# Package: econio (via r-universe)

July 14, 2026

**Type** Package

**Title** Input-Output Analysis

**Version** 0.1.0

**Description** Provides a set of functions for input-output analysis.

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**Imports** broom, cli, dibble (>= 0.3.1.9000), dplyr, ggplot2, glue, lifecycle, MASS, pillar, purrr, rlang (>= 1.1.0), scales, tibble, tidyr, tidyselect, vctrs

**Suggests** econiodatajp, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**Depends** R (>= 4.1)

**Roxygen** list(markdown = TRUE)

**Remotes** uchidamizuki/dibble, uchidamizuki/econiodatajp, uchidamizuki/tarchives

**Config/roxygen2/version** 8.0.0

**Config/pak/sysreqs** libicu-dev

**Repository** https://uchidamizuki.r-universe.dev

**Date/Publication** 2026-07-06 23:57:23 UTC

**RemoteUrl** https://github.com/UchidaMizuki/econio

**RemoteRef** HEAD

**RemoteSha** a71dc08ea23da25c041e3b9e0e92c9c4bfccc768

## Contents

---

`io_check_axes` *Check input-output table axes*

---

## Description

Check input-output table axes

## Usage

```
io_check_axes(data)
```

## Arguments

| | |
|---|---|
| `data` | An input-output table. |

## Value

An `econ_io_table` object.

## io_check_totals *Check input-output table totals*

### Description

Check input-output table totals

### Usage

```
io_check_totals(data, total_tolerance = .Machine$double.eps^0.5)
```

### Arguments

| | |
|---|---|
| data | An input-output table. |
| total_tolerance | A numeric. The tolerance for the total check. By default, .Machine$double.eps^0.5. |

### Value

An econ_io_table object.

## io_ghosh_inverse *Ghosh inverse matrix*

### Description

Ghosh inverse matrix

### Usage

```
io_ghosh_inverse(data, open_economy = NULL)
```

### Arguments

| | |
|---|---|
| data | An econ_io_table object. |
| open_economy | A scalar logical. If TRUE, open economy assumptions are used. |

### Value

An econ_io_table object of the Ghosh inverse matrix.

`io_import_coef` *Import coefficients*

## Description

Calculate import coefficients for input-output tables. The calculation and interpretation differ based on the import type (competitive vs noncompetitive) and the specified axis.

## Usage

```
io_import_coef(data, axis = c("input", "output"))
```

## Arguments

| | |
|---|---|
| `data` | An `econ_io_table` object. |
| `axis` | A scalar character. Either `"input"` (default) or `"output"`, specifying whether to calculate import coefficients by input industry or by output category. |

## Details

For **noncompetitive import** tables, imports are recorded as a separate input row. When `axis = "input"`, all coefficients are zero (imports are not allocated to industries as inputs). When `axis = "output"`, coefficients represent the import share of regional demand: `import / regional_demand` for same-region demand, zero for cross-region flows.

For **competitive import** tables, imports are recorded as a separate output column. When `axis = "input"`, coefficients represent the import share of regional demand (negative values): `-import / regional_demand` for same-region demand. When `axis = "output"`, all coefficients are zero.

The axis parameter determines whether the coefficient varies by input industry (axis = "input") or output industry/final demand (axis = "output").

## Value

An `econ_io_table` object of import coefficients. The dimensions depend on `axis` and the table type.

## Examples

```
## Not run:
# Noncompetitive import table
io_import_coef(iotable_noncomp, axis = "input")  # all zeros
io_import_coef(iotable_noncomp, axis = "output") # import shares by output

# Competitive import table
io_import_coef(iotable_comp, axis = "input")  # import shares by input
io_import_coef(iotable_comp, axis = "output") # all zeros

## End(Not run)
```

`io_input_coef` *Input coefficients*

## Description

Calculate the industry-by-industry input coefficient matrix, also known as the technical coefficient matrix or A matrix.

## Usage

```
io_input_coef(data, open_economy = NULL)
```

## Arguments

| | |
|---|---|
| `data` | An `econ_io_table` object. |
| `open_economy` | A scalar logical. If `TRUE`, open economy assumptions are used (competitive import tables only). Must be `NULL` for noncompetitive import tables. |

## Details

Each element `a[i, j]` represents the amount of input from industry `i` required to produce one unit of output from industry `j`. The coefficient is calculated as the ratio of intermediate input to total input (gross output).

For competitive import type tables, `open_economy` controls whether to adjust the coefficients for import competition. When `open_economy = TRUE`, the coefficients are adjusted to exclude imports: `a[i, j] - m[i] * a[i, j]` where `m[i]` is the import coefficient for industry `i` in the same region. This gives the domestic (non-import) input coefficient used in open economy Leontief models. For noncompetitive import tables, `open_economy` must be `NULL` as imports are already separated.

## Value

An `econ_io_table` object of input coefficients with dimensions `c("input", "output")`.

## Examples

```
## Not run:
# Noncompetitive import table
io_input_coef(iotable_noncomp)

# Competitive import table, closed economy
io_input_coef(iotable_comp, open_economy = FALSE)

# Competitive import table, open economy (domestic coefficients)
io_input_coef(iotable_comp, open_economy = TRUE)

## End(Not run)
```

## io_leontief_inverse *Leontief inverse matrix*

### Description

Leontief inverse matrix

### Usage

```
io_leontief_inverse(data, open_economy = NULL)
```

### Arguments

| | |
|---|---|
| data | An econ_io_table object. |
| open_economy | A scalar logical. If TRUE, open economy assumptions are used. |

### Value

An econ_io_table object of the Leontief inverse matrix.

## io_output_coef *Output coefficients*

### Description

Calculate the industry-by-industry output coefficient matrix, also known as the allocation coefficient matrix or B matrix, used in the Ghosh model.

### Usage

```
io_output_coef(data, open_economy = NULL)
```

### Arguments

| | |
|---|---|
| data | An econ_io_table object. |
| open_economy | A scalar logical. If TRUE, open economy assumptions are used (competitive import tables only). Must be NULL for noncompetitive import tables. |

### Details

Each element b[i, j] represents the proportion of industry i's output that is sold to industry j. The coefficient is calculated as the ratio of intermediate output to total output (gross output).

For competitive import type tables, open_economy controls the denominator calculation. When open_economy = TRUE, the denominator is adjusted to total_output - (total_export - total_import) to represent domestic production sold domestically. When open_economy = FALSE, the denominator is simply total output. For noncompetitive import tables, open_economy must be NULL.

## Value

An econ_io_table object of output coefficients with dimensions c("input", "output").

## Examples

```
## Not run:
# Noncompetitive import table
io_output_coef(iotable_noncomp)

# Competitive import table, closed economy
io_output_coef(iotable_comp, open_economy = FALSE)

# Competitive import table, open economy
io_output_coef(iotable_comp, open_economy = TRUE)

## End(Not run)
```

---

# io_reclass *Reclassify an input-output table*

---

## Description

Reclassify an input-output table

## Usage

```
io_reclass(
  data,
  input_region_data = NULL,
  input_sector_data = NULL,
  output_region_data = NULL,
  output_sector_data = NULL,
  from_col = "from",
  to_col = "to",
  weight_col = "weight",
  weight_tolerance = .Machine$double.eps^0.5,
  check_axes = TRUE
)
```

## Arguments

| | |
|---|---|
| data | An input-output table. |
| input_region_data, input_sector_data, output_region_data, output_sector_data | Data frames used to reclassify the input and output regions and sectors. If NULL, no reclassification is performed for that axis. By default, NULL. |

`from_col`, `to_col`, `weight_col`
: Names of the columns in the reclassification data. Defaults are `"from"`, `"to"`, and `"weight"`.

`weight_tolerance`
: Tolerance for checking that weights sum to 1. By default, `.Machine$double.eps^0.5`.

`check_axes`
: A scalar logical. If `TRUE`, the input and output axes are checked to be identical. By default, `TRUE`.

## Value

A reclassified input-output table.

---

`io_sector_name` *Get sector names*

---

## Description

Get sector names

## Usage

```
io_sector_name(x)
```

## Arguments

`x`
: An `econ_io_sector` object or a data frame with a column named `sector`.

## Value

A vector of sector names.

---

`io_sector_type` *Get sector types*

---

## Description

Get sector types

## Usage

```
io_sector_type(x)
```

## Arguments

`x`
: An `econ_io_sector` object or a data frame with a column named `sector`.

## Value

A vector of sector types.

io_skyline *Tidy, layer, and plot an input-output table*

## Description

Methods for generics::tidy(), ggplot2::autolayer(), and ggplot2::autoplot() that summarise an econ_io_table for visualization.

## Usage

```
## S3 method for class 'econ_io_table'
tidy(x, type, ...)

## S3 method for class 'econ_io_table'
autolayer(object, type, ...)

## S3 method for class 'econ_io_table'
autoplot(object, type, ...)
```

## Arguments

| | |
|---|---|
| x, object | An econ_io_table object. |
| type | A scalar character. The type of summary. Currently only "skyline" is supported. |
| ... | Additional arguments passed to the underlying summary. For autolayer(), this includes geom ("rect" (default) or "segment") and any further arguments passed to the underlying geometry. |

## Details

Currently only type = "skyline" is supported, which produces a *skyline chart* (a.k.a. an industry-by-industry diagram of production inducement). For each industry the chart shows, as a stacked bar:

- the width: the total production induced by final demand and exports, and
- the height: the self-sufficiency rate and the import rate, expressed as a share of domestic final demand.

The skyline chart is only defined for competitive import type tables (created with io_table_to_competitive_import() or io_table_regional()/io_table_multiregional() with competitive_import = TRUE).

- tidy() returns the underlying data in a tidy tibble::tibble() (one row per industry and rate type), suitable for custom plots.
- autolayer() returns **ggplot2** layers to add to an existing plot. Pass geom = "rect" (default) for the skyline bars or geom = "segment" for the self-sufficiency outline.
- autoplot() returns a complete **ggplot2** skyline chart.

## Value

- `tidy()`: a `tibble::tibble()`.
- `autolayer()`: a list of **ggplot2** layers.
- `autoplot()`: a `ggplot2::ggplot()` object.

## Examples

```
## Not run:
# `iotable` is a competitive import type `econ_io_table`.
library(ggplot2)

# Tidy data for a custom plot.
tidy(iotable, type = "skyline")

# A complete skyline chart.
autoplot(iotable, type = "skyline")

# Or build one up from layers.
ggplot() +
  autolayer(iotable, type = "skyline", geom = "rect")

## End(Not run)
```

---

# io_spectral_embedding *Laplacian spectral embedding*

---

## Description

**[Experimental]**

## Usage

```
io_spectral_embedding(data, dims = 1)
```

## Arguments

| | |
|---|---|
| `data` | An `econ_io_table` object. |
| `dims` | A scalar positive integer giving the number of embedding dimensions (the number of smallest non-zero Laplacian eigenvectors to return). By default, 1, i.e. the Fiedler vector. |

## Details

Embeds the industries of the domestic production network into a low-dimensional space by minimizing the graph Dirichlet energy

$$\sum_{ij} w_{ij}(h_i - h_j)^2 = h^\top \Lambda h,$$

subject to `h` being orthonormal and orthogonal to the constant vector. The minimizers are the eigenvectors of the symmetric graph Laplacian $\Lambda = \mathrm{diag}(u) - (W + W^\top)$ associated with the smallest non-zero eigenvalues, where `u` is the total weight (in plus out) of each node. With `dims = 1` this is the **Fiedler vector**.

Like `io_trophic_level()`, the table is first converted to a noncompetitive import type table with `io_table_to_noncompetitive_import()` so that the industry-by-industry block holds purely domestic intermediate transactions, which are treated as the edge weights `w[i, j]`.

The embedding is **undirected**: it uses only the symmetrized weights `W + Wt` and therefore captures industry clusters but not the direction of production. It is complementary to the directed `io_trophic_level()`; pairing the two gives a map of the production network with a clustering coordinate (this function) and a flow level (`io_trophic_level()`).

The sign of each eigenvector is arbitrary, so it is fixed deterministically by orienting each component to make its largest-magnitude entry positive.

## Value

A `dibble`. With `dims = 1`, one value per industry. With `dims > 1`, an `industry` by `component` array.

## See Also

`io_trophic_level()`

## Examples

```
## Not run:
io_spectral_embedding(iotable)
io_spectral_embedding(iotable, dims = 2)

## End(Not run)
```

---

# io_streamness *Upstreamness, downstreamness, and supply chain length*

---

## Description

Sector-level summaries of how far each industry is from final demand (upstreamness) and from primary inputs (downstreamness), derived from the Ghosh and Leontief inverse matrices respectively.

## Usage

```
io_upstreamness(data, open_economy = NULL, normalize = FALSE)

io_downstreamness(data, open_economy = NULL, normalize = FALSE)

io_streamness_length(data, open_economy = NULL, normalize = FALSE)
```

## Arguments

| | |
|---|---|
| data | An econ_io_table object. |
| open_economy | A scalar logical passed to io_leontief_inverse() / io_ghosh_inverse(). If TRUE, open economy assumptions are used. |
| normalize | A scalar logical. If TRUE, the measure is rescaled to have a mean of 1 across sectors. By default, FALSE. |

## Details

io_upstreamness() returns the row sums of the Ghosh inverse, i.e. the total forward linkage (also called *output upstreamness*). A larger value means the industry sells more of its output, directly and indirectly, to other industries rather than to final demand.

io_downstreamness() returns the column sums of the Leontief inverse. This is identical to the **output multiplier** (the simple, Type I multiplier) and to the **backward linkage** (also called *input downstreamness*). A larger value means a unit of final demand for the industry pulls more total output, directly and indirectly, from the rest of the economy.

io_streamness_length() returns the sum of io_upstreamness() and io_downstreamness(). Following Antràs and Chor (2018), this sum measures the full length of the supply chain passing through the industry, i.e. the total number of production stages from pure value added to final consumption.

When normalize = TRUE the measure is rescaled so that its mean across sectors equals 1. The normalized io_downstreamness() is the **power of dispersion** (a backward-linkage index). The normalized io_upstreamness() is the Ghosh-based forward-linkage index; note that this is **not** the same as the classical *sensitivity of dispersion*, which is based on the row sums of the Leontief (not Ghosh) inverse. When normalize = TRUE, io_streamness_length() has a mean of 2 across sectors.

## Value

An econ_io_table object with one value per industry.

## References

Antràs, P. and Chor, D. (2018). On the measurement of upstreamness and downstreamness in global value chains. NBER Working Paper No. 24185.

## See Also

io_streamness_position()

## Examples

```
## Not run:
# `iotable` is a competitive import type `econ_io_table`.
io_upstreamness(iotable)
io_downstreamness(iotable) # equals the output multiplier
io_downstreamness(iotable, normalize = TRUE) # power of dispersion
io_streamness_length(iotable) # total number of production stages

## End(Not run)
```

---

# io_streamness_position

*Supply chain position*

---

## Description

**[Experimental]**

## Usage

```
io_streamness_position(
  data,
  open_economy = NULL,
  normalize = FALSE,
  type = c("difference", "relative", "log_ratio")
)
```

## Arguments

| | |
|---|---|
| data | An econ_io_table object. |
| open_economy | A scalar logical passed to io_leontief_inverse() / io_ghosh_inverse(). If TRUE, open economy assumptions are used. |
| normalize | A scalar logical. If TRUE, the measure is rescaled to have a mean of 1 across sectors. By default, FALSE. |
| type | One of "difference", "relative", or "log_ratio". See Details. |

## Details

Net position of each industry within the supply chain, derived from io_upstreamness() (U) and io_downstreamness() (D). A positive value indicates that the industry sits relatively further upstream (closer to primary inputs); a negative value indicates it sits relatively further downstream (closer to final demand).

type selects the combination:

- "difference" (default): U − D.

- `"relative"`: `(U - D) / (U + D)`, bounded in `[-1, 1]`.
- `"log_ratio"`: `log(U / D)`, scale-robust.

Unlike `io_streamness_length()`, this is not a single named index from a specific paper; these are natural arithmetic complements to `io_upstreamness()` and `io_downstreamness()`, provided as an experimental convenience. The exact formulation(s) may change in future releases.

When `normalize = TRUE`, `U` and `D` are each rescaled to a mean of 1 before combining, so `type = "difference"` has a mean of 0 across industries: a relative-position measure centered on the average.

## Value

An `econ_io_table` object with one value per industry.

## See Also

`io_streamness_length()`

## Examples

```
## Not run:
io_streamness_position(iotable)
io_streamness_position(iotable, normalize = TRUE) # mean 0 across industries
io_streamness_position(iotable, type = "relative")
io_streamness_position(iotable, type = "log_ratio")

## End(Not run)
```

---

`io_table_multiregional` *Create a multi-regional input-output table*

---

## Description

Create a multi-regional input-output table

## Usage

```
io_table_multiregional(
  data,
  input_cols = c("input_region", "input_sector_type", "input_sector_name"),
  output_cols = c("output_region", "output_sector_type", "output_sector_name"),
  competitive_import = NULL,
  total_tolerance = .Machine$double.eps^0.5,
  check_axes = TRUE
)
```

### Arguments

| | |
|---|---|
| data | A data frame. |
| input_cols | <tidy-select> Input region, sector type and name columns. |
| output_cols | <tidy-select> Output region, sector type and name columns. |
| competitive_import | A scalar logical. If TRUE, the table is assumed to be a competitive import type. |
| total_tolerance | A numeric. The tolerance for the total check. By default, .Machine$double.eps^0.5. |
| check_axes | A scalar logical. If TRUE, the input and output axes are checked to be identical. By default, TRUE. |

### Value

An econ_io_table object.

---

## io_table_regional *Create a regional input-output table*

---

### Description

Create a regional input-output table

### Usage

```
io_table_regional(
  data,
  input_cols = c("input_sector_type", "input_sector_name"),
  output_cols = c("output_sector_type", "output_sector_name"),
  competitive_import = NULL,
  total_tolerance = .Machine$double.eps^0.5,
  check_axes = TRUE
)
```

### Arguments

| | |
|---|---|
| data | A data frame. |
| input_cols | <tidy-select> Input sector type and name columns. |
| output_cols | <tidy-select> Output sector type and name columns. |
| competitive_import | A scalar logical. If TRUE, the table is assumed to be a competitive import type. |
| total_tolerance | A numeric. The tolerance for the total check. By default, .Machine$double.eps^0.5. |
| check_axes | A scalar logical. If TRUE, the input and output axes are checked to be identical. By default, TRUE. |

## Value

An `econ_io_table` object.

---

`io_table_to_competitive_import` *Convert an input-output table to a competitive import type table*

---

## Description

Convert an input-output table to a competitive import type table

## Usage

```
io_table_to_competitive_import(
  data,
  data_import = NULL,
  import_sector_name = NA_character_,
  import_total_tolerance = .Machine$double.eps^0.5
)
```

## Arguments

| | |
|---|---|
| `data` | An input-output table. |
| `data_import` | An optional import table. By default, `NULL`. |
| `import_sector_name` | A name for the import sector. By default, `NA_character_`. |
| `import_total_tolerance` | A numeric tolerance for checking import totals. By default, `.Machine$double.eps^0.5`. |

## Value

A competitive import type input-output table.

---

`io_table_to_noncompetitive_import` *Convert an input-output table to a noncompetitive import type table*

---

## Description

Convert an input-output table to a noncompetitive import type table

### Usage

```
io_table_to_noncompetitive_import(
  data,
  data_import = NULL,
  import_sector_name = NA_character_,
  import_total_tolerance = .Machine$double.eps^0.5
)
```

### Arguments

| | |
|---|---|
| `data` | An input-output table. |
| `data_import` | An optional import table. By default, `NULL`. |
| `import_sector_name` | A name for the import sector. By default, `NA_character_`. |
| `import_total_tolerance` | A numeric tolerance for checking import totals. By default, `.Machine$double.eps^0.5`. |

### Value

A noncompetitive import type input-output table.

---

## `io_table_to_regional` *Convert a multiregional input-output table to a regional input-output table*

---

### Description

Convert a multiregional input-output table to a regional input-output table

### Usage

```
io_table_to_regional(
  data,
  export_sector_name = NA_character_,
  import_sector_name = NA_character_
)
```

### Arguments

| | |
|---|---|
| `data` | A multiregional input-output table. |
| `export_sector_name` | A name for the export sector. By default, `NA_character_`. |
| `import_sector_name` | A name for the import sector. By default, `NA_character_`. |

### Value

A list of regional input-output tables.

## io_total_input *Get total input*

### Description

Get total input

### Usage

```
io_total_input(
  data,
  input_sector_type = c("industry", "import", "value_added"),
  output_sector_type = "industry",
  same_region = FALSE
)
```

### Arguments

| | |
|---|---|
| data | An econ_io_table object. |
| input_sector_type | A character vector of input sector types. |
| output_sector_type | A character vector of output sector types. |
| same_region | A logical scalar. If TRUE, values between different regions are ignored. |

### Value

An econ_io_table object of total input.

## io_total_output *Get total output*

### Description

Get total output

### Usage

```
io_total_output(
  data,
  input_sector_type = "industry",
  output_sector_type = c("industry", "final_demand", "export", "import"),
  same_region = FALSE
)
```

### Arguments

| | |
|---|---|
| `data` | An `econ_io_table` object. |
| `input_sector_type` | A character vector of input sector types. |
| `output_sector_type` | A character vector of output sector types. |
| `same_region` | A logical scalar. If `TRUE`, values between different regions are ignored. |

### Value

An `econ_io_table` object of total output.

---

## `io_trophic` *Trophic levels and trophic incoherence*

---

### Description

Network measures of where each industry sits in the domestic production hierarchy, following the directed-network trophic level of MacKay, Johnson and Sansom (2020).

### Usage

```
io_trophic_level(data)

io_trophic_incoherence(data)
```

### Arguments

| | |
|---|---|
| `data` | An `econ_io_table` object. |

### Details

The table is first converted to a noncompetitive import type table with `io_table_to_noncompetitive_import()` so that the industry-by-industry block holds purely domestic intermediate transactions. These transactions are treated as a weighted directed network `w[i, j]` (the flow from industry `i` to industry `j`), and the trophic level `h` solves the Laplacian system

$$(\mathrm{diag}(u) - (W + W^{\top}))\, h = v,$$

where `u` is the total weight (in plus out) of each node and `v` is its imbalance (in minus out). The levels are shifted so that the minimum is 0.

Value-added and final-demand sectors are not nodes and do not receive a trophic level, but they are **not** ignored: because the table balances, an industry's imbalance equals `(final demand + exports) - (value added + imp` Primary, value-added-heavy industries therefore receive low trophic levels and final-demand-facing industries receive high ones (the level increases with distance from primary inputs).

`io_trophic_incoherence()` returns the trophic incoherence

$$F_0 = \frac{\sum_{ij} w_{ij}(h_j - h_i - 1)^2}{\sum_{ij} w_{ij}},$$

a scalar in `[0, 1]`-ish range that is 0 when the network is perfectly coherent (every edge spans exactly one level) and larger when production contains more cycles and feedback.

## Value

- `io_trophic_level()`: a `dibble` with one trophic level per industry.
- `io_trophic_incoherence()`: a scalar numeric.

## References

MacKay, R. S., Johnson, S. and Sansom, B. (2020). How directed is a directed network? *Royal Society Open Science*, 7(9), 201138.

## Examples

```
## Not run:
io_trophic_level(iotable)
io_trophic_incoherence(iotable)

## End(Not run)
```

# Index