Package: whep (via r-universe)

June 8, 2026

Title Processing Agro-Environmental Data

Version 0.3.0.9000

Description A set of tools for processing and analyzing data developed in the context of the ``Who Has Eaten the Planet'' (WHEP) project, funded by the European Research Council (ERC). For more details on multi-regional input–output model ``Food and Agriculture Biomass Input–Output'' (FABIO) see Bruckner et al. (2019) <doi:10.1021/acs.est.9b03554>.

License MIT + file LICENSE

Imports arrow, cli, data.table, dplyr, fs, FAOSTAT, httr, Matrix, methods, nanoparquet, pins, purrr, rappdirs, readr, rlang, stringr, tibble, tidyr, withr, yaml, zoo

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Suggests ggplot2, googlesheets4, here, htmltools, jsonlite, knitr, pointblank, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

URL https://eduaguilera.github.io/whep/, https://github.com/eduaguilera/whep

BugReports https://github.com/eduaguilera/whep/issues

Depends R (>= 3.5)

LazyData true

Repository https://eduaguilera.r-universe.dev

Date/Publication 2026-06-08 14:18:05 UTC

RemoteUrl https://github.com/eduaguilera/whep

RemoteRef HEAD

RemoteSha 5e06cfe120aff86fba2d39a81bfe3a0afa1b1375

Contents

`add_area_code` *Get area codes from area names*

Description

Add a new column to an existing tibble with the corresponding code for each name. The codes are assumed to be from those defined by the `FABIO` model.

Usage

```
add_area_code(table, name_column = "area_name", code_column = "area_code")
```

Arguments

`table`	The table that will be modified with a new column.
`name_column`	The name of the column in `table` containing the names.
`code_column`	The name of the output column containing the codes.

Value

A tibble with all the contents of `table` and an extra column named `code_column`, which contains the codes. If there is no code match, an `NA` is included.

Examples

```
table <- tibble::tibble(
  area_name = c("Armenia", "Afghanistan", "Dummy Country", "Albania")
)

add_area_code(table)

table |>
  dplyr::rename(my_area_name = area_name) |>
  add_area_code(name_column = "my_area_name")

add_area_code(table, code_column = "my_custom_code")
```

---

add_area_name *Get area names from area codes*

Description

Add a new column to an existing tibble with the corresponding name for each code. The codes are assumed to be from those defined by the `FABIO` model, which them themselves come from `FAOSTAT` internal codes. Equivalences with ISO 3166-1 numeric can be found in the *Area Codes* CSV from the zip file that can be downloaded from FAOSTAT. TODO: Think about this, would be nice to use ISO3 codes but won't be enough for our periods.

Usage

```
add_area_name(table, code_column = "area_code", name_column = "area_name")
```

Arguments

`table`	The table that will be modified with a new column.
`code_column`	The name of the column in `table` containing the codes.
`name_column`	The name of the output column containing the names.

Value

A tibble with all the contents of `table` and an extra column named `name_column`, which contains the names. If there is no name match, an `NA` is included.

Examples

```
table <- tibble::tibble(area_code = c(1, 2, 4444, 3))

add_area_name(table)

table |>
  dplyr::rename(my_area_code = area_code) |>
  add_area_name(code_column = "my_area_code")

add_area_name(table, name_column = "my_custom_name")
```

`add_footprint_product_stage` *Add a final-demand product-area stage to footprints.*

Description

Split footprint rows by the area that supplied the final-demand product, using shares from `y_mat`. This preserves the standard footprint totals while adding a FABIO-viewer style phase: origin product -> product/supplier area -> product -> final-demand area.

This is a compact global-view helper. It does not recompute the full origin-sector by product-sector Leontief cube. Instead, each existing footprint row is allocated over the product-area shares observed in final demand for the same `target_area`, `target_fd`, and `target_item`.

Usage

```
add_footprint_product_stage(
  footprints,
  y_mat,
  labels,
  fd_labels,
  max_product_areas = 5,
  other_area_name = "Other",
  min_share = 0
)
```

Arguments

`footprints`	Footprint table from `compute_footprint()` with `target_area`, `target_item`, `target_fd`, and `value`.
`y_mat`	Final demand matrix from `build_io_model()`.
`labels`	Tibble mapping Y rows to `area_code` and `item_cbs_code`.
`fd_labels`	Tibble mapping Y columns to `area_code` and `fd_col`.
`max_product_areas`	Maximum number of supplier/product areas to keep separately for each final-demand area, item, and demand category. Smaller supplier areas are grouped into `other_area_name`.

other_area_name
: Label for grouped supplier/product areas.

min_share
: Drop split paths smaller than this percentage of the total input footprint value. Use 0 to keep all split paths.

Value

`footprints` with `product_area`, `product_area_name`, `product_item`, and `product_share` columns. `value` is replaced by the split path value.

`add_item_cbs_code` *Get commodity balance sheet item codes from item names*

Description

Add a new column to an existing tibble with the corresponding code for each commodity balance sheet item name. The codes are assumed to be from those defined by FAOSTAT.

Usage

```
add_item_cbs_code(
  table,
  name_column = "item_cbs_name",
  code_column = "item_cbs_code"
)
```

Arguments

`table`	The table that will be modified with a new column.
`name_column`	The name of the column in `table` containing the names.
`code_column`	The name of the output column containing the codes.

Value

A tibble with all the contents of `table` and an extra column named `code_column`, which contains the codes. If there is no code match, an `NA` is included.

Examples

```
table <- tibble::tibble(
  item_cbs_name = c("Cottonseed", "Eggs", "Dummy Item")
)
add_item_cbs_code(table)

table |>
  dplyr::rename(my_item_cbs_name = item_cbs_name) |>
  add_item_cbs_code(name_column = "my_item_cbs_name")

add_item_cbs_code(table, code_column = "my_custom_code")
```

`add_item_cbs_name` *Get commodity balance sheet item names from item codes*

Description

Add a new column to an existing tibble with the corresponding name for each commodity balance sheet item code. The codes are assumed to be from those defined by FAOSTAT.

Usage

```
add_item_cbs_name(
  table,
  code_column = "item_cbs_code",
  name_column = "item_cbs_name"
)
```

Arguments

`table`	The table that will be modified with a new column.
`code_column`	The name of the column in `table` containing the codes.
`name_column`	The name of the output column containing the names.

Value

A tibble with all the contents of `table` and an extra column named `name_column`, which contains the names. If there is no name match, an `NA` is included.

Examples

```
table <- tibble::tibble(item_cbs_code = c(2559, 2744, 9876))
add_item_cbs_name(table)

table |>
  dplyr::rename(my_item_cbs_code = item_cbs_code) |>
  add_item_cbs_name(code_column = "my_item_cbs_code")

add_item_cbs_name(table, name_column = "my_custom_name")
```

`add_item_prod_code` *Get production item codes from item names*

Description

Add a new column to an existing tibble with the corresponding code for each production item name. The codes are assumed to be from those defined by FAOSTAT.

Usage

```
add_item_prod_code(
  table,
  name_column = "item_prod_name",
  code_column = "item_prod_code"
)
```

Arguments

`table`	The table that will be modified with a new column.
`name_column`	The name of the column in `table` containing the names.
`code_column`	The name of the output column containing the codes.

Value

A tibble with all the contents of `table` and an extra column named `code_column`, which contains the codes. If there is no code match, an `NA` is included.

Examples

```
table <- tibble::tibble(
  item_prod_name = c("Rice", "Cabbages", "Dummy Item")
)
add_item_prod_code(table)

table |>
  dplyr::rename(my_item_prod_name = item_prod_name) |>
  add_item_prod_code(name_column = "my_item_prod_name")

add_item_prod_code(table, code_column = "my_custom_code")
```

`add_item_prod_name` *Get production item names from item codes*

Description

Add a new column to an existing tibble with the corresponding name for each production item code. The codes are assumed to be from those defined by FAOSTAT.

Usage

```
add_item_prod_name(
  table,
  code_column = "item_prod_code",
  name_column = "item_prod_name"
)
```

Arguments

`table`	The table that will be modified with a new column.
`code_column`	The name of the column in `table` containing the codes.
`name_column`	The name of the output column containing the names.

Value

A tibble with all the contents of `table` and an extra column named `name_column`, which contains the names. If there is no name match, an `NA` is included.

Examples

```
table <- tibble::tibble(item_prod_code = c(27, 358, 12345))
add_item_prod_name(table)

table |>
  dplyr::rename(my_item_prod_code = item_prod_code) |>
  add_item_prod_name(code_column = "my_item_prod_code")

add_item_prod_name(table, name_column = "my_custom_name")
```

`animals_codes` *Animal codes and classifications*

Description

Maps live animal CBS items to their livestock classifications, process codes, and associated product items used in livestock modeling.

Usage

```
animals_codes
```

Format

A tibble where each row corresponds to one live animal CBS item. It contains the following columns:

- `Item_Code`: Numeric FAOSTAT item code for the live animal.
- `item_cbs`: Name of the CBS item (e.g., `"Cattle"`, `"Asses"`).
- `proc_code`: Short process code used internally (e.g., `"p092"`).
- `proc`: Descriptive process name (e.g., `"Asses"`).
- `item_cbs_code`: Numeric CBS item code (often equal to `Item_Code`).
- `Farm_class`: Broad farm classification grouping the animal. One of `"Cattle"`, `"Dairy_cows"`, `"Monogastric"`, `"Sheep_goats"`, `"Bees"`, `"Game"`.
- `Item_product`: Name of the primary product derived from this animal, if applicable (e.g., milk for dairy cows).
- `Item_Code_product`: Numeric FAOSTAT code for the associated product item.
- `Liv_prod_cat`: Livestock product category the animal belongs to.
- `Graniv_grazers`: Broad feeding behaviour classification. One of `"Grazers"`, `"Granivores"`, `"Bees"`, `"Game"`.
- `Livestock_name`: Internal livestock identifier used across datasets (e.g., `"Cattle"`, `"Dairy_cows"`, `"Asses"`).
- `Animal_class`: Fine-grained animal class, including production type distinctions (e.g., `"Broilers"`, `"Hens"`, `"Hogs"`, `"Dairy_cows"`).
- `Item_FAOmanure`: Name of the corresponding FAOSTAT manure management item.
- `Item_Code_FAOmanure`: Numeric code of the FAOSTAT manure management item.
- `Cat_Labour`: Labour category used in labour-related analyses. One of `"Cattle"`, `"Equines"`, `"Dairy_cows"`, `"Birds"`, `"Small_ruminants"`, `"Pigs"`, `"Bees"`.
- `Cat_FAO1`: Top-level FAO category. Currently always `"Animal"`.
- `item_bouwman`: Item name used in Bouwman et al. livestock datasets.

Source

Derived from FAOSTAT data and internal livestock classification work.

Examples

```
head(animals_codes)
```

`biomass_coefs` *Biomass coefficients for crops and livestock products*

Description

Provides dry-matter, nutrient, and energy conversion coefficients for agricultural products and residues. Used to convert fresh-matter production quantities into biomass flows, nutrient budgets, and energy content.

Usage

```
biomass_coefs
```

Format

A tibble where each row corresponds to one product or item. It contains 68 columns:

- `Code`: Item code (character), corresponding to FAOSTAT production codes.
- `Name_biomass`: Item name as used in biomass accounting.
- `Equiv`: Reference equivalence item used when coefficients are borrowed from another similar commodity (e.g., `"Wheat"` for oats).
- `Category`: Broad commodity category (e.g., `"Cereals, other"`, `"Barley"`, `"Vegetables"`).
- `BG_Biomass_kgDM_ha`: Below-ground biomass in kg dry matter per hectare.
- `Root_Shoot_ratio`: Ratio of root to aerial biomass (dimensionless).
- `Product_kgDM_kgFM`: Product dry-matter content in kg DM per kg fresh matter.
- `Residue_kgDM_kgFM`: Residue dry-matter content in kg DM per kg fresh matter of product.
- `Conventional_kgDM_ha`: Conventional yield in kg dry matter per hectare.
- `Organic_kgDM_ha`: Organic yield in kg dry matter per hectare.
- `GE_product_edible_portion_MJ_kgFM`: Gross energy of the edible portion in MJ per kg fresh matter.
- `GE_product_residue_MJ_kgFM`: Gross energy of the residue in MJ per kg fresh matter (may be character due to source formatting).
- `GE_product_MJ_kgFM`: Gross energy of the whole product in MJ per kg fresh matter.
- `GE_residue_MJ_kg`: Gross energy of the residue in MJ per kg.
- `kg_product_kg_aerial_biomass`: Fraction of aerial biomass that is product (harvest index, kg/kg).

- kg_residue_kg_aerial_biomass_FM: Fraction of aerial biomass that is residue, on fresh matter basis.
- kg_residue_kg_product_FM: Ratio of residue to product on fresh matter basis.
- Carcass_to_LW: Carcass-to-live-weight ratio (livestock only; logical placeholder for crop items).
- Edible_portion: Edible fraction of the product (kg edible / kg fresh matter).
- N_kgN_kgFM: Nitrogen content in kg N per kg fresh matter.
- Lipids_g_kgFM: Lipid content in g per kg fresh matter.
- Carbohydrates_g_kgFM: Carbohydrate content in g per kg fresh matter.
- Calcium_mg_kgFM: Calcium content in mg per kg fresh matter.
- VitaminA_microg_kgFM: Vitamin A content in micrograms per kg fresh matter.
- Edible_kgDM_kgFM: Edible dry matter in kg per kg fresh matter.
- Edible_kgC_kgFM: Edible carbon in kg C per kg fresh matter.
- Edible_N_kgFM: Edible nitrogen in kg N per kg fresh matter.
- Edible_kgP_kgFM: Edible phosphorus in kg P per kg fresh matter.
- Edible_K_kgFM: Edible potassium in kg K per kg fresh matter.
- NonEdible_kgDM_kgFM: Non-edible dry matter in kg per kg fresh matter.
- NonEdible_kgC_kgFM: Non-edible carbon in kg C per kg fresh matter.
- NonEdible_kgN_kgFM: Non-edible nitrogen in kg N per kg fresh matter.
- NonEdible_kgP_kgFM: Non-edible phosphorus in kg P per kg fresh matter.
- NonEdible_kgK_kgFM: Non-edible potassium in kg K per kg fresh matter.
- Product_kgN_kgDM: Nitrogen content of product in kg N per kg dry matter.
- Product_kgP_kgDM: Phosphorus content of product in kg P per kg dry matter.
- Product_kgK_kgDM: Potassium content of product in kg K per kg dry matter.
- Product_kgC_kgDM: Carbon content of product in kg C per kg dry matter.
- Residue_kgN_kgDM: Nitrogen content of residue in kg N per kg dry matter.
- Residue_kgP_kgDM: Phosphorus content of residue in kg P per kg dry matter.
- Residue_kgK_kgDM: Potassium content of residue in kg K per kg dry matter.
- Residue_kgC_kgDM: Carbon content of residue in kg C per kg dry matter.
- Residue_humified_kgC_kgC: Humification coefficient of residue carbon (fraction of residue C stabilised as soil organic matter).
- MgDM_m3: Megagrams dry matter per cubic metre (bulk density proxy).
- Root_kgC_kgDM: Carbon content of roots in kg C per kg root dry matter.
- Root_humified_kgC_kgC: Humification coefficient for root carbon.
- Root_mass_kgC_kgDM: Root carbon mass in kg C per kg crop dry matter.
- Rhizodeposits_mass_kgC_kgDM: Rhizodeposit carbon in kg C per kg crop dry matter.
- Residue_C_N: Carbon-to-nitrogen ratio of the residue.
- Root_kgN_kgDM: Nitrogen content of roots in kg N per kg root dry matter.

- GE_Roots_MJ_kgDM: Gross energy of roots in MJ per kg dry matter.
- Rhizodeposits_N_kgN_kgRootN: Rhizodeposit nitrogen as a fraction of root nitrogen.
- Fiber_g_kgFM: Dietary fibre content in g per kg fresh matter.
- SFA_g_kgFM: Saturated fatty acid content in g per kg fresh matter.
- MUFA_g_kgFM: Monounsaturated fatty acid content in g per kg fresh matter.
- PUFA_g_kgFM: Polyunsaturated fatty acid content in g per kg fresh matter.
- PUFA_n3_g_kgFM: Omega-3 PUFA content in g per kg fresh matter.
- Iron_mg_kgFM: Iron content in mg per kg fresh matter.
- Zinc_mg_kgFM: Zinc content in mg per kg fresh matter.
- Magnesium_mg_kgFM: Magnesium content in mg per kg fresh matter.
- Cadmium_microg_kgFM: Cadmium content in micrograms per kg fresh matter.
- VitaminB12_microg_kgFM: Vitamin B12 content in micrograms per kg fresh matter.
- VitaminD_microg_kgFM: Vitamin D content in micrograms per kg fresh matter.
- Folate_microg_kgFM: Folate content in micrograms per kg fresh matter.
- VitaminC_mg_kgFM: Vitamin C content in mg per kg fresh matter.
- VitaminE_mg_kgFM: Vitamin E content in mg per kg fresh matter.
- Flavonoids_mg_kgFM: Flavonoid content in mg per kg fresh matter.
- Carotenoids_mg_kgFM: Carotenoid content in mg per kg fresh matter.

Source

Compiled from multiple sources including FAO food composition data, crop physiology literature, and IPCC Tier 1 coefficients.

Examples

```
head(biomass_coefs)
```

---

build_cbs_prices *Build CBS item prices*

Description

Compute prices for all commodity balance sheet items, including processed products and crop residues. Prices are derived from trade data, with special handling for items without direct trade prices (palm kernels, soy hulls, brans, etc.). Crop residue prices are estimated as a fraction of the product price.

Usage

```
build_cbs_prices(
  cbs,
  trade_prices = NULL,
  residue_price_factor = 0.1,
  example = FALSE
)
```

Arguments

cbs	A tibble of commodity balance sheets, as returned by build_commodity_balances() or get_wide_cbs().
trade_prices	A tibble as returned by build_trade_prices(). If NULL, it is computed internally.
residue_price_factor	Numeric. Relative price of crop residues compared to the product. Default 0.1.
example	Logical. If TRUE, return a small example tibble. Default FALSE.

Value

A tibble with columns:

- year: Integer year.
- element: "import" or "export".
- item_cbs_code: Numeric CBS item code.
- price: Price in KDollars per tonne.

Examples

```
build_cbs_prices(example = TRUE)
```

---

build_commodity_balances

Build commodity balance sheets

Description

Construct commodity balance sheets (CBS) from raw FAOSTAT data. This is a convenience wrapper that chains the three pipeline steps:

1. .read_cbs() — read & reformat FAOSTAT CBS data.
2. .fix_cbs() — processing calibration, trade imputation, destiny filling, and final balancing.
3. .qc_cbs() — flag data-quality anomalies.

Usage

```
build_commodity_balances(
  primary_all,
  start_year = 1850,
  end_year = 2023,
  smooth_carry_forward = FALSE,
  example = FALSE,
  .fixed_data = NULL
)
```

Arguments

primary_all	A tibble of primary production, as returned by build_primary_production().
start_year	Integer. First year to include. Default 1850.
end_year	Integer. Last year to include. Default 2023.
smooth_carry_forward	Logical. If TRUE, carry-forward tails are replaced with a linear trend. Default FALSE.
example	Logical. If TRUE, return a small hardcoded example tibble instead of reading remote data. Default FALSE.
.fixed_data	Optional tibble with the same structure as the output of the internal .read_cbs() |> .fix_cbs() steps. When supplied, primary_all is ignored and the pipeline skips directly to .qc_cbs(). Default NULL.

Value

A tibble in long format with columns: year, area_code, item_cbs_code, element (e.g. "production", "import", "food"), value, source, fao_flag.

Examples

```
build_commodity_balances(example = TRUE)
```

---

build_detailed_trade *Build detailed bilateral trade matrix*

Description

Construct the detailed bilateral trade matrix (DTM) from the FAOSTAT Detailed Trade Matrix pin. Reports trade flows between pairs of countries with their trade shares, aggregated to polity level and mapped to CBS item codes.

Optionally extends the time series by joining with commodity balance sheet years and gap-filling country shares via linear interpolation.

Usage

```
build_detailed_trade(
  raw_trade = NULL,
  cbs = NULL,
  min_share = 1e-04,
  extend_time = FALSE,
  example = FALSE
)
```

Arguments

raw_trade	A data.table or tibble of raw FAOSTAT bilateral trade data. If NULL (default), the data is read from the "faostat-trade-bilateral" pin.
cbs	A tibble of commodity balance sheets in wide format, as returned by build_commodity_balances() or get_wide_cbs(). Required when extend_time = TRUE.
min_share	Numeric. Partners with a country share below this threshold are dropped when extending time. Default 0.0001.
extend_time	Logical. If TRUE, extend the time series using CBS years and linear interpolation of country shares. Default FALSE.
example	Logical. If TRUE, return a small example tibble without downloading remote data. Default FALSE.

Value

A tibble with columns:

- year: Integer year.
- area_code: Numeric polity code of the reporter country.
- area_code_partner: Numeric polity code of the partner country.
- element: Either "import" or "export".
- item_cbs_code: Numeric CBS item code.
- unit: Measurement unit ("tonnes" or "heads").
- value: Trade quantity.
- country_share: Share of total trade for this partner.

Examples

```
build_detailed_trade(example = TRUE)
```

`build_gridded_landuse` *Build gridded landuse dataset*

Description

Disaggregate country-level FAOSTAT crop harvested areas to a 0.5-degree grid. This reproduces the core spatialization workflow of the LandInG toolbox, adapted to WHEP conventions and tidy data structures.

The algorithm follows three main steps:

1. Each crop's country total is distributed to grid cells proportionally to a spatial reference pattern (e.g. Monfreda) weighted by gridded cropland extent (e.g. LUH2/HYDE).
2. If total allocated harvested area in any cell exceeds its capacity (cropland times multi-cropping suitability), excess is iteratively redistributed using a logit-based transformation.
3. Individual crops are aggregated into crop functional types (CFTs).

Usage

```
build_gridded_landuse(
  country_areas,
  crop_patterns,
  gridded_cropland,
  country_grid,
  config = list()
)
```

Arguments

`country_areas` A tibble with country-level crop harvested areas. Expected columns:

- `year`: Integer year.
- `area_code`: Country code (numeric, matching WHEP polities).
- `item_prod_code`: FAOSTAT item code for the crop.
- `harvested_area_ha`: Total harvested area in hectares.
- `irrigated_area_ha`: Irrigated harvested area in hectares (optional, defaults to 0).

`crop_patterns` A tibble with per-cell spatial crop patterns. Expected columns:

- `lon`: Longitude of cell centre.
- `lat`: Latitude of cell centre.
- `item_prod_code`: FAOSTAT item code.
- `harvest_fraction`: Cropping intensity (Monfreda area divided by reference cropland).

`gridded_cropland` A tibble with per-cell cropland extent. Expected columns:

- `lon`: Longitude of cell centre.

- `lat`: Latitude of cell centre.
- `year`: Integer year.
- `cropland_ha`: Total cropland area in hectares.
- `irrigated_ha`: Irrigated cropland in hectares (optional, defaults to 0).

`country_grid` A tibble mapping grid cells to countries. Expected columns:

- `lon`: Longitude of cell centre.
- `lat`: Latitude of cell centre.
- `area_code`: Country code. Optional columns:
- `cell_area_frac` (or `area_frac`): Fraction of the physical cell belonging to this polity compartment. Defaults to 1.
- `polycell_id`, `cell_id`: Stable compartment/cell identifiers preserved in outputs when present.
- `year` or validity intervals (`valid_from`/`valid_to`, `start_year`/`end_year`, `from_year`/`to_year`) for historical, time-varying polity overlays.

`config` Named list of optional extras. Unknown keys raise an error. Recognised keys:

- `years`: Integer vector of years to spatialize. If `NULL` (default), all years present in `country_areas` are processed. When supplied, `country_areas`, `gridded_cropland`, and `type_cropland` are filtered to this set before processing.
- `cft_mapping`: A tibble mapping FAOSTAT items to CFT names (`item_prod_code`, `cft_name`). If `NULL`, no CFT aggregation is performed and individual crop results are returned.
- `type_cropland`: A tibble with per-cell, per-year, per-type cropland (`lon`, `lat`, `year`, `luh2_type`, `type_ha`, `type_irrig_ha`). When provided alongside `type_mapping`, each crop is allocated only into cells containing its LUH2 type. If `NULL`, falls back to total cropland.
- `type_mapping`: A tibble (`item_prod_code`, `luh2_type`) that maps each crop to its LUH2 type. If `NULL`, type-aware allocation is disabled even when `type_cropland` is provided.
- `multicropping`: A tibble with per-cell multi-cropping suitability factors. Required columns: `lon`, `lat`, `mc_rainfed`, `mc_irrigated`. An optional `year` column keys factors to year (one row per cell per year); when present, the table is filtered to the current year before the capacity constraint is applied. When absent, the table is treated as a static spatial layer applied to every year. If `NULL` (default), the capacity constraint still runs with `mc_rainfed` = `mc_irrigated` = `1` (harvested area capped at physical cropland).
- `max_iterations`: Maximum iterations for the redistribution loop. Default: `1000L`.
- `expansion_threshold`: Iteration number after which crops are allowed to expand into cells without an existing pattern. Default: `100L`.

Value

A tibble with gridded crop (or CFT) harvested areas. Columns:

- `lon`, `lat`: Cell centre coordinates.
- `year`: Integer year.
- `area_code`: WHEP polity code for this cell compartment.
- `polycell_id`, `cell_id`: Preserved when supplied in `country_grid`.
- `crop_name` or `cft_name`: Crop or CFT identifier.
- `rainfed_ha`: Rainfed harvested area in the cell.
- `irrigated_ha`: Irrigated harvested area in the cell.

Methodology

This function reimplements the spatial crop allocation from the LandInG toolbox (Ostberg et al. 2023, doi:10.5194/gmd-16-3375-2023) with the following extensions:

- LUH2 crop-functional-type constraints (`type_cropland` + `type_mapping` parameters) restrict each crop to cells containing its LUH2 type (c3ann, c4ann, c3per, c3nfx). LandInG allocates to total cropland without type constraints.
- MIRCA2000 crop-specific irrigated fractions (Portmann et al. 2010) for irrigation distribution, falling back to LUH2-proportional allocation.

Data sources

- Country areas: FAOSTAT QCL via `build_primary_production`
- Crop patterns: EarthStat / Monfreda et al. (2008)
- Gridded cropland: LUH2 v2h (Hurtt et al. 2020)
- Irrigation: MIRCA2000 (Portmann et al. 2010) + LUH2

Examples

```
# Minimal example with toy data
country_areas <- tibble::tribble(
  ~year, ~area_code, ~item_prod_code, ~harvested_area_ha,
  2000L, 1L, 15L, 1000
)
crop_patterns <- tibble::tribble(
  ~lon, ~lat, ~item_prod_code, ~harvest_fraction,
  0.25, 50.25, 15L, 0.6,
  0.75, 50.25, 15L, 0.4
)
gridded_cropland <- tibble::tribble(
  ~lon, ~lat, ~year, ~cropland_ha,
  0.25, 50.25, 2000L, 800,
  0.75, 50.25, 2000L, 500
)
country_grid <- tibble::tribble(
  ~lon, ~lat, ~area_code,
  0.25, 50.25, 1L,
  0.75, 50.25, 1L
)
```

```
build_gridded_landuse(
  country_areas, crop_patterns, gridded_cropland, country_grid,
  config = list(years = 2000L)
)
```

`build_gridded_livestock` *Build gridded livestock dataset*

Description

Disaggregate country-level FAOSTAT livestock stocks and emissions to a 0.5-degree grid. Each species group uses a tailored spatial proxy:

- **Ruminants** (cattle, buffalo, sheep/goats, equines): LUH2 managed pasture (`pastr`) plus rangeland (`range`), optionally weighted by a static manure-intensity reference (West et al. 2014).
- **Confined animals** (pigs, poultry): LUH2 aggregate cropland, reflecting intensive farming co-location with crop production.
- **Range specialists** (camels): LUH2 rangeland only.
- **Mixed** (other animals): 50/50 blend of pasture and cropland.

For each country, year, and species group the function distributes the national total proportionally to cell-level proxy weights:

$$\text{cell} = \frac{w_i}{\sum_{j \in \text{country}} w_j} \times T_{\text{country}}$$

where w_i is the proxy weight in cell i (land-use hectares times optional reference-pattern intensity) and T is the country total (heads or emissions).

Methodology:

Livestock spatialization is not covered by LandInG (Ostberg et al. 2023), which focuses on crops only. The approach here extends the LandInG framework by using the same LUH2-based spatial proxies (pasture, rangeland, cropland) for livestock distribution.

Country-level data comes from `build_primary_production()` (stocks) and the `faostat-emissions-livestock` pin (CH4/N2O emissions), with predecessor redistribution and pre-1961 backfill already applied.

The Zenodo livestock density input (Heinke 2025, doi:10.5281/zenodo.14946695) provides an alternative calibrated LSU/ha reference for use with the `glw_density` parameter.

Data sources and references:

Source	Use
FAOSTAT Production_Livestock (FAO 2024)	Country-level heads
FAOSTAT Emissions_livestock (FAO 2024)	Enteric CH4, manure CH4/N2O
LUH2 v2h (Hurtt et al. 2020)	Time-varying pasture + cropland
West et al. (2014)	Static manure-N intensity reference

GLW3 (Gilbert et al. 2018)	Species-specific density (optional)
Heinke (2025)	Calibrated LSU/ha density (optional)
IPCC 2006/2019	N-excretion rates, emission factors

Usage

```
build_gridded_livestock(
  livestock_data,
  gridded_pasture,
  gridded_cropland,
  country_grid,
  species_proxy = NULL,
  manure_pattern = NULL,
  glw_density = NULL,
  years = NULL
)
```

Arguments

`livestock_data` A tibble with country-level livestock data. Required columns:

- `year`: Integer year.
- `area_code`: Country code (WHEP polities).
- `species_group`: Livestock functional-type name (e.g. `"cattle"`, `"pigs"`, `"poultry"`).
- `heads`: Live animal count (number of head). Any additional numeric columns (e.g. `enteric_ch4_kt`, `manure_ch4_kt`, `manure_n2o_kt`, `manure_n_mg`) are distributed to the grid using the same proportional weights as `heads`.

`gridded_pasture`
A tibble with annual gridded pasture extent. Required columns:

- `lon`, `lat`: Cell centre coordinates (0.5 degree).
- `year`: Integer year.
- `pasture_ha`: Managed pasture area in hectares (LUH2 `pastr`).
- `rangeland_ha`: Rangeland area in hectares (LUH2 `range`).

`gridded_cropland`
A tibble with annual gridded cropland extent. Required columns:

- `lon`, `lat`: Cell centre coordinates.
- `year`: Integer year.
- `cropland_ha`: Total cropland area in hectares.

`country_grid` A tibble mapping grid cells to countries. Required columns:

- `lon`, `lat`: Cell centre coordinates.
- `area_code`: Country code. Optional columns:
- `cell_area_frac` (or `area_frac`): Fraction of the physical cell belonging to this polity compartment. Defaults to 1.

- `polycell_id`, `cell_id`: Stable compartment/cell identifiers preserved in outputs when present.
- `year` or validity intervals (`valid_from`/`valid_to`, `start_year`/`end_year`, `from_year`/`to_year`) for historical, time-varying polity overlays.

`species_proxy` A tibble mapping each `species_group` to its spatial proxy type: `"pasture"`, `"cropland"`, `"rangeland"`, or `"mixed"`. Required columns:

- `species_group`: Group name (must match `livestock_data`).
- `spatial_proxy`: One of `"pasture"`, `"cropland"`, `"rangeland"`, or `"mixed"`. If `NULL`, a default mapping is used (see Details).

`manure_pattern` A tibble with static manure-intensity weights (e.g. from West et al. 2014). Optional. Expected columns:

- `lon`, `lat`: Cell centre coordinates.
- `manure_intensity`: Relative intensity (kg N per ha or similar). Values are used multiplicatively with the land-use proxy. If `NULL`, land-use weights are used alone.

`glw_density` A tibble with species-specific gridded livestock density from GLW3 (Gilbert et al. 2018). Optional. Expected columns:

- `lon`, `lat`: Cell centre coordinates.
- `species_group`: Must match `livestock_data`.
- `density`: Heads per cell (reference year ~2010). If provided, this **replaces** the LUH2-based proxy for the matching groups, while still being scaled by LUH2 time trends. If `NULL`, LUH2 proxies are used for all groups.

`years` Integer vector of years to spatialize. If `NULL` (default), all years present in `livestock_data` are processed. When supplied, `livestock_data`, `gridded_pasture`, and `gridded_cropland` are filtered to this set before processing.

Value

A tibble with gridded livestock data. Columns:

- `lon`, `lat`: Cell centre coordinates.
- `area_code`: WHEP polity code for this cell compartment.
- `polycell_id`, `cell_id`: Preserved when supplied in `country_grid`.
- `year`: Integer year.
- `species_group`: Livestock functional type.
- `heads`: Allocated live animal count.
- Any additional numeric columns from `livestock_data` (e.g. `enteric_ch4_kt`, `manure_ch4_kt`).

Examples

```
# Minimal example with toy data
livestock_data <- tibble::tribble(
  ~year, ~area_code, ~species_group, ~heads,
  2000L,         1L,       "cattle",   5000
)
```

```
gridded_pasture <- tibble::tribble(
  ~lon,  ~lat,  ~year, ~pasture_ha, ~rangeland_ha,
   0.25, 50.25, 2000L,         600,           200,
   0.75, 50.25, 2000L,         400,           100
)
gridded_cropland <- tibble::tribble(
  ~lon,  ~lat,  ~year, ~cropland_ha,
   0.25, 50.25, 2000L,          800,
   0.75, 50.25, 2000L,          500
)
country_grid <- tibble::tribble(
  ~lon,  ~lat, ~area_code,
   0.25, 50.25,         1L,
   0.75, 50.25,         1L
)
build_gridded_livestock(
  livestock_data, gridded_pasture, gridded_cropland, country_grid
)
```

`build_io_model` *Build multi-regional input-output model.*

Description

Construct a multi-regional input-output (MRIO) model from supply-use tables, bilateral trade, and commodity balance sheets. Uses the industry technology assumption to derive symmetric product-by-product tables.

The resulting matrices follow the FABIO methodology (Bruckner et al., 2019). Rows and columns of `Z` represent (country, item) pairs. Each entry `Z[i,j]` gives the intermediate flow from sector `i` to sector `j`.

Usage

```
build_io_model(
  supply_use = NULL,
  bilateral_trade = NULL,
  cbs = NULL,
  years = NULL,
  endogenize_losses = FALSE
)
```

Arguments

`supply_use`	Tibble from `build_supply_use()`. By default, this function calls `build_supply_use()` internally. Must have columns: `year`, `area_code`, `proc_group`, `proc_cbs_code`, `item_cbs_code`, `type`, `value`.

bilateral_trade
: Tibble from get_bilateral_trade(). By default, this function calls get_bilateral_trade() internally. Must have columns: year, item_cbs_code, bilateral_trade (list-column of matrices).

cbs
: Tibble from get_wide_cbs(). By default, this function calls get_wide_cbs() internally. Must have columns: year, area_code, item_cbs_code, production, import, export, stock_withdrawal, stock_addition, plus final demand columns (food, other_uses).

years
: Numeric vector of years to compute, or NULL. If NULL, computes all years in the intersection of available data across inputs. If specified, must be a subset of available years.

endogenize_losses
: Logical. If TRUE and cbs contains a losses column, losses are moved from final demand to the diagonal of Z (self-use), following the FABIO convention. The losses column is removed from Y and fd_labels. Defaults to FALSE.

Value

A tibble with one row per year and list-columns:

- Z: Inter-industry flow matrix (product-by-product).
- Y: Final demand matrix.
- X: Total output vector.
- labels: Tibble mapping row/column indices to area_code and item_cbs_code.
- fd_labels: Tibble mapping each Y column to its area_code (consuming country) and fd_col (demand category, e.g. "food"). Pass to compute_footprint() as fd_labels to get a target_fd column in the footprint output.

Examples

```
su <- build_supply_use(example = TRUE)
btd <- get_bilateral_trade(example = TRUE)
cbs <- get_wide_cbs(example = TRUE)
build_io_model(su, btd, cbs)
```

---

build_primary_prices *Build primary item prices*

Description

Compute prices for primary production items. Export trade prices are preferred; when unavailable, production value prices (gross production value divided by quantity) are used as fallback. Gaps are filled via linear interpolation.

Usage

```
build_primary_prices(
  primary_prod,
  value_of_production = NULL,
  trade_prices = NULL,
  example = FALSE
)
```

Arguments

primary_prod	A tibble of primary production, as returned by build_primary_production() or get_primary_production().
value_of_production	A data frame with FAOSTAT Value of Production data. Must contain columns Item.Code (or item_code_prod), Element, Unit, Year, Value, Area.Code (or area_code). If NULL, only trade prices are used.
trade_prices	A tibble as returned by build_trade_prices(). If NULL, it is computed internally.
example	Logical. If TRUE, return a small example tibble. Default FALSE.

Value

A tibble with columns:

- year: Integer year.
- item_prod_code: Numeric production item code.
- price: Price in KDollars per tonne.

Examples

```
build_primary_prices(example = TRUE)
```

---

build_primary_production

Build primary production dataset

Description

Construct the full primary production dataset from raw FAOSTAT inputs. This is a convenience wrapper that chains the three pipeline steps:

1. .read_production() — read & reformat FAOSTAT data.
2. .fix_production() — apply Global-ported corrections.
3. .qc_production() — flag data-quality anomalies.

Usage

```
build_primary_production(
  start_year = 1850,
  end_year = 2023,
  smooth_carry_forward = FALSE,
  example = FALSE,
  show_duplicates = FALSE,
  .raw_data = NULL
)
```

Arguments

start_year	Integer. First year to include. Default 1850.
end_year	Integer. Last year to include. Default 2023.
smooth_carry_forward	Logical. If TRUE, carry-forward tails are replaced with a linear trend. Default FALSE.
example	Logical. If TRUE, return a small hardcoded example tibble instead of reading remote data. Default FALSE.
show_duplicates	Logical. If TRUE, return only the rows that have competing sources in wide format (one column per source) for diagnostic comparison. Default FALSE.
.raw_data	Optional tibble with the same structure as the output of the internal .read_production() step. When supplied, the remote-data read is skipped entirely and the pipeline starts from .fix_production(). Columns required: year, area, area_code, item_prod, item_prod_code, item_cbs, item_cbs_code, live_anim, live_anim_code, unit, value, source. Default NULL.

Value

A tibble with the same columns as get_primary_production(): year, area_code (numeric FAOSTAT), item_prod_code, item_cbs_code, live_anim_code, unit, value. Names can be recovered via add_area_name(), add_item_prod_name(), etc. When show_duplicates = TRUE, returns a wide tibble with one column per source showing the competing values.

Examples

```
build_primary_production(example = TRUE)
```

---

build_processing_coefs

Build processing coefficients

Description

Extract the final calibrated processing coefficients from the CBS building pipeline. These can be used independently for footprint calculations.

Usage

```
build_processing_coefs(
  cbs,
  start_year = 1850,
  end_year = 2023,
  example = FALSE
)
```

Arguments

cbs	A tibble of final CBS in wide format, as returned by build_commodity_balances().
start_year	Integer. First year to include. Default 1850.
end_year	Integer. Last year to include. Default 2023.
example	Logical. If TRUE, return a small hardcoded dataset for illustration without downloading data. Default FALSE.

Value

A tibble with columns: year, area_code, item_cbs_code_to_process, value_to_process, item_cbs_code_processed, initial_conversion_factor, initial_value_processed, conversion_factor_scaling, final_conversion_factor, final_value_processed.

Examples

```
build_processing_coefs(example = TRUE)
```

---

build_supply_use *Supply and use tables*

Description

Create a table with processes, their inputs (*use*) and their outputs (*supply*).

Usage

```
build_supply_use(example = FALSE)
```

Arguments

example	If TRUE, return a small example output without downloading remote data. Default is FALSE.

Value

A tibble with the supply and use data for processes. It contains the following columns:

- `year`: The year in which the recorded event occurred.
- `area_code`: The code of the country where the data is from. For code details see e.g. `add_area_name()`.
- `proc_group`: The type of process taking place. It can be one of:
  - `crop_production`: Production of crops and their residues, e.g. rice production, coconut production, etc.
  - `husbandry`: Animal husbandry, e.g. dairy cattle husbandry, non-dairy cattle husbandry, layers chickens farming, etc.
  - `animal_draught`: Annual useful work energy generated by draft animals.
  - `processing`: Derived subproducts obtained from processing other items. The items used as inputs are those that have a non-zero processing use in the commodity balance sheet. See `get_wide_cbs()` for more details. In each process there is a single input. In some processes like olive oil extraction or soyabean oil extraction this might make sense. Others like alcohol production need multiple inputs (e.g. multiple crops work), so in this data there would not be a process like alcohol production but rather a *virtual* process like 'Wheat and products processing', giving all its possible outputs. This is a constraint because of how the data was obtained and might be improved in the future. See `get_processing_coefs()` for more details.
- `proc_cbs_code`: The code of the main item in the process taking place. Together with `proc_group`, these two columns uniquely represent a process. The main item is predictable depending on the value of `proc_group`:
  - `crop_production`: The code is from the item for which seed usage (if any) is reported in the commodity balance sheet (see `get_wide_cbs()` for more). For example, the rice code for a rice production process or the cottonseed code for the cotton production one.
  - `husbandry`: The code of the farmed animal, e.g. bees for beekeeping, non-dairy cattle for non-dairy cattle husbandry, etc.
  - `processing`: The code of the item that is used as input, i.e., the one that is processed to get other derived products. This uniquely defines a process within the group because of the nature of the data that was used, which you can see in `get_processing_coefs()`.

  For code details see e.g. `add_item_cbs_name()`.
- `item_cbs_code`: The code of the item produced or used in the process. Note that this might be the same value as `proc_cbs_code`, e.g., in rice production process for the row defining the amount of rice produced or the amount of rice seed as input, but it might also have a different value, e.g. for the row defining the amount of straw residue from rice production. For code details see e.g. `add_item_cbs_name()`.
- `type`: Can have two values:
  - `use`: The given item is an input of the process.
  - `supply`: The given item is an output of the process.
- `value`: Quantity in the item's own unit. Most items are measured in tonnes; animal draught is measured as annual useful work energy (TJ).

Examples

```
build_supply_use(example = TRUE)
```

`build_trade_prices` *Build global trade prices*

Description

Compute global prices of traded items from FAOSTAT trade data. For each item and element (import/export), the price is `KDollars / tonnes` aggregated across all countries.

Usage

```
build_trade_prices(raw_trade = NULL, example = FALSE)
```

Arguments

`raw_trade`	A data.table or tibble of FAOSTAT bilateral trade data with columns `year`, `item_trade`, `item_code_trade`, `unit`, `element`, and `value`. Must include both quantity (`"tonnes"`) and value (`"1000 US$"`) rows. If `NULL` (default), the data is read from the `"faostat-trade-bilateral"` pin.
`example`	Logical. If `TRUE`, return a small example tibble. Default `FALSE`.

Value

A tibble with columns:

- `year`: Integer year.
- `item_trade`: Trade item name.
- `item_code_trade`: Numeric FAOSTAT trade item code.
- `element`: `"import"` or `"export"`.
- `kdollars`: Total trade value in thousand US dollars.
- `tonnes`: Total trade quantity in tonnes.
- `price`: Price in KDollars per tonne.

Examples

```
build_trade_prices(example = TRUE)
```

calculate_cohorts_systems *Calculate cohort and production system distribution.*

Description

Distributes national herd totals across GLEAM-defined cohorts and production systems using `gleam_livestock_categorie` and regional weight data.

Usage

```
calculate_cohorts_systems(data, system_shares = NULL)
```

Arguments

`data`	Dataframe with `species`, `heads`, and optionally `iso3` or `region`.
`system_shares`	Optional dataframe with `species`, `system`, `share` columns. If `NULL`, uses GLEAM defaults.

Value

Dataframe expanded to cohort level with `cohort`, `system`, `cohort_heads`, and `cohort_fraction` columns.

Examples

```
tibble::tibble(
  species = "Cattle", heads = 10000,
  iso3 = "DEU"
) |>
  calculate_cohorts_systems()
```

calculate_enteric_ch4 *Calculate enteric methane emissions.*

Description

Wrapper that selects Tier 1 or 2 for enteric CH4 based on data availability.

Usage

```
calculate_enteric_ch4(data, tier = NULL)
```

Arguments

data	Dataframe with species, heads. For Tier 2, also needs cohort, weight, and diet_quality. For Tier 1, iso3 is used to select regional emission factors.
tier	Integer 1 or 2. If NULL (default), auto-selects based on data completeness.

Value

Dataframe with all input columns preserved, plus:

- method_enteric: tracking label ("IPCC_2019_Tier1" or "IPCC_2019_Tier2").
- Tier 1: enteric_ef_kgch4 (emission factor), enteric_ch4_tier1 (total kg CH4).
- Tier 2: gross_energy, ym_factor, enteric_ch4_per_head (kg CH4/head/yr), enteric_ch4_tier2 (total kg CH4).

Examples

```
tibble::tibble(
  species = "Cattle", heads = 1000, iso3 = "DEU"
) |>
  calculate_enteric_ch4(tier = 1)
```

---

calculate_livestock_emissions

Calculate all livestock emissions.

Description

Main dispatcher that runs the full IPCC 2019 livestock emissions pipeline: energy demand (Tier 2), enteric CH4, manure CH4, and manure N2O.

Selects tier automatically: Tier 2 when cohort-level data (weight, diet) are available; Tier 1 otherwise.

Usage

```
calculate_livestock_emissions(data, tier = NULL)
```

Arguments

data	Dataframe with at minimum species and heads. For Tier 2, also needs cohort, weight (or iso3), diet_quality, and production columns.
tier	Integer 1 or 2. If NULL (default), auto-selects based on data completeness.

Value

Dataframe with all emission columns, method tracking, and original data columns preserved.

Examples

```
tibble::tibble(
  species = "Dairy Cattle",
  cohort = "Adult Female",
  heads = 1000,
  weight = 600,
  diet_quality = "High",
  milk_yield_kg_day = 20
) |>
  calculate_livestock_emissions() |>
  dplyr::select(species, cohort, heads,
    enteric_ch4_tier2, manure_ch4_tier2,
    manure_n2o_total)
```

`calculate_lmdi` *Calculate LMDI decomposition.*

Description

Performs LMDI (Log Mean Divisia Index) decomposition analysis with flexible identity parsing, automatic factor detection, and support for multiple periods and groupings. Supports sectoral decomposition using bracket notation for both summing and grouping operations.

Usage

```
calculate_lmdi(
  data,
  identity,
  identity_labels = NULL,
  time_var = year,
  periods = NULL,
  periods_2 = NULL,
  .by = NULL,
  rolling_mean = 1,
  output_format = "clean",
  verbose = TRUE
)
```

Arguments

`data`	A data frame containing the variables for decomposition. Must include all variables specified in the identity, time variable, and any grouping variables.
`identity`	Character. Decomposition identity in format `"target:factor1*factor2*..."`. The target appears before the colon, factors after, separated by asterisks. Supports explicit ratios with `/` and structural decomposition with `[]`.

`identity_labels`
: Character vector. Custom labels for factors to use in output instead of variable names. The first element labels the target, and subsequent elements label each factor in order. Default: NULL uses variable names as-is.

`time_var`
: Unquoted name of the time variable column in the data. Default: `year`. Must be numeric or coercible to numeric.

`periods`
: Numeric vector. Years defining analysis periods. Each consecutive pair defines one period. Default: NULL uses all available years.

`periods_2`
: Numeric vector. Additional period specification for complex multi-period analyses. Default: NULL.

`.by`
: Character vector. Grouping variables for performing separate decompositions. Default: NULL (single decomposition for all data).

`rolling_mean`
: Numeric. Window size for rolling mean smoothing applied before decomposition. Default: 1 (no smoothing).

`output_format`
: Character. Format of output data frame. Options: `"clean"` (default) or `"total"`.

`verbose`
: Logical. If TRUE (default), prints progress messages during decomposition.

Details

The LMDI method decomposes changes in a target variable into contributions from multiple factors using logarithmic mean weights. This implementation supports:

Flexible identity specification:

- Automatic factor detection from identity string.
- Support for ratio calculations (implicit division).
- Sectoral aggregation with `[]` notation.
- Sectoral grouping with `{}` notation.

Period analysis: The function can decompose changes over single or multiple periods. Periods are defined by consecutive pairs in the `periods` vector.

Grouping capabilities: Use `.by` to perform separate decompositions for different groups (e.g., countries, regions) while maintaining consistent factor structure.

Value

A tibble with LMDI decomposition results containing:

- Time variables and grouping variables (if specified).
- `additive`: Additive contributions (sum equals total change in target).
- `multiplicative`: Multiplicative indices (product equals target ratio).
- `multiplicative_log`: Log of multiplicative indices.
- Period identifiers and metadata.

Identity Syntax

The identity parameter uses a special syntax to define decomposition:

Basic format: `"target:factor1*factor2*factor3"`

Simple decomposition (no sectors):

- Basic: `"emissions:gdp*(emissions/gdp)"`
- Complete: `"emissions:(emissions/gdp)*(gdp/population)*population"`

Understanding bracket notation:

Square brackets `[]` specify variables to sum across categories, enabling structural decomposition.
The bracket aggregates values BEFORE calculating ratios.

Single-level structural decomposition:

- `"emissions:activity*(activity[sector]/activity)*(emissions[sector]/activity[sector])"`
- Creates 3 factors: Activity level, Sectoral structure, Sectoral intensity.

Multi-level structural decomposition:

- Two levels: `"emissions:activity*(activity[sector]/activity)*(activity[sector+fuel]/activity[secto`
- Creates 4 factors: Activity level, Sector structure, Fuel structure, Sectoral-fuel intensity.

Data Requirements

The input data frame must contain:

- All variables mentioned in the identity.
- The time variable (default: "year").
- Grouping variables if using `.by`.
- No missing values in key variables for decomposition periods.

Examples

```
# In these examples, 'activity' is a measure of scale
# (e.g., GDP in million USD) and 'intensity' is the target
# variable per unit activity (e.g., emissions per million USD).
# The units are illustrative; adapt to your context.
# --- Shared sample data ---
data_simple <- tibble::tribble(
  ~year, ~activity, ~intensity, ~emissions,
  2010,  1000,      0.10,       100,
  2011,  1100,      0.12,       132,
  2012,  1200,      0.09,       108,
  2013,  1300,      0.10,       130
)

# --- 1. Year-over-year decomposition (default) ---
# Decompose annual emission changes into activity and intensity effects.
# The additive column sums to the total change in emissions each period.
calculate_lmdi(
```

```
  data_simple,
  identity = "emissions:activity*intensity",
  time_var = year,
  verbose = FALSE
) |>
  dplyr::select(
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )

# --- 2. Single baseline-to-end period ---
# Pass a two-element periods vector to get a single cumulative period
# instead of year-over-year results.
calculate_lmdi(
  data_simple,
  identity = "emissions:activity*intensity",
  time_var = year,
  periods = c(2010, 2013),
  verbose = FALSE
) |>
  dplyr::select(
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )

# --- 3. Year-over-year AND one cumulative summary period ---
# Use periods_2 to append an extra comparison period alongside the
# year-over-year results.
calculate_lmdi(
  data_simple,
  identity = "emissions:activity*intensity",
  time_var = year,
  periods = c(2010, 2011, 2012, 2013),
  periods_2 = c(2010, 2013),
  verbose = FALSE
) |>
  dplyr::select(
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )

# --- 4. Per-country decomposition with .by ---
# Separate LMDI runs per country; results are stacked with a country column.
data_countries <- tibble::tribble(
```

```
  ~year, ~country, ~activity, ~intensity, ~emissions,
  2010, "ESP", 1000, 0.10, 100,
  2011, "ESP", 1100, 0.11, 121,
  2012, "ESP", 1200, 0.10, 120,
  2010, "FRA", 2000, 0.05, 100,
  2011, "FRA", 2200, 0.05, 110,
  2012, "FRA", 2400, 0.05, 120
)

calculate_lmdi(
  data_countries,
  identity = "emissions:activity*intensity",
  time_var = year,
  .by = "country",
  verbose = FALSE
) |>
  dplyr::select(
    country,
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )

# --- 5. Ratio notation ---
# Express factors as explicit ratios (e.g. intensity = emissions/activity).
# Factor labels in the output preserve the ratio form for clarity.
calculate_lmdi(
  data_simple,
  identity = "emissions:(emissions/activity)*activity",
  time_var = year,
  verbose = FALSE
) |>
  dplyr::select(
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )

# --- 6. Structural (sectoral) decomposition with [] notation ---
# Decomposes emissions into:
#   total_activity * sector_structure * sector_intensity
# [] sums the bracketed variable across sector before forming ratios,
# enabling proper structural decomposition.
data_sectors <- tibble::tribble(
  ~year, ~sector, ~activity, ~emissions,
  2010, "industry", 600, 60,
  2010, "transport", 400, 40,
  2011, "industry", 700, 63,
  2011, "transport", 500, 55
```

```
) |>
  dplyr::group_by(year) |>
  dplyr::mutate(total_activity = sum(activity)) |>
  dplyr::ungroup()

calculate_lmdi(
  data_sectors,
  identity = paste0(
    "emissions:",
    "total_activity*",
    "(activity[sector]/total_activity)*",
    "(emissions[sector]/activity[sector])"
  ),
  time_var = year,
  verbose = FALSE
) |>
  dplyr::select(
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )

# --- 7. Custom factor labels ---
# Replace raw variable names with readable labels for reporting.
# Supply one label per term (target first, then each factor in order).
calculate_lmdi(
  data_simple,
  identity = "emissions:activity*intensity",
  identity_labels = c(
    "Total Emissions",
    "Activity Effect",
    "Intensity Effect"
  ),
  time_var = year,
  verbose = FALSE
) |>
  dplyr::select(
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )

# --- 8. Rolling mean smoothing before decomposition ---
# A 3-year rolling mean reduces noise in volatile series before
# computing LMDI weights. Edge years use partial windows (fewer
# than k observations) so no periods are lost.
data_smooth <- tibble::tibble(
  year      = 2010:2020,
  activity  = seq(1000, 2000, length.out = 11),
```

```
  intensity = rep(0.1, 11),
  emissions = seq(1000, 2000, length.out = 11) * 0.1
)

calculate_lmdi(
  data_smooth,
  identity = "emissions:activity*intensity",
  time_var = year,
  rolling_mean = 3,
  verbose = FALSE
) |>
  dplyr::select(
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )
```

`calculate_manure_emissions` *Calculate manure emissions (CH4 + N2O).*

Description

Wrapper that selects Tier 1 or 2 for manure CH4 and computes N2O (Tier 2 only; skipped for Tier 1).

Usage

```
calculate_manure_emissions(data, tier = NULL)
```

Arguments

`data`	Dataframe with `species`, `heads`. For Tier 2, also needs `cohort`, `weight`, and `diet_quality`. For Tier 1, `iso3` is used to select regional emission factors.
`tier`	Integer 1 or 2. If `NULL` (default), auto-selects based on data completeness.

Value

Dataframe with all input columns preserved, plus:

- `method_manure_ch4`: tracking label.
- Tier 1: `manure_ef_kgch4`, `manure_ch4_tier1`.
- Tier 2: `volatile_solids`, `methane_potential`, `weighted_mcf`, `manure_ch4_per_head`, `manure_ch4_tier2`.
- N2O (Tier 2 only): `method_manure_n2o`, `n_excretion`, `manure_n2o_direct`, `manure_n2o_indirect`, `manure_n2o_total`.

Examples

```
tibble::tibble(
  species = "Cattle", heads = 1000, iso3 = "DEU"
) |>
  calculate_manure_emissions(tier = 1)
```

`calculate_nue_crops` *N soil inputs and Nitrogen Use Efficiency (NUE) for crop*

Description

N inputs (deposition, fixation, synthetic fertilizers, urban sources, manure) and N production in Spain from 1860 to the present for the GRAFS model at the provincial level. The crop NUE is defined as the percentage of produced nitrogen relative to the total nitrogen inputs to the soil. Total soil inputs are calculated as: inputs = deposition + fixation + synthetic + manure + urban

Usage

```
calculate_nue_crops(example = FALSE)
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A tibble containing nitrogen use efficiency (NUE) for crops. It includes the following columns:

- `year`: Year.
- `province_name`: The Spanish province.
- `item`: The item which was produced, defined in `names_biomass_cb`.
- `box`: One of the two systems of the GRAFS model: cropland or semi-natural agroecosystems.
- `nue`: Nitrogen Use Efficiency as a percentage (%).

Examples

```
calculate_nue_crops(example = TRUE)
```

`calculate_nue_livestock` *NUE for Livestock*

Description

Calculates Nitrogen Use Efficiency (NUE) for livestock categories (excluding pets).

The livestock NUE is defined as the percentage of nitrogen in livestock products relative to the nitrogen in feed intake: nue = prod_n / feed_n * 100

Additionally, a mass balance is calculated to check the recovery of N in products and excretion relative to feed intake: mass_balance = (prod_n + excretion_n) / feed_n

Usage

```
calculate_nue_livestock(example = FALSE)
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A tibble containing:

- `year`: Year
- `province_name`: Spanish province
- `livestock_cat`: Livestock category
- `item`: Produced item
- `prod_n`: Nitrogen in livestock products (Mg)
- `feed_n`: Nitrogen in feed intake (Mg)
- `excretion_n`: Nitrogen excreted (Mg)
- `nue`: Nitrogen Use Efficiency (%)
- `mass_balance`: Mass balance ratio (%)

Examples

```
calculate_nue_livestock(example = TRUE)
```

calculate_system_nue *System NUE*

Description

Calculates the NUE for Spain at the provincial level. The system NUE is defined as the percentage of total nitrogen production (`total_prod`) relative to the sum of all nitrogen inputs (`inputs`) into the soil system.

Usage

```
calculate_system_nue(n_soil_inputs = create_n_soil_inputs(), example = FALSE)
```

Arguments

`n_soil_inputs`	A tibble of nitrogen soil input (deposition, fixation, synthetic, manure, urban). If not provided and `example = FALSE`, it will be computed from `create_n_soil_inputs()`.
`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A tibble with the following columns:

- `year`: Year
- `province_name`: Spanish province
- `total_prod`: Total nitrogen production (Mg)
- `inputs`: Total nitrogen inputs (Mg)
- `nue_system`: System-level Nitrogen Use Efficiency (%)

Examples

```
calculate_system_nue(example = TRUE)
```

calculate_uncertainty_bounds *Calculate uncertainty bounds for livestock emissions.*

Description

Applies IPCC uncertainty ranges to emission estimates. Multipliers sourced from `uncertainty_ranges` table (no hardcoded values).

Usage

```
calculate_uncertainty_bounds(data)
```

Arguments

data	Dataframe with emission columns from calculate_livestock_emissions().

Value

Dataframe with added _lower and _upper columns for each emission estimate.

Examples

```
tibble::tibble(
  species = "Dairy Cattle",
  cohort = "Adult Female",
  heads = 1000,
  weight = 600,
  diet_quality = "High",
  milk_yield_kg_day = 20
) |>
  calculate_livestock_emissions() |>
  calculate_uncertainty_bounds() |>
  dplyr::select(species, cohort, heads,
    enteric_ch4_tier2, enteric_ch4_tier2_lower,
    enteric_ch4_tier2_upper, manure_ch4_tier2,
    manure_ch4_tier2_lower, manure_ch4_tier2_upper)
```

---

cb_processing *Commodity balance sheet processing fractions*

Description

Specifies the product fractions obtained when CBS items are processed, linking processed items to their output CBS categories.

Usage

```
cb_processing
```

Format

A tibble where each row corresponds to one processed-item / output-category combination. It contains the following columns:

- ProcessedItem: Name of the CBS item being processed (e.g., "Apples and products", "Barley and products").
- item_cbs: Name of the output CBS category produced by processing (e.g., "Alcohol, Non-Food").

- `Product_fraction`: Conversion factor from processed input quantity to output product quantity. This can exceed 1 when the output includes added mass, such as water in beverages.
- `Value_fraction`: Economic value fraction associated with the output product (numeric; largely `NA` in current data).
- `Required`: Marks required co-product links in selected processing chains.

Source

Derived from FAOSTAT commodity balance sheet processing assumptions.

Examples

```
head(cb_processing)
```

`cbs_trade_codes` *CBS to trade item code mapping*

Description

Maps detailed FAOSTAT trade item codes to their corresponding CBS item categories, enabling aggregation of bilateral trade data into the CBS framework.

Usage

```
cbs_trade_codes
```

Format

A tibble where each row corresponds to one trade item. It contains the following columns:

- `item_code_trade`: Numeric FAOSTAT trade item code (e.g., `15` for wheat).
- `item_trade`: Name of the trade item (e.g., `"Wheat"`, `"Flour, wheat"`, `"Bran, wheat"`).
- `item_cbs`: Name of the CBS category this trade item belongs to (e.g., `"Wheat and products"`).
- `item_check`: Cross-validation column repeating the mapped CBS name; used to flag mapping inconsistencies during data processing.

Source

Derived from FAOSTAT Detailed Trade Matrix and commodity balance sheet correspondence tables.

Examples

```
head(cbs_trade_codes)
```

cft_mapping *FAOSTAT crop to LPJmL crop functional type (CFT) mapping*

Description

Maps FAOSTAT primary-production item codes to WHEP's granular 33-class crop functional type taxonomy and the coarser LPJmL-compatible parent class. Used by `build_gridded_landuse()` and `run_spatialize()` to aggregate spatialized crop-level output into named crop functional types.

Usage

```
cft_mapping
```

Format

A tibble with one row per mapped FAOSTAT item. Columns:

- `item_prod_code`: Integer FAOSTAT item code.
- `item_prod_name`: Human-readable FAOSTAT item name.
- `cft_name`: Granular WHEP CFT name (33 classes, e.g. `"temperate_cereals"`, `"coffee"`, `"oil_crops_oilpalm"`).
- `cft_lpjml`: LPJmL-compatible parent class; one of the 12 LPJmL v6 named crop CFTs or `"others"`.
- `luh2_type`: LUH2 crop functional type (`c3ann`, `c4ann`, `c3per`, or `c3nfx`).

Source

Adapted from LandInG's `crop_types_FAOSTAT_LPJmL_default.csv` (Ostberg et al. 2023) with WHEP granular extensions.

Examples

```
head(cft_mapping)
```

climate_mcf *Climate-zone MCF values.*

Description

Methane Conversion Factors by MMS type and climate zone (Cool/Temperate/Warm).

Usage

```
climate_mcf
```

Format

A tibble with `mms_type`, `climate_zone`, `mcf_percent`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.17.

Examples

```
climate_mcf
```

`compute_footprint` *Compute environmental footprints.*

Description

Trace environmental extensions through the supply chain using the Leontief inverse, following the FABIO methodology (Bruckner et al., 2019). The footprint shows how much of an environmental pressure (e.g. land use, water, emissions) is embodied in the final consumption of each product in each country.

The multiplier matrix is computed as $MP_{ij} = (e_i / X_i) \cdot L_{ij}$, where e_i is the extension for sector i. For each demand category, the footprint is decomposed per target item using the FABIO diagonal approach: $FP = MP \cdot \text{diag}(y)$, aggregated by item.

For large systems, pass `z_mat` and `x_vec` instead of `l_inv`. This solves $(I - A)x = Y$ directly using a sparse LU factorisation, avoiding the dense Leontief inverse entirely and reducing memory from $O(n^2)$ to $O(nnz)$.

Usage

```
compute_footprint(
  l_inv = NULL,
  x_vec,
  y_mat,
  extensions,
  labels,
  z_mat = NULL,
  fd_labels = NULL,
  output_tol = 1e-08,
  value_added_floor = 0.001,
  conserve_extensions = TRUE
)
```

Arguments

l_inv	Leontief inverse matrix from compute_leontief_inverse(). Ignored when z_mat is provided.
x_vec	Numeric vector of total output per sector.
y_mat	Final demand matrix from build_io_model().
extensions	Numeric vector of environmental extensions (e.g. hectares of land use) per sector. Must have the same length as x_vec.
labels	Tibble with area_code and item_cbs_code mapping row/column indices to their meaning. From build_io_model().
z_mat	Optional inter-industry flow matrix from build_io_model(). When provided, the system is solved directly (sparse LU), and l_inv is not needed.
fd_labels	Optional tibble labelling Y columns. Pass fd_labels[[i]] from build_io_model() output. When provided, footprints are decomposed per target item using the FABIO diagonal approach, and the result includes a target_fd column. When omitted, columns of Y are treated as sectors (appropriate only when Y is square).
output_tol	Minimum output considered valid when computing extension intensities. Sectors with x_vec <= output_tol get zero intensity to avoid infinite or numerically explosive footprints from zero-output residuals.
value_added_floor	Minimum share of each sector's output that is treated as non-intermediate leakage when constructing A from z_mat. Column sums larger than 1 - value_added_floor are rescaled to that maximum. Ignored when a precomputed l_inv is supplied without z_mat.
conserve_extensions	If TRUE, rescale positive footprint flows within each origin area/item so their sum does not exceed the corresponding positive extension total. This keeps footprint outputs conservative when capped coefficients or negative final demand columns would otherwise make positive-only paths larger than the source extension.

Value

A tibble with footprint results containing:

- origin_area: Country where the pressure occurs.
- origin_item: Item causing the pressure.
- target_area: Country consuming the product.
- target_item: Item consumed.
- target_fd: Demand category (e.g. "food"). Only present when fd_labels is provided.
- value: Footprint value in extension units.

Examples

```
z_mat <- matrix(c(0, 5, 10, 0), nrow = 2)
x_vec <- c(100, 200)
l_inv <- compute_leontief_inverse(z_mat, x_vec)
```

```
y_mat <- matrix(c(85, 195), ncol = 1)
extensions <- c(50, 30)
labels <- tibble::tibble(
  area_code = c(1L, 1L),
  item_cbs_code = c(1L, 2L)
)

# Small system: pass pre-computed L
compute_footprint(l_inv, x_vec, y_mat, extensions, labels)

# Using Z directly (computes L internally)
compute_footprint(
  x_vec = x_vec, y_mat = y_mat,
  extensions = extensions, labels = labels,
  z_mat = z_mat
)
```

`compute_footprint_paths` *Compute first-use footprint paths.*

Description

Decompose an origin footprint into the first sector that directly uses the origin product before the footprint reaches final demand. This is useful for Sankey views that show paths such as origin product -> first-use area -> first-use product -> final-demand area.

The decomposition uses the IO identity $x = d + Ax$. For each selected origin sector i and final-demand target, the origin requirement x_i is split into direct final demand d_i and direct intermediate use $A_{ij}x_j$. Values are multiplied by the origin extension intensity e_i/X_i.

Usage

```
compute_footprint_paths(
  z_mat,
  x_vec,
  y_mat,
  extensions,
  labels,
  fd_labels,
  origin_area = NULL,
  origin_item = NULL,
  output_tol = 1e-08,
  value_added_floor = 0.001,
  conserve_extensions = TRUE,
  min_value = 0
)
```

Arguments

z_mat	Inter-industry flow matrix from build_io_model().
x_vec	Numeric vector of total output per sector.
y_mat	Final demand matrix from build_io_model().
extensions	Numeric vector of environmental extensions per sector.
labels	Tibble with area_code and item_cbs_code mapping sectors.
fd_labels	Tibble labelling Y columns, from build_io_model().
origin_area	Optional area code vector limiting origin sectors.
origin_item	Optional item code vector limiting origin sectors.
output_tol	Minimum output considered valid when computing extension intensities.
value_added_floor	Minimum non-intermediate leakage share used when constructing technical coefficients from z_mat.
conserve_extensions	If TRUE, rescale positive paths within each origin area/item so their sum does not exceed the corresponding positive extension total.
min_value	Drop paths with values less than or equal to this value before returning.

Value

A tibble with origin_area, origin_item, use_area, use_item, target_area, target_item, target_fd, path_type, and value.

---

compute_fp_product_paths

Compute final-product footprint paths.

Description

Decompose an origin footprint by the area and item of the product supplied to final demand. This adds the missing FABIO-viewer style phase between origin product and final-demand area: origin product -> supplied product area -> supplied product -> final-demand area.

Unlike compute_footprint_paths(), this does not show the first direct intermediate input. It shows the downstream product row in Y whose final demand carries the origin footprint.

Usage

```
compute_fp_product_paths(
  z_mat,
  x_vec,
  y_mat,
  extensions,
  labels,
```

```
  fd_labels,
  origin_area = NULL,
  origin_item = NULL,
  output_tol = 1e-08,
  value_added_floor = 0.001,
  conserve_extensions = TRUE,
  min_value = 0
)
```

Arguments

z_mat	Inter-industry flow matrix from build_io_model().
x_vec	Numeric vector of total output per sector.
y_mat	Final demand matrix from build_io_model().
extensions	Numeric vector of environmental extensions per sector.
labels	Tibble with area_code and item_cbs_code mapping sectors.
fd_labels	Tibble labelling Y columns, from build_io_model().
origin_area	Optional area code vector limiting origin sectors.
origin_item	Optional item code vector limiting origin sectors.
output_tol	Minimum output considered valid when computing extension intensities.
value_added_floor	Minimum non-intermediate leakage share used when constructing technical coefficients from z_mat.
conserve_extensions	If TRUE, rescale positive paths within each origin area/item so their sum does not exceed the corresponding positive extension total.
min_value	Drop paths with values less than or equal to this value before returning.

Value

A tibble with origin_area, origin_item, product_area, product_item, target_area, target_fd, and value.

---

compute_leontief_inverse

Compute Leontief inverse.

Description

Compute the Leontief inverse matrix from intermediate flows and total output. The Leontief inverse captures both direct and indirect requirements across the entire supply chain, enabling footprint tracing.

The technical coefficients matrix is computed as $A_{ij} = Z_{ij}/X_j$, representing the input of sector i needed per unit of output from sector j. Column sums of A are capped below 1 using

value_added_floor (FABIO convention plus an explicit leakage floor) to ensure $(I - A)$ is invertible even when supply-use data are inconsistent. The Leontief inverse is then $L = (I - A)^{-1}$.

For large systems (thousands of sectors) this function is not usable: the dense L matrix requires $n^2 \times 8$ bytes of memory (e.g. ~4.8 GiB for n = 25 000). Use compute_footprint() directly with z_mat and x_vec instead, which solves $(I - A)x = Y$ without ever materialising L.

Accepts both dense and sparse (Matrix package) inputs.

Usage

```
compute_leontief_inverse(z_mat, x_vec, max_n = 5000, value_added_floor = 0.001)
```

Arguments

z_mat	Square numeric matrix of inter-industry flows. Entry Z_{ij} is the flow from sector i to sector j. Can be dense or sparse.
x_vec	Numeric vector of total output per sector. Must have the same length as nrow(z_mat).
max_n	Maximum system size before aborting. Defaults to 5000. Set higher at your own risk of memory exhaustion.
value_added_floor	Minimum share of each sector's output that is treated as non-intermediate leakage when constructing A. Column sums larger than 1 - value_added_floor are rescaled to that maximum. Use 0 to recover the previous cap-at-one behavior, though this can leave singular systems.

Value

The Leontief inverse matrix L. Negative values are set to zero. Returns a dense matrix.

Examples

```
z_mat <- matrix(c(0, 5, 10, 0), nrow = 2)
x_vec <- c(100, 200)
compute_leontief_inverse(z_mat, x_vec)
```

---

create_n_nat_destiny *GRAFS Nitrogen (N) flows at Spain national level*

Description

Provides N flows of the Spanish agro-food system on a national level between 1860 and 2020. This dataset is the national equivalent of the provincial GRAFS model and represents Spain as a single system without internal trade between provinces. All production, consumption and soil inputs are aggregated nationally before calculating trade with the outside.

Usage

```
create_n_nat_destiny(example = FALSE)
```

Arguments

example	If TRUE, return a small example output without downloading remote data. Default is FALSE.

Value

A final tibble containing national N flow data by origin and destiny. It includes the following columns:

- year: The year in which the recorded event occurred.
- item: The item which was produced, defined in names_biomass_cb.
- irrig_cat: Irrigation form (irrigated or rainfed)
- box: One of the GRAFS model systems: cropland, Semi-natural agroecosystems, Livestock, Fish, or Agro-industry.
- origin: The origin category of N: Cropland, Semi-natural agroecosystems, Livestock, Fish, Agro-industry, Deposition, Fixation, Synthetic, People (waste water), Livestock (manure).
- destiny: The destiny category of N: population_food, population_other_uses, livestock_mono, livestock_rum (feed), export, Cropland (for N soil inputs).
- mg_n: Nitrogen amount in megagrams (Mg).
- province_name: Set to "Spain" for all national-level rows.

Examples

```
create_n_nat_destiny(example = TRUE)
```

---

create_n_production *N production for Spain*

Description

Calculates N production at the provincial level in Spain. Production is derived from consumption, export, import, and other uses.

Usage

```
create_n_production(example = FALSE)
```

Arguments

example	If TRUE, return a small example output without downloading remote data. Default is FALSE.

Value

A tibble containing:

- year: Year
- province_name: Spanish province
- item: Product item
- box: Ecosystem box
- prod: Produced N (Mg)

Examples

```
create_n_production(example = TRUE)
```

---

create_n_prov_destiny *GRAFS Nitrogen (N) flows*

Description

Provides N flows of the spanish agro-food system on a provincial level between 1860 and 2020. This dataset is the the base of the GRAFS model and contains data in megagrams of N (MgN) for each year, province, item, origin and destiny. Thereby, the origin column represents where N comes from, which includes N soil inputs, imports and production. The destiny column shows where N goes to, which includes export, population food, population other uses and feed or cropland (in case of N soil inputs). Processed items, residues, woody crops, grazed weeds are taken into account.

Usage

```
create_n_prov_destiny(example = FALSE)
```

Arguments

example	If TRUE, return a small example output without downloading remote data. Default is FALSE.

Value

A final tibble containing N flow data by origin and destiny. It includes the following columns:

- year: The year in which the recorded event occurred.
- province_name: The Spanish province where the data is from.
- item: The item which was produced, defined in names_biomass_cb.
- irrig_cat: Irrigation form (irrigated or rainfed)
- box: One of the GRAFS model systems: cropland, Semi-natural agroecosystems, Livestock, Fish, or Agro-industry.

- `origin`: The origin category of N: Cropland, Semi-natural agroecosystems, Livestock, Fish, Agro-industry, Deposition, Fixation, Synthetic, People (waste water), Livestock (manure).
- `destiny`: The destiny category of N: population_food, population_other_uses, livestock_mono, livestock_rum (feed), export, Cropland (for N soil inputs).
- `mg_n`: Nitrogen amount in megagrams (Mg).

Examples

```
create_n_prov_destiny(example = TRUE)
```

`create_n_soil_inputs` *Nitrogen (N) soil inputs for Spain*

Description

Calculates total nitrogen inputs to soils in Spain at the provincial level. This includes contributions from:

- Atmospheric deposition (`deposition`)
- Biological nitrogen fixation (`fixation`)
- Synthetic fertilizers (`synthetic`)
- Manure (excreta, solid, liquid) (`manure`)
- Urban sources (`urban`)

Special land use categories and items are aggregated:

- Semi-natural agroecosystems (e.g., Dehesa, Pasture_Shrubland)
- Firewood biomass (e.g., Conifers, Holm oak)

Usage

```
create_n_soil_inputs(example = FALSE)
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A tibble containing:

- `year`: Year
- `province_name`: Spanish province
- `item`: Crop, land use, or biomass item
- `irrig_cat`: Irrigation form (irrigated or rainfed)

- box: Land use or ecosystem box for aggregation
- deposition: N input from atmospheric deposition (Mg)
- fixation: N input from biological N fixation (Mg)
- synthetic: N input from synthetic fertilizers (Mg)
- manure: N input from livestock manure (Mg)
- urban: N input from urban sources (Mg)

Examples

```
create_n_soil_inputs(example = TRUE)
```

---

crops_eurostat *Eurostat crop classification codes*

Description

Maps Eurostat crop codes to their full crop category names, used when integrating Eurostat agricultural statistics.

Usage

```
crops_eurostat
```

Format

A tibble where each row corresponds to one Eurostat crop category. It contains the following columns:

- Crop: Eurostat crop code (e.g., "G0000", "G1000").
- Name_Eurostat: Full name of the crop category as used in Eurostat (e.g., "Plants harvested green from arable land", "Temporary grasses and grazings").

Source

Eurostat Agricultural Statistics.

Examples

```
head(crops_eurostat)
```

crops_manure_n *Manure nitrogen application by crop and country*

Description

Country- and crop-level estimates of manure nitrogen applied to cropland, from West et al. (2014). Used as a reference for spatializing manure N inputs in the WHEP pipeline.

Usage

```
crops_manure_n
```

Format

A tibble with one row per crop-country combination containing:

- `Crop_name`: Crop name (character).
- `ISO`: ISO 3166-1 alpha-3 country code.
- `Continent`: Three-letter continent code (e.g. `"AFR"`, `"ASI"`).
- `Manure_N_Mg`: Manure nitrogen applied in megagrams (Mg).

Source

West, P. C. et al. (2014). Leverage points for improving global food security and the environment. *Science*, 345(6194), 325–328. doi:10.1126/science.1246067

Examples

```
head(crops_manure_n)
```

estimate_energy_demand *Estimate energy demand (Gross Energy) - Tier 2*

Description

Calculate gross energy (GE) intake per IPCC 2019 Tier 2 equations (Vol 4, Ch 10). Estimates net energy components for maintenance, activity, lactation, work, pregnancy, growth, and wool, then derives total gross energy using the REM/REG ratio approach from IPCC Eq 10.16.

All coefficients come from internal package data.

Usage

```
estimate_energy_demand(data, method = "ipcc2019")
```

Arguments

data	A dataframe with columns species, cohort, heads, and optionally iso3. Optional production columns: weight, milk_yield_kg_day, fat_percent, weight_gain_kg_day, work_hours_day, pregnant_fraction, temperature_c, diet_quality, grazing_distance_km, system.
method	Method for calculation (default "ipcc2019").

Value

Dataframe with added gross_energy (MJ/day), intermediate net energy components, and method_energy tracking column.

Examples

```
tibble::tibble(
  species = "Dairy Cattle", cohort = "Adult Female",
  heads = 100, weight = 600, diet_quality = "High",
  milk_yield_kg_day = 20
) |>
  estimate_energy_demand() |>
  dplyr::select(species, cohort, heads, ne_maintenance,
    ne_activity, ne_lactation, ne_growth, gross_energy)
```

---

expand_trade_sources *Trade data sources*

Description

Create a new dataframe where each row has a year range into one where each row is a single year, effectively 'expanding' the whole year range.

Usage

```
expand_trade_sources(trade_sources)
```

Arguments

trade_sources	A tibble dataframe where each row contains the year range.

Value

A tibble dataframe where each row corresponds to a single year for a given source.

Examples

```
trade_sources <- tibble::tibble(
  Name = c("a", "b", "c"),
  Trade = c("t1", "t2", "t3"),
  Info_Format = c("year", "partial_series", "year"),
  Timeline_Start = c(1, 1, 2),
  Timeline_End = c(3, 4, 5),
  Timeline_Freq = c(1, 1, 2),
  `Imp/Exp` = "Imp",
  SACO_link = NA,
)
expand_trade_sources(trade_sources)
```

---

feed_characteristics *Feed characteristics by diet quality.*

Description

DE%, NDF%, GE content, and crude protein percentage for High/Medium/Low diet quality levels.

Usage

```
feed_characteristics
```

Format

A tibble with `diet_quality`, `de_percent`, `ndf_percent`, `ge_content_mj_kg_dm`, `cp_percent`.

Source

IPCC 2019, Vol 4, Ch 10.

Examples

```
feed_characteristics
```

---

fill_linear *Fill gaps by linear interpolation, or carrying forward or backward.*

Description

Fills gaps (`NA` values) in a time-dependent variable by linear interpolation between two points, or carrying forward or backwards the last or initial values, respectively. It also creates a new variable indicating the source of the filled values.

Usage

```
fill_linear(
  data,
  value_col,
  time_col = year,
  interpolate = TRUE,
  fill_forward = TRUE,
  fill_backward = TRUE,
  value_smooth_window = NULL,
  .by = NULL,
  .copy = TRUE
)
```

Arguments

data	A data frame containing one observation per row.
value_col	The column containing gaps to be filled.
time_col	The column containing time values. Default: year.
interpolate	Logical. If TRUE (default), performs linear interpolation.
fill_forward	Logical. If TRUE (default), carries last value forward.
fill_backward	Logical. If TRUE (default), carries first value backward.
value_smooth_window	An integer specifying the window size for a centered moving average applied to the variable before gap-filling. Useful for variables with high inter-annual variability. If NULL (default), no smoothing is applied.
.by	A character vector with the grouping variables (optional).
.copy	Logical. If TRUE (default), data.table inputs are defensively copied before mutation. Set to FALSE when the caller owns the data and does not need the original preserved.

Value

A tibble data frame (ungrouped) where gaps in value_col have been filled, and a new "source" variable has been created indicating if the value is original or, in case it has been estimated, the gapfilling method that has been used.

Examples

```
sample_tibble <- tibble::tibble(
  category = c("a", "a", "a", "a", "a", "a", "b", "b", "b", "b", "b", "b"),
  year = c(
    "2015", "2016", "2017", "2018", "2019", "2020",
    "2015", "2016", "2017", "2018", "2019", "2020"
  ),
  value = c(NA, 3, NA, NA, 0, NA, 1, NA, NA, NA, 5, NA),
)
fill_linear(sample_tibble, value, .by = c("category"))
```

```
fill_linear(
  sample_tibble,
  value,
  interpolate = FALSE,
  .by = c("category"),
)
```

`fill_proxy_growth` *Fill gaps using growth rates from proxy variables*

Description

Fills missing values using growth rates from a proxy variable (reference series). Supports regional aggregations, weighting, and linear interpolation for small gaps.

Usage

```
fill_proxy_growth(
  data,
  value_col,
  proxy_col,
  time_col = year,
  .by = NULL,
  max_gap = Inf,
  max_gap_linear = 3,
  fill_scope = NULL,
  value_smooth_window = NULL,
  proxy_smooth_window = 1,
  output_format = "clean",
  verbose = TRUE
)
```

Arguments

`data`	A data frame containing time series data.
`value_col`	The column containing values to fill.
`proxy_col`	Character or vector. Proxy variable(s) for calculating growth rates. Supports multiple syntax formats: • **Simple numeric proxy** (e.g., `"population"`): Auto-detects numeric columns and uses them as proxy variable. Inherits the `.by` parameter to compute proxy values per group. • **Simple categorical proxy** (e.g., `"region"`): Auto-detects categorical columns and interprets as `value_col:region`. Aggregates `value_col` by the specified groups. • **Advanced syntax** (e.g., `"gdp:region"`): Format is `"variable:group1+group2"`. Aggregates variable by specified groups.

- **Hierarchical fallback** (e.g., `c("population", "gdp:region")`): Tries first proxy, falls back to second if first fails.
- **Weighted aggregation** (e.g., `"gdp[population]"`): Weight variable by specified column during aggregation.

`time_col` The column containing time values. Default: `year`.

`.by` A character vector with the grouping variables (optional).

`max_gap` Numeric. Maximum gap size to fill using growth method. Default: Inf.

`max_gap_linear` Numeric. Maximum gap size for linear interpolation fallback. Default: 3.

`fill_scope` Quosure. Filter expression to limit filling scope. Default: NULL.

`value_smooth_window`
Integer. Window size for a centered moving average applied to the value column before gap-filling. Useful for variables with high inter-annual variability. If `NULL` (default), no smoothing is applied.

`proxy_smooth_window`
Integer. Window size for moving average smoothing of proxy reference values before computing growth rates. Default: 1.

`output_format` Character. Output format: "clean" or "detailed". Default: "clean".

`verbose` Logical. Print progress messages. Default: TRUE.

Details

Combined Growth Sequence (Hierarchical Interpolation):

When using multiple proxies with hierarchical fallback, the function implements an intelligent combined growth sequence strategy:

1. Better proxies (earlier in hierarchy) are tried first for each gap.
2. If a better proxy has partial coverage within a gap, those growth rates are used for the covered positions.
3. Fallback proxies fill only the remaining positions where better proxies are not available.
4. Values filled by better proxies are protected from being overwritten.

Value

A data frame with filled values. If output_format = "clean", returns original columns with updated value_col and added source column. If "detailed", includes all intermediate columns.

See Also

`fill_linear()`, `fill_sum()`

Examples

```
# Fill GDP using population as proxy
data <- tibble::tibble(
  country = rep("ESP", 4),
  year = 2010:2013,
```

```
  gdp = c(1000, NA, NA, 1200),
  population = c(46, 46.5, 47, 47.5)
)

fill_proxy_growth(
  data,
  value_col = gdp,
  proxy_col = "population",
  .by = "country"
)
```

`fill_sum` *Fill gaps summing the previous value of a variable to the value of another variable.*

Description

Fills gaps in a variable with the sum of its previous value and the value of another variable. When a gap has multiple observations, the values are accumulated along the series. When there is a gap at the start of the series, it can either remain unfilled or assume an invisible 0 value before the first observation and start filling with cumulative sum.

Usage

```
fill_sum(
  data,
  value_col,
  change_col,
  time_col = year,
  start_with_zero = TRUE,
  .by = NULL
)
```

Arguments

`data`	A data frame containing one observation per row.
`value_col`	The column containing gaps to be filled.
`change_col`	The column whose values will be used to fill the gaps.
`time_col`	The column containing time values. Default: `year`.
`start_with_zero`	Logical. If TRUE (default), assumes an invisible 0 value before the first observation and fills with cumulative sum starting from the first change_col value. If FALSE, starting NA values remain unfilled.
`.by`	A character vector with the grouping variables (optional).

Value

A tibble dataframe (ungrouped) where gaps in value_col have been filled, and a new "source" variable has been created indicating if the value is original or, in case it has been estimated, the gapfilling method that has been used.

Examples

```
sample_tibble <- tibble::tibble(
  category = c("a", "a", "a", "a", "a", "a", "b", "b", "b", "b", "b", "b"),
  year = c(
    "2015", "2016", "2017", "2018", "2019", "2020",
    "2015", "2016", "2017", "2018", "2019", "2020"
  ),
  value = c(NA, 3, NA, NA, 0, NA, 1, NA, NA, NA, 5, NA),
  change_variable = c(1, 2, 3, 4, 1, 1, 0, 0, 0, 0, 0, 1)
)
fill_sum(
  sample_tibble,
  value,
  change_variable,
  start_with_zero = FALSE,
  .by = c("category")
)
fill_sum(
  sample_tibble,
  value,
  change_variable,
  start_with_zero = TRUE,
  .by = c("category")
)
```

---

get_bilateral_trade *Bilateral trade data*

Description

Reports trade between pairs of countries in given years.

Usage

```
get_bilateral_trade(example = FALSE, cbs = NULL)
```

Arguments

example	If TRUE, return a small example output without downloading remote data. Default is FALSE.
cbs	Optional pre-computed wide CBS tibble from get_wide_cbs(). If NULL (default), it is built internally.

Value

A tibble with the reported trade between countries. For efficient memory usage, the tibble is not exactly in tidy format. It contains the following columns:

- `year`: The year in which the recorded event occurred.
- `item_cbs_code`: FAOSTAT internal code for the item that is being traded. For code details see e.g. `add_item_cbs_name()`.
- `bilateral_trade`: Square matrix of `NxN` dimensions where `N` is the total number of countries being considered. The matrix row and column names are exactly equal and they represent country codes.
  - Row name: The code of the country where the data is from. For code details see e.g. `add_area_name()`.
  - Column name: FAOSTAT internal code for the country that is importing the item. See row name explanation above.

  If `m` is the matrix, the value at `m["A", "B"]` is the trade in tonnes from country `"A"` to country `"B"`, for the corresponding year and item. The matrix can be considered *balanced*. This means:
  - The sum of all values from row `"A"`, where `"A"` is any country, should match the total exports from country `"A"` reported in the commodity balance sheet (which is considered more accurate for totals).
  - The sum of all values from column `"A"`, where `"A"` is any country, should match the total imports into country `"A"` reported in the commodity balance sheet (which is considered more accurate for totals).

  The sums may not be exactly the expected values because of precision issues and/or the iterative proportional fitting algorithm not converging fast enough, but should be relatively very close to the desired totals.

The step by step approach to obtain this data tries to follow the FABIO model and is explained below. All the steps are performed separately for each group of year and item.

- From the FAOSTAT reported bilateral trade, there are sometimes two values for one trade flow: the exported amount claimed by the reporter country and the import amount claimed by the partner country. Here, the export data was preferred, i.e., if country `"A"` says it exported `X` tonnes to country `"B"` but country `"B"` claims they got `Y` tonnes from country `"A"`, we trust the export data `X`. This choice is only needed if there exists a reported amount from both sides. Otherwise, the single existing report is chosen.
- Complete the country data, that is, add any missing combinations of country trade with NAs, which will be estimated later. In the matrix form, this doesn't increase the memory usage since we had to build a matrix anyway (for the balancing algorithm), and the *empty* parts also take up memory. This is also done for total imports/exports from the commodity balance sheet, but these are directly filled with 0s instead.
- The total imports and exports from the commodity balance sheet are balanced by downscaling the largest of the two to match the lowest. This is done in the following way:
  - If `total_imports > total_exports`: Set `import` as `total_exports * import / total_import`.
  - If `total_exports > total_exports`: Set `export` as `total_exports * export / total_export`.
- The missing data in the matrix must be estimated. It's done like this:

  - For each pair of exporter `i` and importer `j`, we estimate a bilateral trade `m[i, j]` using the export shares of `i` and import shares of `j` from the commodity balance sheet:
    * `est_1 <- exports[i] * imports[j] / sum(imports)`, i.e., total exports of country `i` spread among other countries' import shares.
    * `est_2 <- imports[j] * exports[i] / sum(exports)`, i.e. total imports of country `j` spread among other countries' export shares.
    * `est <- (est_1 + est_2) / 2`, i.e., the mean of both estimates.

    In the above computations, exports and imports are the original values before they were balanced.
  - The estimates for data that already existed (i.e. non-NA) are discarded. For the ones left, for each row (i.e. exporter country), we get the difference between its balanced total export and the sum of original non-estimated data. The result is the `gap` we can actually fill with estimates, so as to not get past the reported total export. If the sum of non-discarded estimates is larger, it must be downscaled and spread by computing `gap * non_discarded_estimate / sum(non_discarded_estimates)`.
  - The estimates are divided by a *trust factor*, in the sense that we don't rely on the whole value, thinking that a non-present value might actually be because that specific trade was 0, so we don't overestimate too much. The chosen factor is 10%, so only 10% of the estimate's value is actually used to fill the NA from the original bilateral trade matrix.

- The matrix is balanced, as mentioned before, using the iterative proportional fitting algorithm. The target sums for rows and columns are respectively the balanced exports and imports computed from the commodity balance sheet.

Examples

```
get_bilateral_trade(example = TRUE)
```

`get_faostat_data` *Scrapes activity_data from FAOSTAT and slightly post-processes it*

Description

Important: Dynamically allows for the introduction of subsets as `"..."`.

Note: overhead by individually scraping FAOSTAT code QCL for crop data; it's fine.

Usage

```
get_faostat_data(activity_data, ...)
```

Arguments

`activity_data`	activity data required from FAOSTAT; needs to be one of `c('livestock','crop_area','crop_yield',`
`...`	can be whichever column name from `get_faostat_bulk`, particularly `year`, `area` or `ISO3_CODE`.

Value

data.frame of FAOSTAT for activity_data; default is for all years and countries.

Examples

```
## Not run:
get_faostat_data("livestock", year = 2010, area = "Portugal")

## End(Not run)
```

---

get_feed_intake *Livestock feed intake*

Description

Get amount of items used for feeding livestock.

Usage

```
get_feed_intake(example = FALSE)
```

Arguments

example	If TRUE, return a small example output without downloading remote data. Default is FALSE.

Value

A tibble with the feed intake data. It contains the following columns:

- year: The year in which the recorded event occurred.
- area_code: The code of the country where the data is from. For code details see e.g. add_area_name().
- live_anim_code: Commodity balance sheet code for the type of livestock that is fed. For code details see e.g. add_item_cbs_name().
- item_cbs_code: The code of the item that is used for feeding the animal. For code details see e.g. add_item_cbs_name().
- feed_type: The type of item that is being fed. It can be one of:
  - animals: Livestock product, e.g. Bovine Meat, Butter, Ghee, etc.
  - crops: Crop product, e.g. Vegetables, Other, Oats, etc.
  - residues: Crop residue, e.g. Straw, Fodder legumes, etc.
  - grass: Grass, e.g. Grassland, Temporary grassland, etc.
  - scavenging: Other residues. Single Scavenging item.
- supply: The computed amount in tonnes of this item that should be fed to this animal, when sharing the total item feed use from the Commodity Balance Sheet among all livestock.

- intake: The actual amount in tonnes that the animal needs, which can be less than the theoretical used amount from supply.
- intake_dry_matter: The amount specified by intake but only considering dry matter, so it should be less than intake.
- loss: The amount that is not used for feed. This is supply - intake.
- loss_share: The percent that is lost. This is loss / supply.

Examples

```
get_feed_intake(example = TRUE)
```

`get_land_fp_production` *Get land footprint data for local production.*

Description

Read and clean the land_fp input file, returning only land-use entries from local production (Impact == "Land" and Origin == "Production").

Usage

```
get_land_fp_production(example = FALSE)
```

Arguments

example	If TRUE, return a small example output without downloading remote data. Default is FALSE.

Value

A tibble with columns:

- year: Year.
- area_code: Numeric area code.
- item_cbs_code: Commodity balance sheet item code.
- impact: Impact category.
- element: Source element.
- origin: Origin type.
- group: Group category.
- impact_u: Land footprint value (TODO: find which unit).

Examples

```
get_land_fp_production(example = TRUE)
```

get_primary_production *Primary items production*

Description

Get amount of crops, livestock and livestock products.

Usage

```
get_primary_production(example = FALSE)
```

Arguments

example	If TRUE, return a small example output without downloading remote data. Default is FALSE.

Value

A tibble with the item production data. It contains the following columns:

- year: The year in which the recorded event occurred.
- area_code: The code of the country where the data is from. For code details see e.g. add_area_name().
- item_prod_code: FAOSTAT internal code for each produced item.
- item_cbs_code: FAOSTAT internal code for each commodity balance sheet item. The commodity balance sheet contains an aggregated version of production items. This field is the code for the corresponding aggregated item.
- live_anim_code: Commodity balance sheet code for the type of livestock that produces the livestock product. It can be:
  - NA: The entry is not a livestock product.
  - Non-NA: The code for the livestock type. The name can also be retrieved by using add_item_cbs_name().
- unit: Measurement unit for the data. Here, keep in mind three groups of items: crops (e.g. Apples and products, Beans...), livestock (e.g. Cattle, dairy, Goats...) and livestock products (e.g. Poultry Meat, Offals, Edible...). Then the unit can be one of:
  - tonnes: Available for crops and livestock products.
  - ha: Hectares, available for crops.
  - t_ha: Tonnes per hectare, available for crops.
  - heads: Number of animals (stocks), available for livestock.
  - slaughtered_heads: Number of animals slaughtered, available for livestock.
  - LU: Standard Livestock Unit measure, available for livestock.
  - t_head: tonnes per head, available for livestock products.
  - t_LU: tonnes per Livestock Unit, available for livestock products.
- value: The amount of item produced, measured in unit.

Examples

```
get_primary_production(example = TRUE)
```

`get_primary_residues` *Crop residue items*

Description

Get type and amount of residue produced for each crop production item.

Usage

```
get_primary_residues(example = FALSE)
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A tibble with the crop residue data. It contains the following columns:

- `year`: The year in which the recorded event occurred.
- `area_code`: The code of the country where the data is from. For code details see e.g. `add_area_name()`.
- `item_cbs_code_crop`: FAOSTAT internal code for each commodity balance sheet item. This is the crop that is generating the residue.
- `item_cbs_code_residue`: FAOSTAT internal code for each commodity balance sheet item. This is the obtained residue. In the commodity balance sheet, this can be three different items right now:
  - `2105`: `Straw`
  - `2106`: `Other crop residues`
  - `2107`: `Firewood`

  These are actually not FAOSTAT defined items, but custom defined by us. When necessary, FAOSTAT codes are extended for our needs.
- `value`: The amount of residue produced, measured in tonnes.

Examples

```
get_primary_residues(example = TRUE)
```

`get_processing_coefs` *Processed products share factors*

Description

Reports quantities of commodity balance sheet items used for `processing` and quantities of their corresponding processed output items.

Usage

```
get_processing_coefs(example = FALSE)
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A tibble with the quantities for each processed product. It contains the following columns:

- `year`: The year in which the recorded event occurred.
- `area_code`: The code of the country where the data is from. For code details see e.g. `add_area_name()`.
- `item_cbs_code_to_process`: FAOSTAT internal code for each one of the items that are being processed and will give other subproduct items. For code details see e.g. `add_item_cbs_name()`.
- `value_to_process`: tonnes of this item that are being processed. It matches the amount found in the `processing` column from the data obtained by `get_wide_cbs()`.
- `item_cbs_code_processed`: FAOSTAT internal code for each one of the subproduct items that are obtained when processing. For code details see e.g. `add_item_cbs_name()`.
- `initial_conversion_factor`: estimate for the number of tonnes of `item_cbs_code_processed` obtained for each tonne of `item_cbs_code_to_process`. It will be used to compute the `final_conversion_factor`, which leaves everything balanced. TODO: explain how it's computed.
- `initial_value_processed`: first estimate for the number of tonnes of `item_cbs_code_processed` obtained from `item_cbs_code_to_process`. It is computed as `value_to_process * initial_conversion_factor`.
- `conversion_factor_scaling`: computed scaling needed to adapt `initial_conversion_factor` so as to get a final balanced total of subproduct quantities. TODO: explain how it's computed.
- `final_conversion_factor`: final used estimate for the number of tonnes of `item_cbs_code_processed` obtained for each tonne of `item_cbs_code_to_process`. It is computed as `initial_conversion_factor * conversion_factor_scaling`.
- `final_value_processed`: final estimate for the number of tonnes of `item_cbs_code_processed` obtained from `item_cbs_code_to_process`. It is computed as `initial_value_processed * final_conversion_factor`.

For the final data obtained, the quantities `final_value_processed` are balanced in the following sense: the total sum of `final_value_processed` for each unique tuple of `(year, area_code, item_cbs_code_processed)` should be exactly the quantity reported for that year, country and `item_cbs_code_processed` item in the `production` column obtained from `get_wide_cbs()`. This is because they are not primary products, so the amount from 'production' is actually the amount of subproduct obtained. TODO: Fix few data where this doesn't hold.

Examples

```
get_processing_coefs(example = TRUE)
```

---

get_wide_cbs *Commodity balance sheet data.*

Description

Retrieve supply and use parts for each commodity balance sheet (CBS) item. Stock variations are split into two non-negative columns following the FABIO methodology.

Usage

```
get_wide_cbs(example = FALSE)
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A tibble with the commodity balance sheet data in wide format. It contains the following columns:

- `year`: The year in which the recorded event occurred.
- `area_code`: The code of the country where the data is from. For code details see e.g. `add_area_name()`.
- `item_cbs_code`: FAOSTAT internal code for each item. For code details see e.g. `add_item_cbs_name()`.

The other columns are quantities where total supply and total use should be balanced. Units are tonnes for most items, and heads for live animals (see items_cbs `item_type`).

For supply:

- `production`: Produced locally.
- `import`: Obtained from importing from other countries.
- `stock_withdrawal`: Biomass taken out of storage (non-negative). Positive when stocks decrease.

For use:

- food: Food for humans.
- feed: Food for animals.
- export: Released as export for other countries.
- seed: Intended for new production.
- processing: Used to obtain other subproducts.
- other_uses: Any other use not included above.
- stock_addition: Biomass placed into storage (non-negative). Positive when stocks increase.

There is an additional column domestic_supply which is computed as total use excluding export.

Examples

```
get_wide_cbs(example = TRUE)
```

---

gleam_animal_weights *GLEAM animal weights.*

Description

Typical live weights by region, species, system, and cohort.

Usage

```
gleam_animal_weights
```

Format

A tibble with region, species, system, cohort, weight_kg.

Source

MacLeod et al. (2018) GLEAM 3.0.

Examples

```
gleam_animal_weights
```

gleam_crop_residue_nitrogen

Nitrogen parameters for crop residues of feed materials.

Description

Nitrogen content of above- and below-ground residues and root-to-shoot ratios for feed materials.

Usage

```
gleam_crop_residue_nitrogen
```

Format

A tibble with columns:

material_number Sequential material identifier.

material Feed material code.

n_ag Nitrogen content of above-ground residues.

rbg_bio Ratio of below-ground residues to above-ground biomass.

n_bg Nitrogen content of below-ground residues.

species_group `"ruminant"` or `"monogastric"`.

Source

MacLeod et al. (2018) GLEAM 3.0 Supplement S1, Tables S.6.7 and S.6.8.

Examples

```
gleam_crop_residue_nitrogen
```

gleam_crop_residue_params

GLEAM crop residue parameters.

Description

Dry matter content and parameters for calculating crop residue yield by crop type.

Usage

```
gleam_crop_residue_params
```

Format

A tibble with columns:

crop Crop name.

dry_matter_pct Dry matter content (percent).

slope Slope for residue yield calculation.

intercept Intercept for residue yield calculation.

Source

MacLeod et al. (2018) GLEAM 3.0 Supplement S1, Table S.3.1. doi:10.1088/17489326/aad4d8

Examples

```
gleam_crop_residue_params
```

---

gleam_dressing_percentages

GLEAM dressing percentages.

Description

Carcass weight as percentage of live weight by species, production system, cohort, and GLEAM region. Includes country-specific overrides for industrial pig systems in Western Europe.

Usage

```
gleam_dressing_percentages
```

Format

A tibble with columns:

species Animal species (Cattle, Buffaloes, Sheep, Goats, Pigs, Chicken).

production_system Production system (Dairy, Beef, Backyard, Intermediate, Industrial, Layers, Broilers). NA for species without system breakdown.

cohort Cohort (e.g. Adult and replacement female). NA for species without cohort breakdown.

country Country name for country-specific values. NA for regional values.

gleam_region GLEAM region abbreviation (NA, RUS, WE, EE, NENA, ESEA, OCE, SA, LAC, SSA).

dressing_percent Dressing percentage.

Source

MacLeod et al. (2018) GLEAM 3.0 Supplement S1, Table S.9.1. doi:10.1088/17489326/aad4d8

Examples

```
gleam_dressing_percentages
```

`gleam_energy_use_ef` *Energy use emission factors by species and system.*

Description

Emission factors for embedded and direct energy use in livestock production, by species, production system, and climate zone.

Usage

```
gleam_energy_use_ef
```

Format

A tibble in long format with columns:

grouping Country or country group (e.g. `"OECD"`, `"EU 27"`).

species Animal species or group (e.g. `"dairy_cattle"`, `"pigs"`).

system Production system (e.g. `"Grassland based"`, `"industrial"`). `NA` when not applicable.

climate Climate zone (`"arid"`, `"humid"`, `"temperate"`). `NA` when not applicable.

energy_type `"embedded"` or `"direct"`.

emission_factor Emission factor in kg CO2-eq per unit product.

Source

MacLeod et al. (2018) GLEAM 3.0 Supplement S1, Tables S.7.1 through S.7.7.

Examples

```
gleam_energy_use_ef
```

gleam_enteric_params *GLEAM enteric fermentation parameters.*

Description

Ym (% GE) values by species and production system. Feedlot cattle use 3.0% per IPCC 2019 Table 10.12.

Usage

```
gleam_enteric_params
```

Format

A tibble with `species`, `system`, `ym_percent`, `notes`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.12.

Examples

```
gleam_enteric_params
```

gleam_feed_categories *GLEAM feed categories.*

Description

Feed classification used in GLEAM 3.0.

Usage

```
gleam_feed_categories
```

Format

A tibble with `feed_category`, `feed_type`, `description`.

Source

MacLeod et al. (2018) GLEAM 3.0.

Examples

```
gleam_feed_categories
```

gleam_feed_composition

GLEAM feed use efficiency.

Description

Regional feed use efficiency (FUE) values for forages and crop residues of ruminant species.

Usage

```
gleam_feed_composition
```

Format

A tibble with columns:

feed_group Feed material group (1-6 or 9-15).

feed_type Feed type (mixed, grassland, or all).

gleam_region GLEAM geographic region.

feed_use_efficiency FUE value (0-1 fraction).

Source

MacLeod et al. (2018) GLEAM 3.0 Supplement S1, Table S.3.2.

Examples

```
gleam_feed_composition
```

gleam_feed_conversion_ratios

GLEAM feed conversion ratios for monogastrics.

Description

Nutritional values for feed materials of monogastric species (chicken and pigs).

Usage

```
gleam_feed_conversion_ratios
```

Format

A tibble with columns:

number Feed material number.

material Feed material code.

gross_energy_j_kg Gross energy (J per kg).

n_content_g_kg Nitrogen content (g per kg DM).

me_chicken_j_kg Metabolisable energy for chicken (J per kg).

me_pigs_j_kg Metabolisable energy for pigs (J per kg).

digestibility_pct Digestibility (percent).

Source

MacLeod et al. (2018) GLEAM 3.0 Supplement S1, Table S.3.4.

Examples

```
gleam_feed_conversion_ratios
```

---

gleam_feed_digestibility

GLEAM feed digestibility for ruminants.

Description

Nutritional values for feed materials of ruminant species, including gross energy, nitrogen content, and digestibility.

Usage

```
gleam_feed_digestibility
```

Format

A tibble with columns:

number Feed material number.

material Feed material code.

gross_energy_mj_kg Gross energy (MJ per kg DM).

n_content_g_kg Nitrogen content (g per kg DM).

digestibility_pct Digestibility (percent).

Source

MacLeod et al. (2018) GLEAM 3.0 Supplement S1, Table S.3.3.

Examples

```
gleam_feed_digestibility
```

`gleam_field_operation_ef` *Emission factors for field operations on feed materials.*

Description

CO2-equivalent emissions per hectare from field operations for ruminant and monogastric feed materials.

Usage

```
gleam_field_operation_ef
```

Format

A tibble with columns:

material_number Sequential material identifier.

material Feed material code (e.g. `"GRASSF"`, `"WHEAT"`).

emission_factor_kg_co2eq_ha Emission factor in kg CO2-eq per hectare.

species_group `"ruminant"` or `"monogastric"`.

Source

MacLeod et al. (2018) GLEAM 3.0 Supplement S1, Tables S.6.1 and S.6.2.

Examples

```
gleam_field_operation_ef
```

`gleam_fracremove` *Country-level fraction of crop residues removed.*

Description

Countries whose FracReMove value differs from the GLEAM default.

Usage

```
gleam_fracremove
```

Format

A tibble with columns:

country Country name.

continent Continent.

region GLEAM region.

fracremove Fraction of crop residues removed (0 to 1).

Source

MacLeod et al. (2018) GLEAM 3.0 Supplement S1, Table S.6.9.

Examples

```
gleam_fracremove
```

---

gleam_geographic_hierarchy

GLEAM geographic hierarchy.

Description

Maps countries (ISO3) to GLEAM regions, FAOSTAT regions, and classification indicators.

Usage

```
gleam_geographic_hierarchy
```

Format

A tibble with columns:

iso3 ISO3 country code.

country Country name.

continent Continent.

faostat_region FAOSTAT regional grouping.

gleam_region GLEAM regional grouping.

Source

MacLeod et al. (2018) GLEAM 3.0 Supplement S1, Tables S.A1-S.A2. doi:10.1088/17489326/aad4d8

Examples

```
gleam_geographic_hierarchy
```

gleam_livestock_categories

GLEAM livestock categories.

Description

Species, production systems, and cohort definitions from GLEAM 3.0.

Usage

```
gleam_livestock_categories
```

Format

A tibble with columns:

species Animal species.

production_system Dairy, Beef, Meat, etc.

cohort Age/sex cohort.

description Cohort description.

Source

MacLeod et al. (2018) GLEAM 3.0 Model Description.

Examples

```
gleam_livestock_categories
```

gleam_mechanization_levels

Country-level mechanization levels for feed materials.

Description

Mechanization level by country for each feed material, for ruminant and monogastric species.

Usage

```
gleam_mechanization_levels
```

Format

A tibble in long format with columns:

country Country name.

continent Continent.

region GLEAM region.

feed_material Feed material code in lowercase.

mechanization_level Numeric mechanization level.

species_group `"ruminant"` or `"monogastric"`.

Source

MacLeod et al. (2018) GLEAM 3.0 Supplement S1, Tables S.6.3 and S.6.4.

Examples

```
gleam_mechanization_levels
```

---

gleam_milk_production *GLEAM milk production.*

Description

Average annual milk yields and lactation lengths by region.

Usage

```
gleam_milk_production
```

Format

A tibble with `region`, `species`, `system`, `milk_kg_head_yr`, `lactation_days`.

Source

MacLeod et al. (2018) GLEAM 3.0.

Examples

```
gleam_milk_production
```

gleam_mms_shares *GLEAM manure management system shares.*

Description

Regional MMS allocation by species and system.

Usage

```
gleam_mms_shares
```

Format

A tibble with `region`, `species`, `system`, `mms`, `share_percent`.

Source

MacLeod et al. (2018) GLEAM 3.0.

Examples

```
gleam_mms_shares
```

gleam_processing_transport_ef *Processing and transport emission factors for feeds.*

Description

Emission factors for processing and transport of feed materials, for ruminant and monogastric species.

Usage

```
gleam_processing_transport_ef
```

Format

A tibble with columns:

material_number Sequential material identifier.

material Feed material code.

processing_g_co2eq_kg_dm Processing emission factor in g CO2-eq per kg dry matter.

transport_g_co2eq_kg_dm Transport emission factor in g CO2-eq per kg dry matter.

species_group `"ruminant"` or `"monogastric"`.

Source

MacLeod et al. (2018) GLEAM 3.0 Supplement S1, Tables S.6.5 and S.6.6.

Examples

```
gleam_processing_transport_ef
```

---

grazing_energy_coefs *Grazing energy coefficients.*

Description

Walking energy cost for grazing animals (MJ/kg body weight/km).

Usage

```
grazing_energy_coefs
```

Format

A tibble with `parameter`, `value_mj_kg_km`, `source`.

Source

NRC 2001 (0.00045 Mcal/kg/km converted to MJ).

Examples

```
grazing_energy_coefs
```

---

harmonize_interpolate *Harmonize advanced cases with interpolation for 1:N groups*

Description

Harmonize data containing `"simple"` and `"1:n"` mappings. `"simple"` covers both 1:1 and N:1 relationships (values are summed). For `"1:n"` groups (one original item splits into several harmonized items) this function computes value shares across the full year range, interpolates missing shares, and applies them to split values.

Important for 1:n mappings: For each original item that splits into multiple harmonized items (e.g., "wheatrice" into "wheat" and "rice"), provide **one row per target** `item_code_harm`. Each row should have the same `item`, `year`, and `value`, differing only in `item_code_harm`. For example, to disaggregate "wheatrice":

- Row 1: `item` = "wheatrice", `item_code_harm` = 1
- Row 2: `item` = "wheatrice", `item_code_harm` = 2

Do not provide a single row; the function will not create duplicates automatically.

Usage

```
harmonize_interpolate(data, ...)
```

Arguments

`data`	A data frame containing at least columns: • `item`: String, original item name. • `item_code_harm`: Numeric, code for harmonized item. • `year`: Numeric, year of observation. • `value`: Numeric, value of observation. • `type`: String, `"simple"` or `"1:n"`.
`...`	Additional grouping columns provided as bare names.

Value

A tibble with columns:

- `item_code`: Numeric, code for harmonized item.
- `year`: Numeric, year of observation.
- `value`: Numeric, summed value of observation.
- and any additional grouping columns.

Examples

```
# Simple-only data (no 1:n rows)
df_simple <- tibble::tribble(
  ~item, ~item_code_harm, ~year, ~value, ~type,
  "wheat", 1, 2000, 5, "simple",
  "barley", 2, 2000, 3, "simple",
  "oats", 2, 2000, 2, "simple"
)
harmonize_interpolate(df_simple)

# Mixed simple + 1:n data
df_mixed <- tibble::tribble(
  ~item, ~item_code_harm, ~year, ~value, ~type,
  "wheatrice", 1, 2000, 20, "1:n",
  "wheatrice", 2, 2000, 20, "1:n",
  "wheat", 1, 2000, 8, "simple",
  "rice", 2, 2000, 12, "simple"
)
harmonize_interpolate(df_mixed)

# Multiple years with share interpolation
# Shares are known in 2000 and 2002; 2001 is interpolated.
df_years <- tibble::tribble(
  ~item, ~item_code_harm, ~year, ~value, ~type,
  "wheat", 1, 2000, 6, "simple",
  "rice", 2, 2000, 4, "simple",
```

```
  "wheatrice", 1, 2001, 10, "1:n",
  "wheatrice", 2, 2001, 10, "1:n",
  "wheat", 1, 2002, 8, "simple",
  "rice", 2, 2002, 2, "simple"
)
harmonize_interpolate(df_years)

# With extra grouping columns
df_grouped <- tibble::tribble(
  ~item, ~item_code_harm, ~year, ~value, ~type, ~country,
  "wheat", 1, 2000, 6, "simple", "usa",
  "rice", 2, 2000, 4, "simple", "usa",
  "wheatrice", 1, 2001, 10, "1:n", "usa",
  "wheatrice", 2, 2001, 10, "1:n", "usa",
  "wheat", 1, 2002, 8, "simple", "usa",
  "rice", 2, 2002, 2, "simple", "usa",
  "wheat", 1, 2002, 8, "simple", "germany"
)
harmonize_interpolate(df_grouped, country)
```

`harmonize_simple` *Harmonize rows labeled "simple" by summing values*

Description

Sum `value` for rows where `type == "simple"`. This covers both 1:1 and N:1 item mappings, since in both cases the values are simply summed. The results are grouped by `item_code_harm`, `year` and any additional grouping columns supplied via `...`.

Usage

```
harmonize_simple(data, ...)
```

Arguments

`data`	A data frame containing at least columns: • `item_code_harm`: Numeric, code for harmonized item. • `year`: Numeric, year of observation. • `value`: Numeric, value of observation. • `type`: String, harmonization type. Only `"simple"` rows are used.
`...`	Additional grouping columns supplied as bare names.

Value

A tibble with columns:

- `item_code_harm`: Numeric, code for harmonized item.
- `year`: Numeric, year of observation.
- `value`: Numeric, summed value of observation.
- and any additional grouping columns.

Examples

```
# 1:1 mapping: one original item -> one harmonized code
df_one_to_one <- tibble::tribble(
  ~item_code_harm, ~year, ~value, ~type,
  1, 2000, 10, "simple",
  2, 2000, 3, "simple",
  1, 2001, 12, "simple",
  2, 2001, 5, "simple"
)
harmonize_simple(df_one_to_one)

# N:1 mapping: multiple items map to the same code
df_many_to_one <- tibble::tribble(
  ~item_code_harm, ~year, ~value, ~type,
  1, 2000, 4, "simple",
  1, 2000, 6, "simple",
  2, 2000, 3, "simple"
)
harmonize_simple(df_many_to_one)

# With an extra grouping column (e.g. country)
df_grouped <- tibble::tribble(
  ~item_code_harm, ~year, ~value, ~type, ~country,
  1, 2000, 4, "simple", "usa",
  1, 2000, 6, "simple", "usa",
  1, 2000, 9, "simple", "germany",
  2, 2000, 3, "simple", "usa"
)
harmonize_simple(df_grouped, country)

# Rows with type != "simple" are ignored
df_mixed <- tibble::tribble(
  ~item_code_harm, ~year, ~value, ~type,
  1, 2000, 10, "simple",
  1, 2000, 99, "1:n",
  2, 2000, 3, "simple"
)
harmonize_simple(df_mixed)
```

`indirect_n2o_ef` *Indirect N2O emission factors.*

Description

Parameters for indirect N2O emissions from manure management: EF4 (volatilization), EF5 (leaching), FracGasMS, FracLeach.

Usage

```
indirect_n2o_ef
```

Format

A tibble with `parameter`, `value`, `description`.

Source

IPCC 2019, Vol 4, Ch 10, Table 10.22; Vol 4, Ch 11, Table 11.3.

Examples

```
indirect_n2o_ef
```

---

ipcc_2006_enteric_ef *IPCC 2006 Tier 1 enteric emission factors.*

Description

Table 10.11 (2006): Tier 1 regional EFs for enteric fermentation.

Usage

```
ipcc_2006_enteric_ef
```

Format

A tibble with `region`, `category`, `ef_kg_head_yr`.

Source

IPCC 2006, Vol 4, Ch 10, Table 10.11.

Examples

```
ipcc_2006_enteric_ef
```

ipcc_2006_manure_ef *IPCC 2006 Tier 1 manure emission factors.*

Description

Table 10.14 (2006): Tier 1 regional EFs for manure CH4.

Usage

```
ipcc_2006_manure_ef
```

Format

A tibble with `region`, `category`, `ef_kg_head_yr`, `temp_zone`.

Source

IPCC 2006, Vol 4, Ch 10, Table 10.14.

Examples

```
ipcc_2006_manure_ef
```

ipcc_2006_mcf_temp *IPCC 2006 MCF by temperature.*

Description

Table 10.17 (2006): MCF values by MMS type and annual temperature.

Usage

```
ipcc_2006_mcf_temp
```

Format

A tibble with `system`, `temp_c`, `mcf_percent`.

Source

IPCC 2006, Vol 4, Ch 10, Table 10.17.

Examples

```
ipcc_2006_mcf_temp
```

ipcc_2019_bo *IPCC 2019 Bo values (Table 10.16).*

Description

Maximum CH4 producing capacity of manure (m3 CH4/kg VS). Dairy cattle (0.24) differs from other cattle (0.18).

Usage

```
ipcc_2019_bo
```

Format

A tibble with `category`, `bo_m3_kg_vs`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.16.

Examples

```
ipcc_2019_bo
```

ipcc_2019_cfi *IPCC 2019 Cfi values (Table 10.4).*

Description

Net energy maintenance coefficients (MJ/day/kg^0.75). Dairy (lactating) cattle use 0.386; non-dairy 0.322.

Usage

```
ipcc_2019_cfi
```

Format

A tibble with `category`, `subcategory`, `cfi_mj_day_kg075`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.4.

Examples

```
ipcc_2019_cfi
```

ipcc_2019_enteric_ef_cattle

IPCC 2019 enteric EF for cattle.

Description

Table 10.10: Tier 1 enteric fermentation emission factors for cattle by region (kg CH4/head/yr).

Usage

```
ipcc_2019_enteric_ef_cattle
```

Format

A tibble with `region`, `category`, `ef_kg_head_yr`, `source`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.10.

Examples

```
ipcc_2019_enteric_ef_cattle
```

ipcc_2019_enteric_ef_other

IPCC 2019 enteric EF for non-cattle.

Description

Table 10.11: Tier 1 enteric fermentation emission factors for non-cattle species (kg CH4/head/yr).

Usage

```
ipcc_2019_enteric_ef_other
```

Format

A tibble with `category`, `ef_kg_head_yr`, `source`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.11.

Examples

```
ipcc_2019_enteric_ef_other
```

`ipcc_2019_manure_ch4_ef_cattle`
IPCC 2019 manure CH4 EF for cattle.

Description

Table 10.14: Tier 1 manure management CH4 emission factors for cattle by region (kg CH4/head/yr).

Usage

```
ipcc_2019_manure_ch4_ef_cattle
```

Format

A tibble with `region`, `category`, `ef_kg_head_yr`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.14.

Examples

```
ipcc_2019_manure_ch4_ef_cattle
```

`ipcc_2019_manure_ch4_ef_other`
IPCC 2019 manure CH4 EF for non-cattle.

Description

Table 10.14: Tier 1 manure management CH4 emission factors for non-cattle species (kg CH4/head/yr).

Usage

```
ipcc_2019_manure_ch4_ef_other
```

Format

A tibble with `category`, `ef_kg_head_yr`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.14.

Examples

```
ipcc_2019_manure_ch4_ef_other
```

ipcc_2019_mcf_manure *IPCC 2019 MCF for manure management.*

Description

Table 10.17: Methane Conversion Factors by manure management system and annual average temperature.

Usage

```
ipcc_2019_mcf_manure
```

Format

A tibble with `system`, `annual_temp_c`, `mcf_percent`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.17.

Examples

```
ipcc_2019_mcf_manure
```

ipcc_2019_n_excretion *IPCC 2019 nitrogen excretion rates.*

Description

Table 10.19: Daily N excretion rates by species and region (kg N/1000 kg animal mass/day).

Usage

```
ipcc_2019_n_excretion
```

Format

A tibble with `region`, `category`, `nex_kg_per_1000kg_day`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.19.

Examples

```
ipcc_2019_n_excretion
```

ipcc_2019_n2o_ef_direct *IPCC 2019 direct N2O emission factors.*

Description

Table 10.21: EF3 values (kg N2O-N/kg N) by manure management system.

Usage

```
ipcc_2019_n2o_ef_direct
```

Format

A tibble with `mms_type`, `ef3_kg_n2on_kg_n`, `source`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.21.

Examples

```
ipcc_2019_n2o_ef_direct
```

ipcc_2019_ym *IPCC 2019 Ym values (Table 10.12).*

Description

Methane conversion rate (% GE) by species and feed situation. The 2019 Refinement differentiates:

- Cattle feedlot (>90% concentrate): 3.0%.
- Sheep >= 75 kg body weight: 6.7%.
- Sheep < 75 kg body weight: 4.7%.

Usage

```
ipcc_2019_ym
```

Format

A tibble with `category`, `feed_situation`, `ym_percent`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.12.

Examples

```
ipcc_2019_ym
```

`ipcc_tier2_bo_values` *Tier 2 Bo values.*

Description

Maximum CH4 producing capacity by detailed category. Dairy cattle 0.24 vs other cattle 0.18.

Usage

```
ipcc_tier2_bo_values
```

Format

A tibble with `category`, `bo_m3_kg_vs`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.16.

Examples

```
ipcc_tier2_bo_values
```

`ipcc_tier2_energy_coefs`
Tier 2 energy coefficients.

Description

Coefficients for IPCC Tier 2 GE calculation including Cfi (maintenance), Ca (activity), Cp (pregnancy), Cw (work), and energy content of weight gain. Now includes `subcategory` column to differentiate dairy (lactating) vs non-dairy cattle.

Usage

```
ipcc_tier2_energy_coefs
```

Format

A tibble with columns:

category Species (Cattle, Buffalo, Sheep, etc.).

subcategory Dairy, Non-Dairy, or All.

cfi_mj_day_kg075 NEm coefficient (MJ/day/kg^0.75).

ca_pasture Activity coefficient for grazing.

ca_feedlot Activity coefficient for confined.

cp Pregnancy coefficient.

cw Work coefficient.

energy_content_gain_mj_kg Energy per kg gain.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Eq 10.3-10.16; Tables 10.4-10.5.

Examples

```
ipcc_tier2_energy_coefs
```

---

ipcc_tier2_manure_ash *Tier 2 manure ash content.*

Description

Ash content of manure as percent of dry matter, used in VS calculation (Eq 10.24).

Usage

```
ipcc_tier2_manure_ash
```

Format

A tibble with `category`, `ash_percent`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10.

Examples

```
ipcc_tier2_manure_ash
```

ipcc_tier2_n_retention *Tier 2 nitrogen retention fractions.*

Description

Fraction of N intake retained in animal products. Dairy cattle 0.20 vs other cattle 0.07.

Usage

```
ipcc_tier2_n_retention
```

Format

A tibble with `category`, `n_retention_frac`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.20.

Examples

```
ipcc_tier2_n_retention
```

ipcc_tier2_ym_values *Tier 2 Ym values.*

Description

Methane conversion rate by species and feed situation for Tier 2 enteric CH4. Includes feedlot distinction and sheep body weight differentiation.

Usage

```
ipcc_tier2_ym_values
```

Format

A tibble with `category`, `feed_situation`, `ym_percent`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.12.

Examples

```
ipcc_tier2_ym_values
```

items_cbs *Commodity balance sheet items*

Description

Defines name/code correspondences for commodity balance sheet (CBS) items.

Usage

```
items_cbs
```

Format

A tibble where each row corresponds to one CBS item. It contains the following columns:

- `item_cbs_code`: A numeric code used to refer to the CBS item.
- `item_cbs_name`: A natural language name for the item.
- `item_type`: An ad-hoc grouping of items. This is a work in progress evolving depending on our needs, so for now it only has two possible values:
  - `livestock`: The CBS item represents a live animal.
  - `other`: Not any of the previous groups.

Source

Inspired by FAOSTAT data.

items_full *Full CBS item table*

Description

Extended item reference table covering all CBS items, including their process and commodity codes, feed type classifications, and default material flow destinations.

Usage

```
items_full
```

Format

A tibble where each row corresponds to one CBS item. It contains the following columns:

- `item_cbs`: Name of the CBS item.
- `item_cbs_code`: Numeric CBS item code.
- `comm_code`: Commodity code used in process-based modelling (may contain `"#N/A"` when not applicable).
- `proc_code`: Process code (may contain `"#N/A"` when not applicable).
- `proc`: Process name (may contain `"#N/A"` when not applicable).
- `unit`: Measurement unit (typically `"tonnes"`).
- `group`: Broad item group. Common values include `"Additives"`, `"Crop products"`, `"Crop residues"`, `"Draught"`, `"Fish"`, `"Forestry"`, `"Grass"`, `"Livestock"`, and others.
- `feedtype_graniv`: Feed type classification for granivores (e.g., `"additives"`, `"concentrates"`, `"roughages"`).
- `feedtype_grazers`: Feed type classification for grazers.
- `comm_group`: Sub-group of the commodity (e.g., `"Additives"`, `"Alcohol"`, `"Ethanol"`, `"Oil cakes"`, `"Other processing residues"`).
- `Cat_1`: Primary category label used in material flow accounting.
- `Name_biomass`: Corresponding item name in `biomass_coefs`, enabling joins with the biomass coefficient table.
- `dbMFA_items`: Item identifier used in the material flow analysis database.
- `FEDNA`: Item name used in FEDNA feed composition tables.
- `default_destiny`: Default CBS use category for this item. One of `"Feed"`, `"Food"`, `"Other_uses"`, `"Processing"`, or `NA`.

Source

Derived from FAOSTAT data and internal commodity classification work.

Examples

```
head(items_full)
```

---

items_prim *Primary production items linked to CBS*

Description

Maps FAOSTAT primary production items and crop products to their CBS counterparts, along with farm and labour classifications.

Usage

```
items_prim
```

Format

A tibble where each row corresponds to one production item. It contains the following columns:

- `item_prod`: Name of the production item (e.g., `"Wheat"`, `"Rice"`).
- `item_prod_code`: FAOSTAT production item code (character).
- `item_cbs`: Name of the corresponding CBS item.
- `item_cbs_code`: Numeric CBS item code.
- `Farm_class`: Farm system classification. Crop items use codes such as `"COP"` (cereals, oilseeds, protein crops), `"Vegetables"`, `"Fruits"`, `"Olive"`, `"Grapevine"`, `"Other_crops"`. Livestock items use `"Dairy_cows"`, `"Cattle"`, `"Monogastric"`, `"Sheep_goats"`, `"Bees"`, `"Game"`. `NA` for non-farm items.
- `Cat_Labour`: Labour category used in agricultural labour analyses.
- `Cat_FAO1`: Top-level FAO commodity category.
- `group`: Item group classification. One of `"Primary crops"`, `"Crop products"`, `"Livestock products"`, `"Grass"`, `"Crop residues"`, `"Scavenging"`, `"Livestock"`.

Source

Derived from FAOSTAT Production data.

Examples

```
head(items_prim)
```

`items_prod` *Primary production items*

Description

Defines name/code correspondences for production items.

Usage

```
items_prod
```

Format

A tibble where each row corresponds to one production item. It contains the following columns:

- `item_prod_code`: A numeric code used to refer to the item.
- `item_prod_name`: A natural language name for the item.
- `item_type`: An ad-hoc grouping of items. This is a work in progress evolving depending on our needs, so for now it only has two possible values:
  - `crop_product`: The CBS item represents a crop product.
  - `other`: Not any of the previous groups.

Source

Inspired by FAOSTAT data.

`items_prod_full` *Full production item table*

Description

Comprehensive reference table for all production items, combining CBS linkages, biomass names, multiple classification schemes, and crop ecological traits.

Usage

```
items_prod_full
```

Format

A tibble where each row corresponds to one production item. It contains the following columns:

- `item_prod`: Name of the production item.
- `item_prod_code`: FAOSTAT production item code (character).
- `item_cbs`: Name of the corresponding CBS item.
- `item_cbs_code`: Numeric CBS item code.
- `group`: Item group. One of `"Primary crops"`, `"Crop products"`, `"Livestock products"`, `"Crop residues"`, `"Grassland"`, `"Scavenging"`.
- `live_anim`: Name of the parent live animal for livestock-derived items (`NA` for crop items).
- `live_anim_code`: Numeric CBS code of the parent live animal (`NA` for crop items).
- `Cat_Krausmann`: Item category used in Krausmann et al. biomass flow accounting.
- `Name_biomass`: Corresponding item name in `biomass_coefs`, enabling joins for biomass coefficients.
- `Name_Eurostat`: Corresponding item name in Eurostat agricultural statistics.
- `Name`: Alternative or display name for the item.
- `Cat_Labour`: Labour category used in agricultural labour analyses.
- `Cat_FAO1`: Top-level FAO commodity category (e.g., `"Cereals"`, `"Oilcrops"`).
- `Cat_Origin`: Origin-based commodity category. One of `"Cereals"`, `"Vegetables and Fruits"`, `"Sugar and Stimulants"`, `"Oil Crops"`, `"Fodder crops"`, `"Fibres and Crude Materials"`, or `NA`.
- `Cat_Use`: Use-based commodity category (e.g., `"Grains"`, `"Oils and Fats"`, `"Fodder crops"`, `"Beverages, sugar and stimulants"`).
- `Order`: Numeric ordering field used for sorting items consistently across outputs.
- `Categ`: General category label used in some analyses.
- `Farm_class`: Farm system classification (see `items_prim` for values).

- c3_c4: Photosynthetic pathway. One of "c3", "c4", or NA.
- ann_per_nfx: Annual/perennial and nitrogen fixation trait. One of "ann" (annual), "per" (perennial), "nfx" (nitrogen-fixing), or NA.
- Cat_1: Primary category label for material flow accounting.
- Cat_2: Secondary category label.
- Cat_3: Tertiary category label.
- Cat_4: Quaternary category label.
- Herb_Woody: Plant growth form. One of "Herbaceous", "Woody", or NA.
- Crop_irrig: Irrigation category used in water-use analyses.
- Cat_Org: Organic farming category classification.
- Cat_Ymax: Maximum attainable yield category.
- Cat_Ymax_leg: Legend label for the Cat_Ymax category.

Source

Derived from FAOSTAT Production data and multiple classification schemes from the literature.

Examples

```
head(items_prod_full)
```

---

lassaletta_grassland_share

Grassland share of synthetic nitrogen by country and year

Description

Country-level time series of the share of synthetic nitrogen applied to grassland (versus cropland). Used to split national N totals between land use types in the WHEP nitrogen pipeline.

Usage

```
lassaletta_grassland_share
```

Format

A tibble with one row per country-year combination containing:

- Country: Country name.
- year: Year (numeric).
- grass_share: Share of synthetic N applied to grassland (0–1).

Source

Lassaletta et al. nitrogen flow dataset. See pipeline documentation for full citation.

Examples

```
head(lassaletta_grassland_share)
```

---

liv_lu_coefs *Livestock unit coefficients*

Description

Provides livestock unit (LU) conversion factors per head for each animal class, used to express heterogeneous livestock populations in comparable units.

Usage

```
liv_lu_coefs
```

Format

A tibble where each row corresponds to one animal class. It contains the following columns:

- `Animal_class`: Animal class identifier (e.g., `"Dairy_cows"`, `"Cattle"`, `"Sheep_goats"`, `"Broilers"`, `"Hens"`, `"Pigs"`).
- `LU_head`: Livestock units per head (numeric). Dairy cows have a value of 1.0 by convention; smaller animals have proportionally lower values.

Source

Based on standard livestock unit definitions from FAO and European agricultural statistics.

Examples

```
head(liv_lu_coefs)
```

---

livestock_constants *Livestock physical constants.*

Description

Named list of physical constants used in livestock emission calculations:

- `energy_ch4_mj_kg`: 55.65 MJ/kg CH4.
- `ge_content_mj_kg_dm`: 18.45 MJ/kg DM.
- `ue_factor`: 0.04 (urinary energy as fraction of GE).
- `n_to_protein`: 6.25 (N to protein conversion).
- `default_de_percent`: 65%.
- `ev_wool_mj_kg`: 24.0 MJ/kg clean wool.

Usage

```
livestock_constants
```

Format

A named list.

Source

IPCC 2019 Refinement, Vol 4, Ch 10.

Examples

```
str(livestock_constants)
```

---

livestock_production_defaults

Default production parameters.

Description

Default values for fat%, protein%, lactose%, weight gain, work hours, and pregnancy fraction by species.

Usage

```
livestock_production_defaults
```

Format

A tibble with columns:

category Species or animal class.

fat_percent Milk fat content (percent).

protein_percent Milk protein content (percent).

lactose_percent Milk lactose content (percent).

weight_gain_kg_day Average daily weight gain (kg/day).

work_hours_day Hours of draft work per day.

pregnant_fraction Fraction of females pregnant.

Source

NRC 2001; IPCC 2019, Vol 4, Ch 10.

Examples

```
livestock_production_defaults
```

mueller_synthetic_n *Synthetic nitrogen application rates by crop and country*

Description

Country- and crop-process-level synthetic nitrogen application rates (kg N ha^{-1}), derived from Mueller et al. (2012). Used as reference crop-specific N rates in the WHEP nitrogen pipeline.

Usage

```
mueller_synthetic_n
```

Format

A tibble with one row per crop-process-country combination containing:

- `proc_code`: Internal process code (e.g. `"p001"`).
- `crop_process`: Descriptive crop process name (e.g. `"Rice production"`).
- `crop_original`: Crop name as in the source dataset.
- `unit`: Unit of the rate value (always `"kgN/ha"`).
- `iso3c`: ISO 3166-1 alpha-3 country code.
- `rate_value`: Nitrogen application rate (kg N ha^{-1}).

Source

Mueller, N. D. et al. (2012). Closing yield gaps through nutrient and water management. *Nature*, 490(7419), 254–257. doi:10.1038/nature11420

Examples

```
head(mueller_synthetic_n)
```

plot_footprint_sankey *Interactive footprint Sankey viewer*

Description

Create a browser-based Sankey viewer from a footprint table, such as the output of `compute_footprint()`. The viewer includes stage filters, a per-stage node limit controls, a minimum-flow threshold, hover tooltips, click-to-highlight interactions, and SVG download.

Each row is treated as one path through the selected `stages`, with `value_col` giving the path size. Rows are summed before plotting, so repeated paths are shown as one flow.

The default stages are `origin_area`, `origin_item`, `target_item`, and `target_area`; `target_fd` is appended automatically when present. If columns such as `origin_area_name` or `target_item_name` are present, they are used as display labels.

Usage

```
plot_footprint_sankey(
  footprints,
  stages = NULL,
  value_col = "value",
  label_cols = NULL,
  max_nodes = 10,
  other_label = "Other",
  stage_max_nodes = NULL,
  stage_other_labels = NULL,
  embed_max_nodes = Inf,
  stage_embed_max_nodes = NULL,
  min_share = 0,
  title = "Footprint Sankey Viewer",
  subtitle = NULL,
  value_label = "footprint",
  width = "100%",
  height = 680,
  file = NULL,
  open = FALSE
)
```

Arguments

`footprints`	A data frame with footprint paths and a numeric value column.
`stages`	Character vector of columns to use as Sankey stages. Defaults to the footprint origin and target columns.
`value_col`	Name of the numeric column containing flow values.
`label_cols`	Optional named character vector mapping stage columns to display-label columns, for example `c(origin_area = "origin_area_name")`. When omitted, `{stage}_name` columns are used where available.
`max_nodes`	Maximum number of individual nodes to show per stage in the current filtered browser view. Less important nodes are grouped into `other_label`. Use `Inf` to show all nodes.
`other_label`	Label used for grouped nodes when `max_nodes` is finite.
`stage_max_nodes`	Optional named numeric vector overriding `max_nodes` for selected stages, for example `c(product = 75, target_area = 12)`.
`stage_other_labels`	Optional named character vector overriding `other_label` for selected stages, for example `c(product = "Other products")`.
`embed_max_nodes`	Maximum number of individual nodes to embed per stage before writing the viewer. Less important nodes are permanently grouped before serialization, reducing file size. Use `Inf` to keep all nodes available to the browser.

stage_embed_max_nodes
: Optional named numeric vector overriding embed_max_nodes for selected stages.

min_share
: Minimum visible path size as a percentage of the current filtered total. Set to 0 to keep diffuse trade links visible. Users can change this in the browser.

title
: Optional viewer title.

subtitle
: Optional viewer subtitle.

value_label
: Label used in tooltips and the summary line.

width
: Viewer width as a CSS value.

height
: Sankey SVG height in pixels.

file
: Optional path to write a standalone HTML viewer.

open
: If TRUE and file is supplied, open the written viewer in the default browser. If TRUE and file is NULL, a temporary HTML file is written and opened.

Value

A browsable HTML object when file is NULL and open is FALSE; otherwise invisibly returns the HTML file path.

Examples

```
footprints <- tibble::tribble(
  ~origin_area, ~origin_item, ~target_item, ~target_area, ~value,
  "Brazil", "Soybeans", "Pigmeat", "China", 10,
  "Brazil", "Soybeans", "Milk", "China", 4,
  "Brazil", "Soybeans", "Soybean Oil", "France", 3
)

if (
  interactive() &&
    requireNamespace("htmltools", quietly = TRUE) &&
    requireNamespace("jsonlite", quietly = TRUE)
) {
  plot_footprint_sankey(footprints)
}
```

---

polities *Polities*

Description

Defines name/code correspondences for polities (political entities).

Usage

```
polities
```

Format

A tibble where each row corresponds to one polity. It contains the following columns: TODO: On polities Pull Request, coming soon

---

polities_cats *Polity categories and regional classifications*

Description

Reference table for countries and political entities (polities) with identifiers from multiple data sources and assignments to various regional groupings used in the literature and international databases.

Usage

```
polities_cats
```

Format

A tibble where each row corresponds to one polity (country or territory). It contains the following columns:

- `polity_code`: ISO 3166-1 alpha-3 country code used as the primary identifier (e.g., `"AFG"`, `"ALB"`).
- `polity_name`: Common country or territory name.
- `V1`: Internal row index from the source table.
- `code`: Numeric FAOSTAT country code.
- `iso3c`: ISO 3166-1 alpha-3 code (character; may duplicate `polity_code` or differ for aggregates).
- `FAOSTAT_name`: Country name as used in FAOSTAT.
- `EU27`: Logical flag; `TRUE` if the polity is a member of the EU27.
- `name`: Country name used in other external databases.
- `eia`: Country name or code used by the US Energy Information Administration (EIA).
- `iea`: Country identifier used by the International Energy Agency (IEA).
- `water_code`: Numeric code used in water statistics datasets.
- `water_area`: Country/area name used in water statistics.
- `baci`: Numeric BACI trade database country code.
- `fish`: Numeric code used in fisheries datasets.
- `region_code`: Numeric regional grouping code.
- `cbs`: Logical flag; `TRUE` if the polity is included in the CBS dataset.
- `fabio_code`: Numeric country code used in the FABIO database.
- `ADB_Region`: Asian Development Bank regional classification.

- region: General world region (e.g., "South Asia", "Eastern Europe").
- uISO3c: Numeric Unicode / UN M49 country code.
- Lassaletta: Country grouping used in Lassaletta et al. nitrogen flow studies.
- region_krausmann: Regional grouping from Krausmann et al. biomass flow accounting.
- region_HANPP: Regional grouping used in human appropriation of net primary production (HANPP) studies.
- region_krausmann2: Alternative Krausmann regional grouping.
- region_UN_sub: UN sub-regional classification (M49 sub-region).
- region_UN: UN macro-regional classification (M49 region).
- region_ILO1: ILO primary regional grouping.
- region_ILO2: ILO secondary regional grouping.
- region_ILO3: ILO tertiary regional grouping.
- region_IEA: IEA regional grouping.
- region_IPCC: IPCC regional grouping used in climate assessments.
- region_labour: Labour-focused regional grouping.
- region_labour_agg: Aggregated labour-focused regional grouping.
- region_labour_mech: Labour mechanisation regional grouping.
- region_test: Experimental/test regional grouping (may be incomplete).

Note

Five trailing columns containing only Excel #REF! errors in the source CSV are dropped at load time and are not part of this dataset.

Source

Compiled from FAOSTAT, UN M49, ILO, IEA, and other international statistical sources.

Examples

```
head(polities_cats)
```

---

prepare_livestock_emissions

Prepare production data for livestock emission calculations.

Description

Bridge between build_primary_production() output and the livestock emission functions. Maps species codes to names, converts area codes to ISO3, extracts milk and meat yields, and optionally expands herds into GLEAM cohorts.

Any extra columns present in the input (e.g., weight, diet_quality, fat_percent) are preserved and flow through to the emission functions automatically.

Usage

```
prepare_livestock_emissions(data, expand_cohorts = FALSE, system_shares = NULL)
```

Arguments

data	A tibble from build_primary_production() with columns item_cbs_code, unit, value, and optionally year, area_code, live_anim_code.
expand_cohorts	Logical. If TRUE, distributes herds across GLEAM cohorts and production systems via calculate_cohorts_systems(). Default FALSE.
system_shares	Optional dataframe with custom system shares. Passed to calculate_cohorts_systems().

Value

A tibble with columns species, heads, iso3 (if area_code present), and optionally milk_yield_kg_day, meat_yield_t_head, cohort columns, plus all extra columns from the input.

Examples

```
tibble::tibble(
  item_cbs_code = 961,
  unit = "heads",
  value = 5000,
  area_code = 79L
) |>
  prepare_livestock_emissions()
```

---

primary_double *Items with double-counting in production statistics*

Description

Identifies production items that appear both as primary crop products and as harvested-area items, requiring special treatment to avoid double-counting in production and biomass accounting.

Usage

```
primary_double
```

Format

A tibble where each row corresponds to one item pair with a double-counting relationship. It contains the following columns:

- Item_area: Name of the item as it appears in harvested-area statistics (e.g., "Seed cotton, unginned").
- item_prod: Name of the derived production item (e.g., "Cotton lint, ginned", "Cotton seed").

- item_prod_code: Numeric FAOSTAT production code of the derived item.
- Multi_type: Classification of the double-counting type:
  - "Primary": The area item is the primary crop; product is a direct output.
  - "Primary_area": Area is recorded under a primary aggregate crop name.
  - "Multi": Multiple products share the same harvested area.
  - "Multi_area": Multiple products share a recorded area aggregate.

Source

Derived from FAOSTAT production methodology documentation.

Examples

```
head(primary_double)
```

---

regional_mms_distribution

Regional MMS distribution.

Description

Fraction of manure managed in each MMS type by region and species.

Usage

```
regional_mms_distribution
```

Format

A tibble with region, species, mms_type, fraction.

Source

GLEAM 3.0 / FAO statistics.

Examples

```
regional_mms_distribution
```

regions_full *Full polity and region reference table*

Description

Extended reference table covering all polities and aggregate regions, including countries, territories, and statistical composites that appear in international databases but may lack standard ISO codes.

Usage

```
regions_full
```

Format

A tibble where each row corresponds to one polity or aggregate region. It contains the following columns (same definitions as `polities_cats`, minus the five trailing `0...36`–`0...40` artefact columns):

- `polity_code`: Primary polity identifier (ISO 3166-1 alpha-3 or `NA` for non-sovereign aggregates).
- `polity_name`: Polity name (`NA` for aggregates not matched to a standard polity).
- `V1`: Internal row index.
- `code`: Numeric FAOSTAT country/region code.
- `iso3c`: ISO 3166-1 alpha-3 code (`NA` for aggregates).
- `FAOSTAT_name`: Name used in FAOSTAT (may be `"#N/A"` for aggregates).
- `EU27`: Logical EU27 membership flag.
- `name`: Name used in external databases.
- `eia`: EIA country identifier.
- `iea`: IEA country identifier.
- `water_code`: Water statistics numeric code.
- `water_area`: Name used in water statistics.
- `baci`: BACI trade database country code.
- `fish`: Fisheries dataset numeric code.
- `region_code`: Numeric regional code.
- `cbs`: Logical CBS dataset membership flag.
- `fabio_code`: FABIO database numeric code.
- `ADB_Region`: Asian Development Bank region.
- `region`: General world region.
- `uISO3c`: UN M49 numeric code.
- `Lassaletta`: Lassaletta et al. nitrogen study grouping.
- `region_krausmann`: Krausmann regional grouping.

- region_HANPP: HANPP study regional grouping.
- region_krausmann2: Alternative Krausmann grouping.
- region_UN_sub: UN M49 sub-region.
- region_UN: UN M49 macro-region.
- region_ILO1: ILO primary region.
- region_ILO2: ILO secondary region.
- region_ILO3: ILO tertiary region.
- region_IEA: IEA region.
- region_IPCC: IPCC region.
- region_labour: Labour-focused region.
- region_labour_agg: Aggregated labour region.
- region_labour_mech: Labour mechanisation region.
- region_test: Experimental regional grouping.

Source

Compiled from FAOSTAT, UN M49, ILO, IEA, and other international statistical sources.

See Also

polities_cats for the subset restricted to sovereign countries.

Examples

```
head(regions_full)
```

---

run_spatialize *Run the gridded land-use spatialization pipeline*

Description

Wrapper around build_gridded_landuse() that resolves a named preset ("lpjml" or "whep") into a consistent bundle of input files, engine flags, and output paths. Use this to produce two comparable outputs from the same prepared parquet inputs: an LPJmL/LandInG-faithful run (for cell-by-cell comparison against LPJmL inputs) and the full WHEP run (all historical years, LUH2 type-aware allocation).

Presets can be combined with per-flag overrides to produce any intermediate configuration; the resolved configuration is written next to the outputs as run_metadata.yaml for traceability.

Usage

```
run_spatialize(
  preset = c("lpjml", "whep"),
  years = NULL,
  components = c("landuse", "livestock"),
  overrides = list(),
  paths = list()
)
```

Arguments

preset	One of "lpjml" or "whep". Selects a default bundle of engine flags and input choices. See *Presets*.
years	Integer vector of years to spatialize. If NULL, the preset default is used: for "lpjml" a 10-year benchmark sequence (seq(1850L, 2020L, by = 10L)), intersected with the years available in country_areas; for "whep" all years present in country_areas.
components	Character vector selecting which engines to run. Defaults to c("landuse", "livestock"). Pass a subset to run only one (e.g. "landuse"). Unknown entries raise an error.
overrides	Named list of flags that override the preset. Unknown keys raise an error. Recognised entries: • use_type_constraint (logical): enable/disable LUH2 type-aware allocation. • aggregate_to_cft (logical, default TRUE): write a CFT-aggregated parquet alongside the crop-level output. • max_iterations, expansion_threshold: forwarded to the landuse engine. • cft_target: one of "whep" (default for preset = "whep") or "lpjml" (default for preset = "lpjml"). Selects which column of cft_mapping.csv drives CFT aggregation: cft_name (granular 33-class WHEP taxonomy) or cft_lpjml (12 LPJmL crop CFTs + single others bucket).
paths	Named list of filesystem paths. Recognised entries: • l_files_dir (required): path to the L_files root. • input_dir: directory holding the prepared input parquets. If NULL, defaults to <l_files_dir>/whep/inputs. • out_dir: output directory. If NULL, defaults to <l_files_dir>/whep/spatialize/<preset> (suffixed with _custom when overrides is non-empty). Created if missing.

Value

Invisibly, a named list with preset, resolved config, years, out_dir, and output_paths.

Presets

`lpjml` LandInG-faithful configuration: no LUH2 type-aware allocation (`use_type_constraint = FALSE`) and a short default year sample suited to comparison against LPJmL inputs.

`whep` Full WHEP configuration: LUH2 type-aware allocation (`use_type_constraint = TRUE`) and the full historical year range present in `country_areas`.

Inputs read from `input_dir`

Landuse (`components` contains `"landuse"`):

- `country_areas.parquet`
- `crop_patterns.parquet`
- `gridded_cropland.parquet`
- `country_grid.parquet`
- `type_cropland.parquet` (required when `use_type_constraint = TRUE`).

Livestock (`components` contains `"livestock"`):

- `livestock_country_data.parquet`
- `gridded_pasture.parquet`
- `gridded_cropland.parquet`, `country_grid.parquet`
- `manure_pattern.parquet` (optional, enables manure-intensity weighting if present).
- `livestock_mapping.csv` from the installed package.

Outputs written to `out_dir`

- `gridded_landuse_crops.parquet` — crop-level output.
- `gridded_landuse.parquet` — CFT-aggregated output (when `aggregate_to_cft = TRUE`).
- `gridded_livestock_emissions.parquet` — gridded livestock stocks and emissions (when livestock component selected).
- `run_metadata.yaml` — resolved preset, components, flags, years, timestamp, and package version.

See Also

`build_gridded_landuse()`.

Examples

```
# Dispatch to the engine with a filtered year range (offline
# example; normally called against prepared parquet inputs).
country_areas <- tibble::tribble(
  ~year, ~area_code, ~item_prod_code, ~harvested_area_ha,
  1999L,         1L,             15L,                500,
  2000L,         1L,             15L,               1000
)
crop_patterns <- tibble::tribble(
```

```
  ~lon,  ~lat, ~item_prod_code, ~harvest_fraction,
   0.25, 50.25,             15L,               0.6,
   0.75, 50.25,             15L,               0.4
)
gridded_cropland <- tibble::tribble(
  ~lon,  ~lat,  ~year, ~cropland_ha,
   0.25, 50.25, 1999L,          800,
   0.75, 50.25, 1999L,          500,
   0.25, 50.25, 2000L,          800,
   0.75, 50.25, 2000L,          500
)
country_grid <- tibble::tribble(
  ~lon,  ~lat, ~area_code,
   0.25, 50.25,         1L,
   0.75, 50.25,         1L
)
build_gridded_landuse(
  country_areas, crop_patterns, gridded_cropland, country_grid,
  config = list(years = 2000L)
)
```

`smil_2001_synthetic_n_global` *Smil (2001) global synthetic nitrogen production, 1913-2000*

Description

Global synthetic-nitrogen production anchors from Smil (2001) "Enriching the Earth", Tables 5.2 and 5.3, cross-checked with Smil (2002) Ambio 31:126-131. Anchor years span 1913 (first commercial Haber-Bosch plant at BASF Oppau) to 2000. Used by `prepare_nitrogen_inputs()` to backcast country-level synthetic N for the pre-FAOSTAT period (years before 1961): the temporal shape is taken from this global series and downscaled to each country using its 1961-1965 share of global FAOSTAT synthetic N.

Pre-1913 values are treated as zero by the consumer and are not stored here.

Usage

```
smil_2001_synthetic_n_global
```

Format

A tibble with one row per anchor year:

- `year`: Integer anchor year (1913, 1920, 1925, ..., 2000).
- `global_kt_n`: Global synthetic-N production in kt N.

Source

Smil, V. (2001) *Enriching the Earth: Fritz Haber, Carl Bosch, and the Transformation of World Food Production*, MIT Press. Tables 5.2 and 5.3.

Examples

```
head(smil_2001_synthetic_n_global)
```

---

temperature_adjustment *Temperature adjustment factors for NEm.*

Description

Adjustment multipliers for net energy maintenance under cold stress, thermoneutral, and heat stress conditions.

Usage

```
temperature_adjustment
```

Format

A tibble with `temp_range`, `temp_min`, `temp_max`, `adjustment_factor`.

Source

NRC 2001; IPCC 2019.

Examples

```
temperature_adjustment
```

---

uncertainty_ranges *Uncertainty ranges for emission parameters.*

Description

Lower and upper multipliers for key emission parameters (Ym, MCF, Bo, EF_N2O, Nex).

Usage

```
uncertainty_ranges
```

Format

A tibble with `parameter`, `lower_mult`, `upper_mult`, `distribution`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10.

Examples

```
uncertainty_ranges
```

`whep_clear_cache` *Clear the build pipeline cache*

Description

Removes cached results from `build_primary_production()`, `build_commodity_balances()`, and `build_processing_coefs()` so that the next call rebuilds from scratch.

Usage

```
whep_clear_cache()
```

Value

Invisible `NULL`.

Examples

```
whep_clear_cache()
```

`whep_inputs` *External inputs*

Description

The information needed for accessing external datasets used as inputs in our modeling.

Usage

```
whep_inputs
```

Format

A tibble where each row corresponds to one external input dataset. It contains the following columns:

- `alias`: An internal name used to refer to this dataset, which is the expected name when trying to get the dataset with `whep_read_file()`.
- `board_url`: The public static URL where the data is found, following the concept of a *board* from the `pins` package, which is what we use for storing these input datasets.
- `version`: The specific version of the dataset, as defined by the `pins` package. The version is a string similar to `"20250714T123343Z-114b5"`. This version is the one used by default if no `version` is specified when calling `whep_read_file()`. If you want to use a different one, you can find the available versions of a file by using `whep_list_file_versions()`.

Source

Created by the package authors.

`whep_list_file_versions` *Input file versions*

Description

Lists all existing versions of an input file from `whep_inputs`.

Usage

```
whep_list_file_versions(file_alias)
```

Arguments

`file_alias`	Internal name of the requested file. You can find the possible values in the `whep_inputs` dataset.

Value

A tibble where each row is a version. For details about its format, see `pins::pin_versions()`.

Examples

```
whep_list_file_versions("read_example")
```

`whep_read_file` *Download, cache and read files*

Description

Used to fetch input files that are needed for the package's functions and that were built in external sources and are too large to include directly. This is a public function for transparency purposes, so that users can inspect the original inputs of this package that were not directly processed here.

If the requested file doesn't exist locally, it is downloaded from a public link and cached before reading it. This is all implemented using the `pins` package. It supports multiple file formats and file versioning.

Usage

```
whep_read_file(file_alias, type = "parquet", version = NULL)
```

Arguments

`file_alias`	Internal name of the requested file. You can find the possible values in the `alias` column of the `whep_inputs` dataset.
`type`	The extension of the file that must be read. Possible values: • `parquet`: This is the default value for code efficiency reasons. • `csv`: Mainly available for those who want a more human-readable option. If the `parquet` version is available, this is useless because this function already returns the dataset in an `R` object, so the origin is irrelevant, and `parquet` is read faster. Saving each file in both formats is for transparency and accessibility purposes, e.g., having to share the data with non-programmers who can easily import a CSV into a spreadsheet. You will most likely never have to set this option manually unless for some reason a file could not be supplied in e.g. `parquet` format but was in another one.
`version`	The version of the file that must be read. Possible values: • `NULL`: This is the default value. A frozen version is chosen to make the code reproducible. Each release will have its own frozen versions. The version is the string that can be found in `whep_inputs` in the `version` column. • `"latest"`: This overrides the frozen version and instead fetches the latest one that is available. This might or might not match the frozen version. • Other: A specific version can also be used. For more details read the `version` column information from `whep_inputs`.

Value

A tibble with the dataset. Some information about each dataset can be found in the code where it's used as input for further processing.

Examples

```
whep_read_file("read_example")
whep_read_file("read_example", type = "parquet", version = "latest")
whep_read_file(
  "read_example",
  type = "csv",
  version = "20250721T152646Z-ce61b"
)
```

Index