd /path/to/econcomplex
3
4 # Run the full example script (guided tour of all indicators)
5 python3 examples/basic_usage.py
6
7 # Run your own script
8 python3 my_script.py
```

Listing 5: Running scripts in the terminal

If the library is **not** installed via `pip`, add the path manually at the beginning of each script:

```
1 import sys
2 sys.path.insert(0, "/path/to/econcomplex")
3
4 import econcomplex as ec
5 print(ec.__version__) # 1.0.0
```

Listing 6: Adding path to `sys.path` (without `pip` install)

## Running in Jupyter Notebook / JupyterLab

```
1 cd /path/to/my_project
2 jupyter notebook
3 # or
4 jupyter lab
```

Listing 7: Opening Jupyter from your project folder

With the library installed via `pip` the first cell is just the imports (use the `sys.path` fallback only if you have not installed it):

```
1 # Cell 1 -- imports
2 import econcomplex as ec
3 import numpy as np
4 import pandas as pd
5
6 print("econcomplex", ec.__version__, "loaded successfully.")
```

Listing 8: Configuration cell in Jupyter

```
1 # Cell 2 -- load data
2 df = pd.read_csv("my_data.csv") # columns: region, sector, jobs
3
4 # Cell 3 -- compute complexity
5 result = ec.compute_complexity(
6     df,
7     cols={"loc": "region", "act": "sector", "val": "jobs"},
8     method="eigenvector",
9     compute_coi_cog=True,
10 )
11 result.head()
12
```

```

13 # Cell 4 -- display ECI by region
14 eci = (result[["region", "eci"]]
15         .drop_duplicates()
16         .set_index("region")
17         .sort_values("eci", ascending=False))
18 print(eci)

```

Listing 9: Subsequent cells — normal usage

## Running in VS Code / PyCharm / Spyder

In any IDE, configure the **Python interpreter** and the **working directory** to point to the scripts folder.

```

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: Configure path in VS Code (settings.json)

In **Spyder**, go to *Tools* → *PYTHONPATH Manager* and add the `econcomplex/` directory.

## Minimal Execution Script

Ready-to-use template for any CSV-format dataset:

```

1 """
2 Usage template for the econcomplex library.
3 Adapt the column names and the CSV file path.
4 """
5 import pandas as pd
6 import econcomplex as ec
7
8 # 1. Load data
9 df = pd.read_csv("data.csv")
10 print("Columns:", df.columns.tolist())
11 print("Shape:", df.shape)
12
13 # 2. Compute all indicators
14 result = ec.compute_complexity(
15     df,
16     cols={
17         "loc": "region_column_name",      # e.g.: "municipality", "state"
18         "act": "activity_column_name",    # e.g.: "cnae", "product"
19         "val": "value_column_name",       # e.g.: "jobs", "exports"
20         "time": "year_column_name",       # optional
21     },
22     method="eigenvector",                # 'eigenvector' | 'reflections' | 'fitness'
23     threshold=1.0,
24     compute_coi_cog=True,
25 )
26
27 # 3. Export results
28 result.to_csv("complexity_results.csv", index=False)

```

```

29 print("Result saved to complexity_results.csv")
30 print("Generated columns:", sorted(result.columns.tolist()))
31
32 # 4. Individual indicators (optional)
33 mat = ec.pivot_to_matrix(df,
34                           "region_column_name",
35                           "activity_column_name",
36                           "value_column_name")
37
38 eci, pci = ec.eci_pci(mat)
39 hachman = ec.hachman_index(mat)
40 krugman = ec.krugman_index(mat)
41 entropy = ec.shannon_entropy(mat)
42
43 output = pd.DataFrame({
44     "eci": eci, "hachman": hachman,
45     "krugman": krugman, "entropy": entropy,
46 })
47 print(output.round(3))

```

Listing 11: Minimal template — copy and adapt

## Running with RAIS / CNAE Data (real use case)

```

1 import pandas as pd
2 import econcomplex as ec
3
4 # Load aggregated RAIS
5 # Expected columns: municipality, cnae2d, year, jobs
6 rais = pd.read_parquet("rais_municipality_cnae.parquet")
7
8 # Filter a single year for analysis
9 rais_2022 = rais[rais["year"] == 2022].copy()
10
11 # Pipeline
12 result = ec.compute_complexity(
13     rais_2022,
14     cols={"loc": "municipality", "act": "cnae2d", "val": "jobs"},
15     method="eigenvector",
16     compute_coi_cog=False, # disable COG for large datasets
17 )
18
19 # Top 20 most complex municipalities
20 top20 = (result[["municipality", "eci"]]
21         .drop_duplicates()
22         .sort_values("eci", ascending=False)
23         .head(20))
24 print(top20.to_string(index=False))
25
26 result.to_csv("complexity_municipalities_2022.csv", index=False)

```

Listing 12: Usage with RAIS data — municipalities x CNAE

## Running a Multi-Year Panel

To analyse the evolution of indicators over time, simply include the time column in the `cols` dictionary:

```

1 # rais with "year" column containing 2010, 2015, 2020
2 panel_result = ec.compute_complexity(
3     rais,
4     cols={
5         "loc": "municipality",
6         "act": "cnae2d",
7         "val": "jobs",
8         "time": "year",          # activates the temporal loop
9     },
10    method="reflections",
11    compute_coi_cog=False,
12 )
13
14 # The result already contains all years
15 print(panel_result["year"].value_counts().sort_index())
16
17 # Median ECI evolution by year
18 eci_trend = (panel_result[["year", "municipality", "eci"]]
19             .drop_duplicates()
20             .groupby("year")["eci"]
21             .median()
22             .round(4))
23 print(eci_trend)

```

Listing 13: Panel execution — multiple years

**Note**

The pipeline automatically iterates over each period, ensuring that the proximity matrix and complexity vectors are recalculated independently for each year.

## Core Indicators (core)

### Revealed Comparative Advantage (RCA)

Balassa's RCA is the main specialization filter:

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

```

1 import econcomplex as ec
2 import pandas as pd
3
4 df = pd.read_csv("data.csv")
5 mat = ec.pivot_to_matrix(df, "region", "sector", "jobs")
6
7 rca      = ec.rca(mat)          # continuous values
8 mcp      = ec.mcp(mat, threshold=1) # 1 if RCA >= 1, 0 otherwise
9 rca_rpop = ec.rpop(mat, pop)    # Puga & Turrini RCA (requires
                                # population vector)

```

Listing 14: Continuous and binary RCA

## Diversity and Ubiquity

```

1 div = ec.diversity(mat)      # number of sectors with RCA >= 1 per region
2 ubi = ec.ubiquity(mat)      # number of regions with RCA >= 1 per sector

```

Listing 15: Diversity and ubiquity

## Economic Complexity (complexity)

### Eigenvector Method (Hidalgo & Hausmann, 2009)

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

where  $\vec{v}_2$  is the second left eigenvector of matrix  $\tilde{M}$ :

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

```

1 eci, pci = ec.eci_pci(mat, method="eigenvector")
2 print(eci.head())    # Series indexed by region
3 print(pci.head())    # Series indexed by product/sector
4
5 # Pre-processing: units with zero diversity/ubiquity are trimmed
6 # automatically and returned as NaN. Disable with trim=False, or
7 # restrict to the well-connected core with dmin=2, umin=2 (sparse data).
8 core = ec.trim_core(mat, dmin=2, umin=2) # also available standalone

```

Listing 16: ECI and PCI using the eigenvector method

### Method of Reflections (Hidalgo & Hausmann, 2009)

```

1 eci_r, pci_r = ec.eci_pci(mat, method="reflections", iterations=20)
2 # tol=1e-10 stops the iterations early once the z-scored reflections
3 # stabilize; signs are oriented as in the eigenvector method
4 # (ECI correlates positively with diversity, PCI negatively
5 # with ubiquity).

```

Listing 17: Method of Reflections

### 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 is a cap (20, as in the R economiocomplexity package);
3 # the loop stops at convergence (tol) and warns if it is genuinely
4 # unstable. log_fitness=True returns the log scale recommended by
5 # Cristelli et al. (2015):
6 fit_log, comp_log = ec.fitness_complexity(mat, log_fitness=True)

```

Listing 18: Fitness-Complexity

## Subnational ECI

```
1 # pci_national: PCI computed from national/international data
2 eci_sub = ec.subnational_eci(mat_sub, pci_national)
```

Listing 19: Subnational ECI with external PCI

## Product Space and Relatedness (relatedness)

### Proximity ( $\phi$ )

$$\phi_{ij} = \min(P(\text{RCA}_i \geq 1 \mid \text{RCA}_j \geq 1), P(\text{RCA}_j \geq 1 \mid \text{RCA}_i \geq 1))$$

```
1 phi_d = ec.proximity(mat)["product"] # based on binary
    Mcp
2 phi_c = ec.proximity(mat, continuous=True)["product"] # based on
    continuous RCA
```

Listing 20: Discrete and continuous proximity

### Density and Distance

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

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

Listing 21: Density and distance

### Co-occurrence and Relatedness Indices

```
1 cooc = ec.co_occurrence(mat)
2 phi_cos = ec.cosine_proximity(mat) # cosine-based
3 phi_cor = ec.correlation_proximity(mat)
4 rca_rel = ec.relatedness(mat, phi_d) # relatedness densities
```

Listing 22: Co-occurrence and relatedness indices

### Cross-Space (relatedness between distinct classifications)

```
1 # phi_cross: proximity between sectors and technologies (different
    matrices)
2 phi_cross = ec.cross_space_proximity(mat_sector, mat_tech)
3 density_c = ec.cross_relatedness(mat_sector, phi_cross)
```

Listing 23: Cross-space proximity and relatedness

## Regional Specialization (specialization)

```

1 lq      = ec.location_quotient(mat)      # equivalent to RCA
2 hachman = ec.hachman_index(mat)          # 0 (diversified) to 1
   (specialized)
3 krugman = ec.krugman_index(mat)          # structural difference between
   regions
4 coef_spec = ec.spec_coefficient(mat)     # specialization coefficient
5 sim_exp  = ec.export_similarity(mat)     # productive structure
   similarity

```

Listing 24: Specialization indices

## Inequality and Concentration (inequality)

```

1 gini_r   = ec.gini(mat)                  # Gini of value distribution
2 gini_loc = ec.locational_gini(mat)       # locational Gini (by sector)
3 hoover   = ec.hoover_gini(mat)          # Hoover index
4 hhi      = ec.hhi(mat)                  # Herfindahl-Hirschman
5 entropy  = ec.shannon_entropy(mat)       # Shannon entropy (by region)

```

Listing 25: Inequality indicators

## Productivity (productivity)

```

1 prody = ec.prody(mat, gdp_per_capita)    # income associated with the
   product
2 expy  = ec.expy(mat, gdp_per_capita)     # sophistication of the export
   basket
3 pgi   = ec.pgi(mat, gini_vec)            # Product Gini Index (regional
   Gini vector)
4 peii  = ec.peii(mat, emissions)         # Product Emissions Intensity
   Index

```

Listing 26: Productivity and product inequality

## Patent-Based Complexity (patents)

```

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

```

Listing 27: Patent indicators (input: patent x technology matrix)

## Temporal Dynamics (dynamics)

```

1 growth = ec.growth_rates(df, "region", "sector", "jobs", "year")
2 entries = ec.entry_tracking(df, "region", "sector", "jobs", "year")
3 exits  = ec.exit_tracking(df, "region", "sector", "jobs", "year")

```

Listing 28: Growth rates and entry/exit tracking

## Complexity Outlook (outlook)

The **COI** measures the latent diversification potential; the **COG** measures the expected complexity gain:

$$\text{COI}_c = \sum_p (1 - M_{cp}) \omega_{cp} Q_p$$

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

Listing 29: COI and COG

## ECI Optimization (optimization)

A target-oriented optimization layer for strategic diversification (Stojkoski & Hidalgo, 2026, *Research Policy* 55, 105454). A stepping-stone forecast model is calibrated on a historical panel, the effort  $W_{cp}$  (the added RCA an economy needs to enter an activity) is obtained in closed form, and a 0–1 integer program selects the minimal-effort portfolio of new activities that reaches a target ECI.

```
1 # 1. Calibrate the stepping-stone model on a panel (t, t+5, t+10)
2 model = ec.calibrate_steppingstone(df, "region", "sector", "jobs",
3                                   "year", horizon=10, steppingstone=5)
4
5 # 2. Effort and no-policy forecast for the latest year
6 W = ec.effort_matrix(mat, model) # added RCA per candidate
7 fc = ec.forecast_specialization(mat, model) # rca, mcp, pci, eci
8
9 # 3. Minimal-effort portfolio raising each location's ECI by 0.1
10 portfolio = ec.eci_optimization(mat, model, delta_eci=0.1)
11
12 # 4. Optional: growth targeting - convert a growth target (3.5%/yr)
13 #    into an ECI target and optimize towards it
14 gm = ec.calibrate_growth_model(panel, "region", "year", "gdppc", "eci")
15 eci_star = ec.eci_target_for_growth(gm, 0.035, gdppc_now)
16 portfolio = ec.eci_optimization(mat, model, target_eci=eci_star)
```

Listing 30: ECI Optimization workflow

## Strategic Diffusion (optimization.diffusion)

Complex-contagion model of diversification on the network of related activities (Alshamsi, Pinheiro & Hidalgo, 2018, *Nature Communications* 9, 1328): the probability of entering an activity is  $p_i = B(\text{fraction of related activities present})^\alpha$ . The greedy strategy of always entering the most related activity is suboptimal; minimal diversification time requires targeting hubs in a narrow early window.

```
1 adj = ec.proximity_network(mat, phi_threshold=0.55) # product space
2 fit = ec.calibrate_contagion(df, "region", "sector", "jobs", "year",
3                             adjacency=adj) # B and alpha
4
5 active0 = ec.mcp(mat).loc["my_region"] # current portfolio
```



```

6 seq = ec.diversification_strategy(adj, active0, B=fit["B"],
7                                 alpha=fit["alpha"], strategy="greedy")
8 table = ec.compare_strategies(adj, active0, B=fit["B"],
9                                alpha=fit["alpha"])
10 best = ec.optimize_sequence(adj, active0, B=fit["B"],
11                             alpha=fit["alpha"])

```

Listing 31: Strategic diffusion workflow

## Unified Pipeline (compute\_complexity)

### Function Signature

```

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",   # 'rca' / 'rpop' / 'both' / 'manual'
9     threshold: float = 1.0,       # binarization threshold
10    iterations: int = 20,          # for reflections/fitness
11    proximity_type: str = "discrete", # 'discrete' / 'continuous'
12    proximity_method: str = "max",  # 'max' / 'sqrt' / 'min'
13    pop: pd.DataFrame | None = None, # population (for 'rpop'/'both')
14    compute_coi_cog: bool = True,
15    time_col: str | None = None,
16 ) -> pd.DataFrame:
17     """
18     Returns the original DataFrame enriched with columns:
19     rca, mcp, diversity, ubiquity, eci, pci,
20     density, distance, [coi, cog]
21     """

```

Listing 32: Pipeline function signature

## Parameters

Parameter	Type	Description
<code>data</code>	<code>DataFrame</code>	Table in long format
<code>cols</code>	<code>dict</code>	Column mapping (see Section 2.3)
<code>method</code>	<code>str</code>	Complexity method (single entry point)
<code>presence_test</code>	<code>str</code>	How to binarize Mcp (RCA, RPOP, both, manual)
<code>threshold</code>	<code>float</code>	Binarization threshold (default: 1.0)
<code>iterations</code>	<code>int</code>	Iterations for iterative methods (default: 20)
<code>proximity_type</code>	<code>str</code>	Discrete (co-occurrence) or continuous proximity
<code>proximity_method</code>	<code>str</code>	Proximity normalization
<code>pop</code>	<code>DataFrame</code>	Population, required for RPOP
<code>compute_coi_cog</code>	<code>bool</code>	Compute COI and COG (slower)

### Note

Degenerate units (zero diversity or ubiquity) are automatically trimmed by the complexity step and returned as `NaN` (see `trim_core`).

## Interpreting the Indicators

Quick-reference guide to reading the main outputs. All complexity indices are **relative measures**: compare them only within the same matrix (same period, same activity classification).

Indicator	Typical range	Reading
ECI / PCI	z-score (mean 0, sd 1)	Higher = more complex location / activity. NaN = unit trimmed for zero diversity/ubiquity.
Fitness / Complexity	$> 0$ (or log scale)	Non-linear analogue of ECI/PCI; ranking is the meaningful output.
RCA / LQ	$\geq 0$ ; 1 = neutral	$> 1$ : location specialized in the activity; basis of the binary matrix $M_{cp}$ .
Proximity $\phi$	$[0, 1]$	Probability-like similarity between activities; higher = easier joint specialization.
Density $\omega$	0–100 %	Share of the activities related to $c$ that the location already does; higher = easier entry.
Relative relatedness $\tilde{\omega}$	z-score	$> 0$ : activity more related than the average of the location's option set.
Distance	$[0, 1]$	$1 - \omega/100$ ; higher = harder entry.
COI	unbounded	Higher = more complex activities within reach (diversification potential).
COG	$\geq 0$ per activity	Gain in COI if the location develops the activity; identifies strategic bets.
Locational Gini	$[0, 0.5]$	Higher = activity geographically more concentrated.
HHI (normalized)	$[0, 1]$	Higher = location's portfolio more concentrated. Entropy reads the opposite way.
Hachman	$[0, 1]$	1 = location mirrors the aggregate structure.
Krugman	$[0, 2]$	Higher = structure more different from the aggregate.
PRODY / EXPY	units of the income vector	Income level associated with an activity / a location's basket.
Effort $W_{cp}$	added RCA	Lower = cheaper entry; $\leq 0$ means the forecast model already predicts entry.
Contagion $B, \alpha$	$B \in (0, 1], \alpha \approx 1$	$\alpha > 1$ : related activities reinforce each other (complex contagion).

### Warning

Inputs such as GDP per capita (PRODY/EXPY), population (RPOP), the regional Gini vector (PGI), and emissions (PEII) are **external data**: the library does not derive them from the matrix, and proxy variables produce indicator values without economic meaning.

## Complete Examples

### Example 1 — Regional Employment Analysis

```

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

```

```

4
5 # Generate synthetic data (or load a real CSV)
6 df = ec.make_sample_data(n_locs=50, n_acts=30, seed=42)
7 # Columns: loc, act, val
8
9 # Full pipeline
10 result = ec.compute_complexity(
11     df,
12     cols={"loc": "loc", "act": "act", "val": "val"},
13     method="eigenvector",
14     compute_coi_cog=True,
15 )
16
17 # Summary of generated indicators
18 print(result.columns.tolist())
19 # ['loc', 'act', 'val', 'rcá', 'mcp', 'diversity', 'ubiquity',
20 #  'eci', 'pci', 'density', 'distancé', 'coí', 'cog']
21
22 # Top 10 regions by ECI
23 top_eci = (result[["loc", "eci"]]
24             .drop_duplicates()
25             .sort_values("eci", ascending=False)
26             .head(10))
27 print(top_eci)
28
29 # Save
30 result.to_csv("regional_result.csv", index=False)

```

Listing 33: Complete regional employment analysis

## Example 2 — Exports with COMEX Data

```

1 import pandas as pd
2 import econcomplex as ec
3
4 # Load COMEX dataset aggregated by municipality x NCM x year
5 comex = pd.read_parquet("comex_mun_ncm.parquet")
6 comex_2023 = comex[comex["year"] == 2023].copy()
7
8 # Cleaning
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 result = 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 # Save
21 result.to_csv("export_complexity_2023.csv", index=False)
22 print("Mean ECI:", result["eci"].dropna().mean().round(4))

```

Listing 34: Export analysis with COMEX/MDIC data

### Example 3 — RAIS Time Panel (2010–2022)

```

1 import pandas as pd
2 import econcomplex as ec
3
4 rais = pd.read_parquet("rais_municipality_cnae_2010_2022.parquet")
5
6 # Temporal pipeline -- iterates over each year automatically
7 panel = ec.compute_complexity(
8     rais,
9     cols={
10         "loc": "municipality",
11         "act": "cnae2d",
12         "val": "jobs",
13         "time": "year",
14     },
15     method="eigenvector",
16     compute_coi_cog=False,
17 )
18
19 # ECI median evolution
20 eci_trend = (panel[["year", "municipality", "eci"]]
21             .drop_duplicates()
22             .groupby("year")["eci"]
23             .median()
24             .round(4))
25 print(eci_trend)
26
27 panel.to_csv("panel_complexity_2010_2022.csv", index=False)

```

Listing 35: Pipeline with panel data — multiple periods

### Example 4 — Diversification Targets (ECI Optimization)

Step-by-step use of the optimization layer with a multi-year panel (needs at least the periods  $t$ ,  $t + \tau$  and  $t + \Delta t$ ):

```

1 import pandas as pd
2 import econcomplex as ec
3
4 # 1. Panel with at least t, t+5, t+10 (e.g. 2005, 2010, 2015)
5 rais = pd.read_parquet("rais_municipality_cnae_2005_2015.parquet")
6
7 # 2. Calibrate the stepping-stone forecast model
8 model = ec.calibrate_steppingstone(
9     rais, "municipality", "cnae2d", "jobs", "year",
10     horizon=10, steppingstone=5,
11 )
12
13 # 3. Matrix of the latest year and minimal-effort portfolios
14 mat = ec.pivot_to_matrix(rais[rais["year"] == 2015],
15                          "municipality", "cnae2d", "jobs")
16 portfolio = ec.eci_optimization(mat, model, delta_eci=0.1)
17 print(portfolio.head()) # location, activity, effort, pci, targets
18
19 # 4. Optional: growth targeting (needs GDP per capita and ECI panel)
20 gm = ec.calibrate_growth_model(macro, "municipality", "year",
21                                "gdppc", "eci", horizon=10)

```

```

22 eci_star = ec.eci_target_for_growth(gm, 0.035, gdppc_now)
23 portfolio = ec.eci_optimization(mat, model, target_eci=eci_star)
24
25 # 5. Optional: diversification timing (strategic diffusion)
26 adj = ec.proximity_network(mat, phi_threshold=0.55)
27 fit = ec.calibrate_contagion(rais, "municipality", "cnae2d",
28                             "jobs", "year", adjacency=adj)
29 active0 = ec.mcp(mat).loc["my_municipality"]
30 best = ec.optimize_sequence(adj, active0,
31                             B=fit["B"], alpha=fit["alpha"])
32 print(best["sequence"][:10], best["improvement"])

```

Listing 36: ECI Optimization end to end

## API Reference

Complete inventory of the 87 public functions, grouped by module. Signatures show parameter names and defaults; see the in-code docstrings (`help(ec.function)`) for full parameter documentation. This section is auto-generated from the installed package by `docs/generate_api_reference.py`.

### core — Core

Function	Description
<code>rca(mat, binary=False, threshold=1.0)</code>	Revealed Comparative Advantage (Balassa Index).
<code>rpop(mat, pop, binary=False, threshold=1.0)</code>	Population-normalized RCA (RPOP).
<code>mcp(mat, presence_test='rca', rca_threshold=1.0, rpop_threshold=1.0, pop=None)</code>	Binary presence matrix (Mcp).
<code>diversity(mat, use_rca=True, threshold=1.0)</code>	Diversity: number of activities in which each region has RCA $\geq$ threshold.
<code>ubiquity(mat, use_rca=True, threshold=1.0)</code>	Ubiquity: number of regions where each activity has RCA $\geq$ threshold.
<code>normalized_ubiquity(mat, threshold=1.0)</code>	Normalized ubiquity: ratio of activity's share of total value to its share of ubiquity.
<code>pivot_to_matrix(df, index, columns, values, fill_value=0.0)</code>	Convert long-format DataFrame to wide (pivot) matrix.
<code>melt_matrix(mat, index_name='location', columns_name='activity', values_name='value')</code>	Convert wide matrix to long-format DataFrame.
<code>binarize(mat, threshold=1.0)</code>	Return binary matrix: 1 where $\text{mat} \geq \text{threshold}$ , else 0.
<code>normalize_zscore(vec)</code>	Z-score normalize a 1-D vector.
<code>normalize_01(vec)</code>	Min-max normalize a 1-D vector to $[0, 1]$ .
<code>make_sample_data(n_locs=50, n_acts=30, seed=42, loc_col='loc', act_col='act', val_col='val')</code>	Generate synthetic long-format data for examples and testing.
<code>trim_core(mat, dmin=1, umin=1, use_rca=True, threshold=1.0, max_iter=100)</code>	Iteratively prune a value matrix to its (dmin, umin)-core.

## complexity — Complexity

Function	Description
<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>	Economic Complexity Index (ECI) and Product Complexity Index (PCI).
<code>eci_pci_eigenvector(mat, use_rca=True, threshold=1.0)</code>	ECI and PCI via the eigenvector method (advanced implementation; prefer <code>'eci_pci(mat, method="eigenvector")'</code> , which adds the automatic trimming of degenerate units).
<code>method_of_reflections(mat, use_rca=True, threshold=1.0, iterations=20, return_both=True, tol=1e-10)</code>	Method of Reflections (iterative) to compute ECI and PCI.
<code>mor_regions(mat, use_rca=True, threshold=1.0, steps=20)</code>	Method of Reflections – region/country side only.
<code>mor_activities(mat, use_rca=True, threshold=1.0, steps=19)</code>	Method of Reflections – activity/technology side only.
<code>fitness_complexity(mat, use_rca=True, threshold=1.0, iterations=20, extremality=1.0, tol=1e-10, log_fitness=False)</code>	Fitness-Complexity algorithm.
<code>subnational_eci(mat, pci_external, use_rca=True, threshold=1.0, standardize=True)</code>	Subnational ECI using an externally supplied PCI vector.

## relatedness — Relatedness and Product Space

Function	Description
<code>proximity(mat, use_rca=True, threshold=1.0, method='max', compute='both', continuous=False, continuous_method='correlation')</code>	Compute product and/or location proximity matrices.
<code>continuous_proximity(rca_mat, method='correlation')</code>	Continuous product proximity from a (continuous) RCA matrix.
<code>cosine_proximity(mat, use_rca=True)</code>	Shortcut for <code>'continuous_proximity(rca(mat), method='cosine')'</code> : cosine similarity between the RCA vectors of each pair of activities.
<code>correlation_proximity(mat, use_rca=True)</code>	Shortcut for <code>'continuous_proximity(rca(mat), method='correlation')'</code> : Pearson correlation between RCA vectors, rescaled to [0, 1].
<code>relatedness_density(mat, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Relatedness density for each (region, activity) pair.

Function	Description
<code>density(mat, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Alias of <code>relatedness_density</code> .
<code>relatedness(mat, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Alias of <code>relatedness_density</code> .
<code>distance(mat, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Distance $(1 - \text{density}/100)$ .
<code>relatedness_density_internal(mat, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Internal relatedness density: density values for activities the region ALREADY has ( $M_{rc} = 1$ ).
<code>relatedness_density_external(mat, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	External relatedness density: density values for activities the region does NOT yet have ( $M_{rc} = 0$ ).
<code>relative_relatedness(mat, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Relative relatedness (Pinheiro et al.
<code>co_occurrence(mat, diagonal=False)</code>	Co-occurrence matrix: number of times two activities appear together across regions/events.
<code>relatedness_index(mat, method='cosine', diagonal=False)</code>	Pairwise relatedness index between activities from co-occurrence.
<code>z_score_novelty(incidence)</code>	Z-score of technological novelty (atypicality of co-occurrence).
<code>cross_proximity(mat_a, mat_b, use_rca=True, threshold=1.0)</code>	Cross-space proximity between activity space A and activity space B.
<code>cross_relatedness(mat_a, x_phi, use_rca=True, threshold=1.0)</code>	Cross-space relatedness density.
<code>cross_space_proximity(mat_a, mat_b, use_rca=True, threshold=1.0)</code>	Alias of <code>cross_proximity</code> .

## specialization — Regional Specialization

Function	Description
<code>location_quotient(mat, binary=False, threshold=1.0)</code>	Location Quotient (identical to RCA / Balassa Index).
<code>location_quotient_avg(mat)</code>	Weighted average LQ per region (Coefficient of Specialization, Hoover 1985).
<code>hachman_index(mat)</code>	Hachman Index (structural similarity to national economy).
<code>specialization_coefficient(mat)</code>	Hoover Coefficient of Specialization.
<code>spec_coefficient(mat)</code>	Alias of <code>specialization_coefficient</code> .
<code>krugman_index(mat)</code>	Krugman Specialization Index.
<code>export_similarity(mat, use_rca=True, epsilon=0.1, log=True)</code>	Export Similarity Index (Bahar et al.



## inequality — Inequality and Concentration

Function	Description
<code>gini(x)</code>	Standard Gini coefficient.
<code>locational_gini(mat)</code>	Krugman Locational Gini per activity (column).
<code>hoover_gini(mat, pop=None)</code>	Hoover-Gini: Gini using population as horizontal axis (Hoover curve).
<code>herfindahl(mat, normalize=True)</code>	Herfindahl-Hirschman Index (HHI) per region.
<code>hhi(mat, normalize=True)</code>	Alias of <code>herfindahl</code> .
<code>shannon_entropy(mat, base=2.0)</code>	Shannon entropy per region.
<code>hoover_index(mat, pop=None)</code>	Hoover Index (Robin Hood Index) per activity.

## productivity — Productivity

Function	Description
<code>prody(mat, gdp, use_rca=True)</code>	PRODY: Income / productivity level of each activity.
<code>expy(mat, gdp, use_rca=True)</code>	EXPY: Income level of a region's export / activity portfolio.
<code>product_gini_index(mat, gini_vec, threshold=1.0)</code>	Product Gini Index (PGI): inequality embedded in a product.
<code>pgi(mat, gini_vec, threshold=1.0)</code>	Alias of <code>product_gini_index</code> .
<code>product_emissions_index(mat, emissions, threshold=1.0)</code>	Product Emissions Intensity Index (PEII).
<code>peii(mat, emissions, threshold=1.0)</code>	Alias of <code>product_emissions_index</code> .

## patents — Patent-Based Complexity

Function	Description
<code>ease_of_recombination(incidence)</code>	Ease of Recombination (EOR) for each technology/class.
<code>modular_complexity(incidence, eor=None)</code>	Modular Complexity of each patent.
<code>modular_complexity_avg(incidence, eor=None)</code>	Average Modular Complexity aggregated per technology (column).

## dynamics — Temporal Dynamics

Function	Description
<code>growth_rate(mat1, mat2, axis=0, pct=True)</code>	Growth rate between two time periods.

Function	Description
<code>growth_matrix(mat1, mat2, pct=True)</code>	Element-wise growth rate matrix.
<code>growth_rates(df, loc, act, val, time, axis=0, pct=True)</code>	Long-format wrapper around ‘growth_rate’ for panel data.
<code>entry(mats, use_rca=True, threshold=1.0)</code>	Industry entry matrix: 1 when a (region, activity) transitions from absent (M=0) to present (M=1) between consecutive time periods.
<code>exit(mats, use_rca=True, threshold=1.0)</code>	Industry exit matrix: 1 when a (region, activity) transitions from present (M=1) to absent (M=0) between consecutive time periods.
<code>entry_exit_summary(mats, use_rca=True, threshold=1.0)</code>	Summary of entry and exit events per (region, activity) pair.
<code>entry_tracking(df, loc, act, val, time, use_rca=True, threshold=1.0)</code>	Long-format wrapper around ‘entry’ for panel data.
<code>exit_tracking(df, loc, act, val, time, use_rca=True, threshold=1.0)</code>	Long-format wrapper around ‘exit’ for panel data.

## outlook — Complexity Outlook

Function	Description
<code>complexity_outlook_index(mat, pci, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Complexity Outlook Index (COI).
<code>complexity_outlook_gain(mat, pci, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Complexity Outlook Gain (COG).
<code>coi(mat, pci, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Alias of <code>complexity_outlook_index</code> .
<code>cog(mat, pci, phi=None, use_rca=True, threshold=1.0, proximity_method='max')</code>	Alias of <code>complexity_outlook_gain</code> .

## optimization — ECI Optimization and Strategic Diffusion

Function	Description
<code>calibrate_steppingstone(df, loc, act, val, time, horizon=10, steppingstone=5, threshold=1.0, proximity_method='max')</code>	Calibrate the stepping-stone forecast model (eq.
<code>effort_matrix(mat, model)</code>	Effort $W_{cp}$ : the added RCA an economy must reach by the steppingstone period for the calibrated model to predict entry ( $RCA = 1$ ) at the horizon (eq.

Function	Description
<code>forecast_specialization(mat, model)</code>	No-policy baseline forecast ( $W = 0$ ): project RCA at $t+\text{horizon}$ with the calibrated stepping-stone model, using entry coefficients for cells with $\text{RCA} < \text{threshold}$ and exit coefficients otherwise.
<code>eci_optimization(mat, model, delta_eci=0.1, target_eci=None, locations=None, solver='milp')</code>	ECI Optimization (Stojkoski & Hidalgo 2026): identify, per location, the minimal-effort portfolio of new specializations that raises the projected ECI to a target.
<code>calibrate_growth_model(df, loc, time, gdppc, eci, horizon=10)</code> <code>expected_growth(model, eci, gdppc_now, period=None)</code>	Calibrate the panel growth regression (eq. Annualized log growth rate implied by the calibrated model for a given ECI and current GDP per capita (using the most recent period's fixed effect unless 'period' is given).
<code>eci_target_for_growth(model, growth_target, gdppc_now, period=None)</code>	Invert the growth regression to find the ECI compatible with a target annualized log growth rate (e.g.
<code>proximity_network(mat, phi=None, phi_threshold=0.55, method='max')</code> <code>activation_probabilities(adjacency, active, B=1.0, alpha=1.0)</code>	Binary network of related activities: $a_{pp'} = 1$ if $\phi_{pp'} \geq \text{threshold}$ . Activation probability of each inactive activity (eq.
<code>calibrate_contagion(df, loc, act, val, time, adjacency=None, phi_threshold=0.55, threshold=1.0, presence_test='rca', n_bins=20)</code> <code>diversification_strategy(adjacency, active0, B=1.0, alpha=1.0, strategy='greedy', seed=None)</code>	Empirically calibrate B and alpha of the contagion model (eq. Generate a diversification sequence with one of the heuristic strategies of Alshamsi et al.
<code>expected_diversification_time(adjacency, active0, sequence, B=1.0, alpha=1.0)</code> <code>compare_strategies(adjacency, active0, B=1.0, alpha=1.0, n_random=10, seed=0)</code>	Expected total time of a fixed sequence of targets (activity labels). Expected total diversification time of each heuristic strategy ('random' averaged over 'n_random' runs).
<code>optimize_sequence(adjacency, active0, B=1.0, alpha=1.0, n_iter=2000, seed=0)</code>	Approximate the optimal diversification sequence by simulated annealing over target orderings (pairwise-swap proposals), starting from the greedy sequence.

## pipeline — Unified Pipeline

Function	Description
<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, time_col=None)</code>	Compute a full suite of economic complexity indicators.

## References

### 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-economy: 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.

## License and Citation

---

This library is distributed under the MIT license. For use in academic work, please cite the library (<https://github.com/eltonfreitas/econcomplex>) together with the original papers of the indicators used (see References).

```
1 import econcomplex as ec
2 print(ec.__version__)    # 1.0.0
```

Listing 37: Check the library version