Package 'rfcipCalibrateExtended'

October 18, 2025

Type Package

Title Tools for FCIP Yield, Revenue, and Preference Calibration

Version 0.0.0.9000

Author Francis Tsiboe [aut, cre] (<https://orcid.org/0000-0001-5984-1072>)

Maintainer Francis Tsiboe <ftsiboe@hotmail.com>

Contributor -

Reviewer -

Creator Francis Tsiboe

Description Description: Provides tools for calibrating yield and preference models for farm-level Federal Crop Insurance Program (FCIP) experience.
It is designed for use alongside the 'rfcip', 'rfcipCalcPass', 'rfcipDemand', and 'rfcipCalibrate' packages.

License GPL-3 + file LICENSE

URL https://github.com/you/rfcipCalibrateExtended

BugReports https://github.com/you/rfcipCalibrateExtended/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Depends R (>= 4.1.0)

Imports data.table, crayon, DEoptim, devtools, future, future.apply, purrr, readr, rfcip

Suggests rfcipCalcPass, rfcipCalibrate, knitr, rmarkdown, numDeriv, dplyr, testthat (>= 3.0.0)

Remotes github::dylan-turner25/rfcip

LazyData true

Cite-us If you find it useful, please consider staring the repository and citing the following studies
- Tsiboe, F. and Turner, D. (2025). ``Incorporating buy-up price loss coverage into the United States farm safety net." Applied Economic Perspectives and Policy.
- Tsiboe, F., et al. (2025). ``Risk reduction impacts of crop insurance in the United States." Applied Economic Perspectives and Policy.
- Gaku, S. and Tsiboe, F. (2024). Evaluation of alternative farm safety net program combination strategies. Agricultural Finance Review.

Contents

`calibrate_plan_residual_factor`
Plan Residual Factor Calibration Workhorse

Description

For a given set of yield inputs, computes the plan-specific residual factors (relative premium rates) for Yield Protection (YP), Revenue Protection (RP), and RP-HPE policies by running the FCIP calculator at unity acres, shares, and price election. These factors are normalized so that the YP factor is 1.

Usage

```
calibrate_plan_residual_factor(
  farmdata,
  farm_identifiers = NULL,
  control = rfcipCalibrateExtended_control()
)
```

Arguments

`farmdata`	A data.frame or data.table of observed or/and calibrated producer records. Must contain at least: `producer_id`: character or numeric ID for each farm/agent row. Defaults to the row identifier `commodity_year`: commodity year, and all columns required by `rfcipCalcPass::fcip_calculator()`
`farm_identifiers`	column name(s) that uniquely identify each record/farm/producer. If `NULL`, a "producer_id" column is created.
`control`	List. A list of control parameters, typically created using `rfcipCalcPass::rfcipCalibrate_contro` to manage internal behavior and data sourcing.

Details

1. Sets `planted_acres`, `insured_share`, and `price_election` to 1 for pure rate calibration.
2. Copies the input three times, stamping `insurance_plan_code` = 1, 2, 3 respectively, and runs `rfcipCalcPass::fcip_calculator()`, returning only `producer_id` and `premium_rate`.
3. Renames each `premium_rate` to its plan residual factor.
4. Merges the three tables back together by `producer_id`.
5. Divides each RP and RP-HPE factor by the YP factor so that `plan_residual_factor_yp` = 1, and the others are scaled accordingly.

Value

A data.table with one row per `producer_id` and columns: `commodity_year`: commodity year `producer_id`: the row identifier `plan_residual_factor_yp`: normalized residual factor for YP (always 1) `plan_residual_factor_rp`: RP factor relative to YP `plan_residual_factor_rphpe`: RP-HPE factor relative to YP

`calibrate_preferences_extended` *Calibrate Agent Preferences via Merit Specification - full version*

Description

For each agent (producer), optimizes a merit (`utility`) function parameters to match observed or counterfactual choices over a menu of insurance options.

Usage

```
calibrate_preferences_extended(
  agent_data,
  merit_assumptions = merit_assumption_control(),
  control = rfcipCalibrateExtended_control()
)
```

Arguments

`agent_data`	A `data.table` or data frame where each row is an agent and includes list-columns: `menu_mu` List of expected returns for each menu option. `menu_<risk_measure>` List of risk measures matching `risk_measure`. `no_insurance` Logical list indicating the `no insurance` option. `observed` Logical list indicating the observed choice option. `menu_covlevl` List of coverage levels for each option. `menu_plan` List of plan identifiers for each option. `merit_function` Must also include a column `producer_id` for grouping.
`merit_assumptions`	a list of assumptions for an Agent's Merit Function; see `merit_assumption_control()`.
`control`	a list of control parameters; see `rfcipCalibrateExtended_control()`. `DEoptim_itermax`, `DEoptim_strategy`, `DEoptim_trace` are passed to `DEoptim::DEoptim.control()`

Value

A `data.table` with one row per `producer_id`, containing:

`preference_parameters` Estimated parameters (bestmem) from `DEoptim`.

`revealed_percentile_rank` Percentile rank of the revealed menu option utility.

`no_insurance_percentile_rank` Percentile rank of the no insurance/coverage menu option utility.

`revealed_merit_value` Utility value of the revealed menu option.

`revealed_marginal_merit_return` Marginal utility w.r.t.\ return at the revealed option.

`revealed_marginal_merit_risk` Marginal utility w.r.t.\ risk at the revealed option.

`revealed_merit_response_return` Elasticity of utility w.r.t.\ return at the revealed option.

`revealed_merit_response_risk` Elasticity of utility w.r.t.\ risk at the revealed option.

`mrs` Marginal rate of substitution (|du_ret/du_risk|).

`agent_type` Categorical label: profit vs.\ risk preference combination.

`revealed_budget` revealed total out-of pocket cost

`revealed_acres` revealed acres reported

`free_acres_cost` = (revealed_budget/revealed_acres)*control$free_acres_factor

`div_pct_rank_max_return` The difference in merit percentile rank between the choice with the maximum return and the revealed choice.

`div_pct_rank_min_risk` The difference in merit percentile rank between the choice with the minimum risk and the revealed choice.

`div_pct_rank_max_liability` The difference in merit percentile rank between the choice with the maximum liability and the revealed choice.

`div_pct_rank_max_subsidy` The difference in merit percentile rank between the choice with the maximum subsidy and the revealed choice.

`div_pct_rank_min_premium` The difference in merit percentile rank between the choice with the minimum premium and the revealed choice.

`div_pct_rank_min_paid` The difference in merit percentile rank between the choice with the minimum paid cost and the revealed choice.

`functional_form` Merit functional form calibrated

`menu_risk_measure` Risk measure.

`expansion_point` expansion point for standardization.

`visible_menu` Fraction of how many menu options are visible.

`visible_coverage` Coverage level held fix.

`visible_plan` Insurance plan level held fix.

`agent_assumption` One of `"profit_max_risk_min"`, `"risk_min"`, `"profit_max"`, or `"free"`.

`mrs_upper_limit` Upper bound on marginal rate of substitution penalty.

`parameter_limit` Absolute limit for each parameter in optimization.

See Also

Other Producer Preference Calibration: `merit_function_specifications`, `merit_objective_function_extended()`, `merit_specification_catalog()`, `normalize_and_filter_menu()`

`farmer_act_of_2024_premium_subsidy_adj` *Apply Farmer Act of 2024 premium-subsidy adjustments*

Description

Revises the `premium_subsidy_percent` column of a subsidy-schedule `data.table` to reflect policy changes contained in the draft *Farmer Act of 2024*.

Usage

```
farmer_act_of_2024_premium_subsidy_adj(current_schedule)
```

Arguments

`current_schedule`
: A **data.table** with (at minimum) the columns `insurance_plan_code`, `unit_structure_code`, `coverage_level_percent`, `coverage_type_code`, and `premium_subsidy_percent`. The table is **updated by reference**.

Details

The adjustments are applied in the following order:

1. **Supplemental Coverage Option (SCO)** products (`insurance_plan_code` 31 - 33) :subsidy raised to 80 %.
2. Enterprise/Whole-Farm units (`EU`, `WU`) with 80 % coverage on plan codes 1-3 or 90 :subsidy raised to 77 %.
3. Same units with 85 % coverage :subsidy raised to 68 %.
4. CAT coverage (`coverage_type_code == "C"`) is fully subsidised (100 %).
5. APH plan (code 4) at 65 % coverage is also fully subsidised.
6. *Producer-base rule:* For any buy-up product the subsidy may not exceed 80 %; we enforce that cap last.

Value

Invisibly returns `current_schedule` with its `premium_subsidy_percent` column updated.

See Also

Other Producer Premium Subsidy Policy: `obbb_act_premium_subsidy_adj()`

Examples

```
## Not run:
library(data.table)
sch <- data.table(
  insurance_plan_code = c(31, 2, 2),
  unit_structure_code = c("OU", "EU", "WU"),
  coverage_level_percent = c(0.70, 0.80, 0.85),
  coverage_type_code = c("B", "B", "B"),
```

```
  premium_subsidy_percent = c(0.65, 0.48, 0.38)
)
farmer_act_of_2024_premium_subsidy_adj(sch)
sch[]

## End(Not run)
```

`get_official_baseline_outcome` *Get Official Baseline Outcomes*

Description

Retrieve and process the Standard Operating Baseline (SOB) data for a given year and crop, computing county-level sums for key outcome measures and returning a long-format `data.table`.

Usage

```
get_official_baseline_outcome(year = 2011, crop = NULL, insurance_plan = NULL)
```

Arguments

`year`	Integer. Commodity year to calibrate (default `2011`).
`crop`	Character or integer vector. Crop name(s) or code(s). To list all valid crops and codes, use `rfcip::get_crop_codes()`.
`insurance_plan`	can be either a character string indicating an insurance plan (ex: `yp` and `yield protection` are both valid) or a numeric value indicating the insurance plan code (i.e. `1`). Inputting a vector with multiple values will return data for multiple insurance plans. To get a data frame containing all the available insurance plans and insurance plan codes use `rfcip::get_insurance_plan_codes()`.

Value

A `data.table` in long format, with one row per `state_code`/`county_code`/`commodity_code`/`outcome` combination:

`state_code` FIPS code of the state (numeric).

`county_code` FIPS code of the county (numeric).

`commodity_code` Numeric code of the crop/commodity.

`outcome` One of `"insured_acres"`, `"liability_amount"`, `"total_premium_amount"`, `"subsidy_amount"`, or `"indemnity_amount"`.

`official_value` County-level average of the outcome.

See Also

Other FCIP Demand Predictors: `predict_final_summary_of_business_crop()`, `predict_initial_summary_of_bu`

Examples

```
## Not run:
# Fetch baseline outcomes for year 2020, crop = "corn"
df <- get_official_baseline_outcome(2020, crop = "corn")
head(df)

## End(Not run)
```

`initialize_simulator` *Initialize the Simulation Environment*

Description

Downloads and caches RMA Actuarial Data Master (ADM) tables for one or more crop years, sets up a calibration directory structure, and then invokes `calibrate_simulator()` for each year/geography to produce and save all simulator inputs and calibration outputs.

Usage

```
initialize_simulator(
  years = 2011:as.numeric(format(Sys.Date(), "%Y")),
  state = NULL,
  clear_rfcip_cache = FALSE,
  clear_sob_cache = TRUE,
  project_name = NULL,
  merit_catalog_index = 1,
  available_cores = 1,
  control = rfcipCalibrateExtended_control()
)
```

Arguments

`years`	Numeric vector. Crop years to process (e.g. `2022` or `c(2022, 2023, 2024)`). Defaults to `2011` through the current year.
`state`	Character or integer. State abbreviation or full name, or FIPS code.
`clear_rfcip_cache`	Logical. If `TRUE`, calls `rmaADM::clear_rmaADM_cache()` before calibration.
`clear_sob_cache`	Logical. If `TRUE`, clears the State-of-Base cache (placeholder-implement as needed).
`project_name`	?????
`merit_catalog_index`	Integer. Row index in `merit_specification_catalog` to select for `rfcipCalibrate::calibrate_` (default 1).
`available_cores`	set number of available cores on current machine to use for parallel computing; defaults to 1. Use `future::availableCores()` to get number of available cores on current machine
`control`	a list of control parameters; see `rfcipCalibrateExtended_control`.

Value

Invisibly returns file and directory inventory for `dir_project`

Examples

```
## Not run:
initialize_simulator(
  years            = c(2022, 2023),
  crop             = c("corn", "soybean"),
  state            = "IA",
  county           = "Polk",
  clear_rfcip_cache  = TRUE,
  calibration_name = "nd_run"
)

## End(Not run)
```

`install_from_private_repo`

Conditionally Install a Private GitHub Package in CI

Description

This helper function installs a private R package from GitHub using a personal access token (PAT) during GitHub Actions runs. It checks if the package is installed and, if not, installs it using `devtools::install_github()` with authentication.

Usage

```
install_from_private_repo(user_name, package_name, secret_name)
```

Arguments

`user_name`	Character. The GitHub username or organization that owns the repository.
`package_name`	Character. The name of the package and repository.
`secret_name`	Character. The name of the environment variable holding the GitHub PAT (excluding the `"github_pat_"` prefix).

Details

This function runs only when the environment variable `GITHUB_ACTIONS` is `"true"`, which indicates it is running in a GitHub Actions workflow. It assumes that the provided PAT is stored in an environment variable named `secret_name`, and the full token is formed as `paste0("github_pat_", Sys.getenv(secret_name))`.

If `devtools` is not available, it will be installed from CRAN first. If the target package is already loaded, it will be detached before reinstallation.

Value

NULL. Called for its side effect of installing a package.

Note

This function is intended for use in CI environments and should not be called during package loading. Ensure the secret specified by `secret_name` is configured in your GitHub repository settings.

`merit_assumption_control` *Create a Control List of Assumptions for an Agent's Merit Function*

Description

Constructs a named list of user-defined assumptions controlling how the agent's merit or objective function is evaluated. This function is typically used as an input to `merit_function()` to specify behavioral assumptions, visibility constraints, and parameter limits governing optimization or model calibration.

Usage

```
merit_assumption_control(
  functional_form = merit_function_specifications$translog$formula,
  risk_measure = "vr",
  expansion_point = "no insurance",
  visible_menu = 1,
  visible_coverage = 0,
  visible_plan = 0,
  agent_assumption = "free",
  mrs_upper_limit = 10,
  parameter_limit = 100
)
```

Arguments

`functional_form`
: Character string specifying the functional form to be passed to `merit_function`. Defaults to the translog specification in `merit_function_specifications$translog$formula`.

`risk_measure`
: Character string indicating the risk measure to be used (e.g., `"vr"` for variance reduction or `"sd"` for standard deviation).

`expansion_point`
: Character string specifying the point around which to linearize or expand the merit function. Must be one of `"none"`, `"no insurance"`, `"observed choice"`, or `"menu mean"`.

`visible_menu`
: Numeric (integer or fraction) controlling how many menu options are visible to the agent when forming expectations or making choices.

`visible_coverage`
: Numeric vector specifying which coverage levels are included in the visible menu. Use `0` to include all.

`visible_plan`
: Numeric vector specifying which plan identifiers are included in the visible menu. Use `0` to include all.

`agent_assumption`
: Character string defining the agent's behavioral assumption. Must be one of `"profit_max_risk_min"`, `"risk_min"`, `"profit_max"`, or `"free"`.

`mrs_upper_limit`
: Numeric upper bound on the marginal rate of substitution (MRS) penalty term applied during optimization.

`parameter_limit`
: Numeric absolute limit on each parameter in the optimization routine.

Value

A named list containing all control parameters for the agent's merit function.

See Also

Other helpers: `rfcipCalibrateExtended_control()`

`merit_function_specifications`
Candidate Utility-Function Specifications

Description

`merit_function_specifications` is a helper **data object** that gathers widely used (and research-frontier) two-argument utility functions for empirical work in finance and applied micro-economics. Each entry is itself a named list with:

Usage

```
merit_function_specifications
```

Format

A named `list` of length 17. Each element is itself a two-component list with objects `formula` and `start_value`.

Details

- `formula`- a one-sided formula object (right-hand side only) that can be supplied directly to `nls()`, `optim()`, or similar optimisers, with arguments `ret` (expected return) and `risk` (a risk measure such as variance or probability of loss).
- `start_value`- a named numeric vector of reasonable starting values for the parameters $b0, b1, \ldots$. All parameters are consistently prefixed **b** for easy programmatic handling.

Contents:

Name in list	Functional family	Key citation
`translog`	Transcendental-logarithmic	Christensen, Jorgenson & Lau (1975)
`cobb_douglas`	Log-linear / Cobb-Douglas	Cobb & Douglas (1928)
`quadratic`	Mean-variance (quadratic)	Markowitz (1952)
`ces`	Constant-elasticity (CES)	Arrow et al. (1961)
`crra`	Power / CRRA	Pratt (1964)
`cara`	Exponential / CARA	Pratt (1964)
`hara`	Hyperbolic ARA	Cox, Ingersoll & Ross (1985)

`expo_power`	Expo-Power (flexible RA)	Saha (1993)
`stone_geary`	Stone-Geary	Stone (1954); Geary (1950)
`prospect`	Prospect-theory value	Kahneman & Tversky (1979)
`ez_static`	Epstein-Zin (static CES)	Epstein & Zin (1989)
`cubic`	Cubic polynomial	Barnett & Serletis (2008)
`habit_external`	Campbell-Cochrane habit	Campbell & Cochrane (1999)
`smooth_ambiguity`	Klibanoff-Marinacci-Mukerji ambiguity	Klibanoff et al. (2005)
`rank_dependent`	Prelec probability weight	Prelec (1998)
`disappointment`	Disappointment aversion	Gul (1991)
`entropic`	Entropic / risk-sensitive	Hansen & Sargent (1995)

Source

https://github.com/your-org/your-package/data-raw/

References

- Barnett, W. A., & Serletis, A. (2008). *Martingales, Nonlinear Dynamics, and Chaos in Economics*.
- Campbell, J. Y., & Cochrane, J. H. (1999). By Force of Habit. *Journal of Political Economy, 107*(2), 205-251. https://doi.org/10.1086/250059
- Christensen, L. R., Jorgenson, D. W., & Lau, L. J. (1975). Transcendental Logarithmic Utility Functions. *American Economic Review, 65*(3), 367-383.
- Cobb, C. W., & Douglas, P. H. (1928). A Theory of Production. *American Economic Review, 18*(Suppl.), 139-165.
- Cox, J. C., Ingersoll, J. E., & Ross, S. A. (1985). An Intertemporal General Equilibrium Model of Asset Prices. *Econometrica, 53*(2), 363-384.
- Epstein, L. G., & Zin, S. E. (1989). Substitution, Risk Aversion, and the Temporal Behaviour of Consumption and Asset Returns. *Econometrica, 57*(4), 937-969.
- Geary, R. C. (1950). A Note on A Constant-Utility Index of the Cost of Living. *Review of Economic Studies, 18*(1), 65-66.
- Gul, F. (1991). A Theory of Disappointment Aversion. *Econometrica, 59*(3), 667-686.
- Hansen, L. P., & Sargent, T. J. (1995). Discounted Linear Exponential Quadratic Gaussian Control. *IEEE T-AC, 40*(5), 968-971.
- Kahneman, D., & Tversky, A. (1979). Prospect Theory: An Analysis of Decision under Risk. *Econometrica, 47*(2), 263-291.
- Klibanoff, P., Marinacci, M., & Mukerji, S. (2005). A Smooth Model of Decision Making under Ambiguity. *Econometrica, 73*(6), 1849-1892.
- Markowitz, H. (1952). Portfolio Selection. *Journal of Finance, 7*(1), 77-91.
- Prelec, D. (1998). The Probability Weighting Function. *Econometrica, 66*(3), 497-527.
- Pratt, J. W. (1964). Risk Aversion in the Small and in the Large. *Econometrica, 32*(1/2), 122-136.
- Saha, A. (1993). Expo-Power Utility: A Flexible Form for Absolute and Relative Risk Aversion. *American Journal of Agricultural Economics, 75*(4), 905-913.
- Stone, R. (1954). Linear Expenditure Systems and Demand Analysis. *Economic Journal, 64*(255), 511-527.
- Arrow, K. J., Chenery, H. B., Minhas, B. S., & Solow, R. M. (1961). Capital-Labour Substitution and Economic Efficiency. *Review of Economics and Statistics, 43*(3), 225-250.

See Also

Other Producer Preference Calibration: `calibrate_preferences_extended()`, `merit_objective_function_extend`
`merit_specification_catalog()`, `normalize_and_filter_menu()`

`merit_objective_function_extended` *Merit-based DEoptim Objective Function - full version*

Description

Constructs the scalar loss that DEoptim will minimize, defined as the sum of squared first-order-condition (FOC) deviations plus any MRS exceedance penalties.

Usage

```
merit_objective_function_extended(
  parameter_values,
  ret,
  risk,
  functional_form,
  agent_assumption,
  mrs_upper_limit
)
```

Arguments

`parameter_values`
: Numeric vector. Candidate parameter values for the merit function.

`ret`
: Numeric vector. Normalized returns for each menu option.

`risk`
: Numeric vector. Normalized risk measures matching the chosen 'risk_measure'. Must match the length of `ret`.

`functional_form`
: A one-sided formula object (e.g., `~ b0 + b1 * ret + b2 * risk + ...`) representing the merit function to analyze.

`agent_assumption`
: Character(1). One of:

  'profit_max_risk_min' Enforce dU/dret > 0 and dU/drisk < 0.

  'risk_min' Enforce dU/dret = 0 and dU/drisk < 0.

  'profit_max' Enforce dU/dret > 0 and dU/drisk = 0.

  'free' No sign constraints on marginal merits

`mrs_upper_limit`
: Numeric(1). Threshold for |dU/dret / dU/drisk| above which a quadratic penalty is applied.

Details

This objective has five components:

- Ranking the first-option utility as highest.
- Ranking the last-option utility as lowest.
- Sign/equality constraints on marginal utility of return.
- Sign/equality constraints on marginal utility of risk.
- Quadratic penalty if the implied marginal rate of substitution exceeds mrs_upper_limit.

It is intended to be passed directly as the 'fn' argument to `DEoptim::DEoptim()`, with bounds and control parameters set by the caller.

Value

Numeric(1). Sum of squared FOC residuals and any MRS-penalty terms; used as the DEoptim objective.

See Also

Other Producer Preference Calibration: `calibrate_preferences_extended()`, `merit_function_specifications`, `merit_specification_catalog()`, `normalize_and_filter_menu()`

Other Producer Preference Calibration: `calibrate_preferences_extended()`, `merit_function_specifications`, `merit_specification_catalog()`, `normalize_and_filter_menu()`

`merit_specification_catalog` *Catalogue of Merit-Function Specifications*

Description

Build a lookup table of all supported combinations of risk measures, functional forms, agent assumptions, visibility settings, and optimization bounds that can be passed to `rfcipCalibrate::calibrate_preference`

Usage

```
merit_specification_catalog()
```

Value

A `data.table` with one row per specification and the following columns:

`catalogue_id` Integer ID (row number).

`functional_form` Character name of the functional form passed to `rfcipCalibrate::merit_function`.

`risk_measure` Character name of the risk measure (e.g. `"vr"`, `"sd"`, `"cv"`, etc.).

`expansion_point` Character; one of `"no insurance"`, `"observed choice"`, `"menu mean"`, or `"none"`.

`visible_menu` Numeric: fraction (0-1) or integer controlling how many menu options are visible.

`visible_coverage` Numeric vector of coverage levels to include (0 = include all).

`visible_plan` Integer vector of plan identifiers to include (0 = include all).

`agent_assumption` Character; one of `"profit_max_risk_min"`, `"risk_min"`, `"profit_max"`, or `"free"`.

`mrs_upper_limit` Numeric; upper bound on the marginal rate of substitution penalty.

`parameter_limit` Numeric; absolute bound on each optimization parameter.

`num_parameters` Integer; number of parameters in the `alpha` vector for `rfcipCalibrate::merit_function`.

See Also

Other Producer Preference Calibration: `calibrate_preferences_extended()`, `merit_function_specifications`, `merit_objective_function_extended()`, `normalize_and_filter_menu()`

`normalize_and_filter_menu` *Normalize and Filter Insurance Menu Returns & Risks*

Description

Given raw vectors of expected returns and risks for a menu of insurance options, this function will

1. Re-scale both returns and risks relative to a chosen expansion point, and
2. Optionally filter out menu items by fraction, coverage level, or plan.

Usage

```
normalize_and_filter_menu(
  ret,
  risk,
  no_insurance,
  observed,
  menu_plan,
  menu_covlevl,
  expansion_point = c("no insurance", "menu mean", "observed choice", "none"),
  visible_menu = 1,
  visible_coverage = 0,
  visible_plan = 0
)
```

Arguments

`ret`	Numeric vector of expected returns for each menu option.
`risk`	Numeric vector of risk measures for each menu option (must align with `ret`).
`no_insurance`	Logical (or length-1 logical list) marking which entry is the `no insurance` option.
`observed`	Logical (or length-1 logical list) marking which entry is the observed choice. **Required** if `expansion_point = "observed choice"`.
`menu_plan`	Vector or list of plan identifiers for each menu option (used if `visible_plan != 0`).
`menu_covlevl`	Vector or list of coverage-level codes for each menu option (used if `visible_coverage != 0`).

`expansion_point`
: Character; one of
  - `none` - no normalization
  - `no insurance` - normalize so that the no-insurance option has value 1
  - `observed choice` - normalize so that the observed choice has value 1
  - `menu mean` - normalize so the mean of all options is 1

`visible_menu`
: Numeric fraction in (0,1] or integer >0. If not 1, randomly samples that fraction of menu rows (always keeping the first and last items).

`visible_coverage`
: Numeric scalar of coverage code to keep (0 = all, 1 = hold coverage fixed at that of the observed policy). (always keep the last item)

`visible_plan`
: Numeric scalar of plan identifier to keep (0 = all , 1 = hold plan fixed at that of the observed policy). (always keep the last item)

Details

This function factors out the normalization-and-filtering logic from `calibrate_preferences()`. If you need to change how `"expansion points"` or menu-visibility work, update here and it will flow through everywhere.

Value

A named list with components

- `ret` : the normalized (and possibly subsetted) return vector
- `risk`: the normalized (and possibly subsetted) risk vector

See Also

Other Producer Preference Calibration: `calibrate_preferences_extended()`, `merit_function_specifications`, `merit_objective_function_extended()`, `merit_specification_catalog()`

`obbb_act_premium_subsidy_adj` *Apply OBBB Act premium-subsidy adjustments*

Description

Implements the premium subsidy schedule proposed in the *One Big Beautiful Bill Act* (H.R. 1, 119th Congress).

Usage

```
obbb_act_premium_subsidy_adj(current_schedule)
```

Arguments

`current_schedule`
: A **data.table** with (at minimum) the columns `insurance_plan_code`, `unit_structure_code`, `coverage_level_percent`, `coverage_type_code`, and `premium_subsidy_percent`. The table is **updated by reference**.

Details

Adjustments are applied in six blocks; within each block the table is filtered by unit structure, insurance plan, and coverage level, then subsidies are increased. For Enterprise (EU) and Whole-Farm (WU) units the statutory increase is scaled relative to the existing differential and capped at 80 %.

Steps (all limited to plan codes 1-3 and 90 unless noted):

1. 55-60 % coverage to 69 % subsidy (EU/WU scaled, then capped at 80 %).
2. 65-70 % coverage to 64 % subsidy (EU/WU scaled + cap).
3. 75 % coverage to 60 % subsidy (EU/WU scaled + cap).
4. 80 % coverage to 51 % subsidy (EU/WU scaled + cap).
5. 85 % coverage to 41 % subsidy (EU/WU scaled + cap).
6. SCO products (plan codes 31-33) to 80 % subsidy.

Value

Invisibly returns the updated `current_schedule`.

See Also

Other Producer Premium Subsidy Policy: `farmer_act_of_2024_premium_subsidy_adj()`

Examples

```
## Not run:
obbb_act_premium_subsidy_adj(sch)  # uses 'sch' from previous example
sch[]

## End(Not run)
```

`predict_final_summary_of_business_crop` *Scale* alternate *Summary-of-Business predictions to the official baseline*

Description

`predict_final_summary_of_business_crop()` reconciles **baseline** and **alternate** model predictions with the officially published Summary-of-Business (SoB) totals for a *single* crop year.

The procedure

1. pulls the official county-commodity totals via `get_official_baseline_outcome()`,
2. rescales *alternate* predictions so that their **county-level** sums equal the official baseline while preserving each record share of the original *alternate* total,
3. aggregates to the actuarial granularity used in crop-insurance financials,
4. applies a loss-cost ratio (LCR) to derive indemnity dollars,
5. backs out premium and subsidy dollars from the simulated premium/liability and subsidy/premium ratios, **and**
6. estimates *uninsured* acres from zero-coverage rows, scales them consistently with the insured acres, and appends them as a `coverage_level_percent == 0` record so the output carries a complete `reported acres` total (insured + uninsured).

Usage

```
predict_final_summary_of_business_crop(
  initial_baseline,
  initial_alternate,
  indemnitor = NULL
)
```

Arguments

`initial_baseline`
: A `data.table` (or a plain `data.frame`, which will be converted) produced by `predict_initial_summary_of_business_crop()`. **Required columns**

  `state_code`, `county_code`, `commodity_code`, `commodity_year`, `insurance_plan_code`, `coverage_type_code`, `coverage_level_percent`, `insured_acres`, `liability_amount`, `total_premium_amount`, `subsidy_amount`

`initial_alternate`
: Same structure and column requirements as `initial_baseline`, but containing the *alternate-scenario* predictions.

`indemnitor`
: Character scalar.

  - `"official_loss_cost_ratio"` (default or `NULL`) - use the county-level LCR supplied by `get_official_baseline_outcome()`.
  - Any other value triggers an error (future hooks for user-supplied LCRs).

Details

Scaling mathematics:

For each *county, commodity, outcome* (`insured_acres`, `liability_amount`):

$$\text{alternate\_sum\_adj} = \frac{\text{official\_value}}{\text{baseline\_sum}} \times \text{alternate\_sum}$$

Each record value is then prorated:

$$\text{alternate\_adj} = \frac{\text{alternate}}{\text{alternate\_sum}} \times \text{alternate\_sum\_adj}$$

Uninsured-acre logic (step 6):

Rows with `coverage_level_percent == 0` represent *uninsured* acreage. Their share of total (insured + uninsured) acres is first calculated at the pool level; that share is then applied to the *scaled* insured acres to arrive at a consistent estimate of uninsured acres, which is appended as a separate record (all dollar columns zeroed).

Value

A `data.table` with **one row per unique combination** of *FCIP-insurance pool* (`FCIP_INSURANCE_POOL`), *election* (`FCIP_INSURANCE_ELECTION`), and (for insured acres) the underwriting keys `insurance_plan_code`, `coverage_type_code`, `coverage_level_percent`. Returned columns

column	units
`insured_acres`	acres
`liability_amount`	USD

total_premium_amount	USD
subsidy_amount	USD
indemnity_amount	USD
reported_acres	acres

See Also

Other FCIP Demand Predictors: get_official_baseline_outcome(), predict_initial_summary_of_business_cr

---

predict_initial_summary_of_business_crop *Predict Initial un-adjusted (to baseline scaling) Summary of Business for Crop Producers*

Description

This function predicts producers' choice of crop insurance policies based on specified preferences, then aggregates then into a summary of business by type, practice, and unit structure for insured acres, liability, premiums, subsidies, and indemnities for a given prediction year.

Usage

```
predict_initial_summary_of_business_crop(
  prediction_year,
  agent_data,
  preference,
  protection_level = c("individual", "group", "none"),
  farm_identifiers = "producer_id",
  calculate_indemnity = TRUE,
  subsidy_calculator = NULL,
  control = rfcipCalibrateExtended_control()
)
```

Arguments

prediction_year	Integer specifying the year for prediction.
agent_data	A data.table of farm-level input. contain at least all columns required by fcip_menu_dispatcher()
preference	A data.table of preference parameters and menu indices per producer for selection generated by calibrate_preferences()
protection_level	Character vector of allowed protection levels: "individual", "group", or "none". Defaults to all three. see fcip_menu_dispatcher()
farm_identifiers	column name(s) that uniquely identify each record/farm/producer. If NULL, a "producer_id" column is created.
calculate_indemnity	Logical; if TRUE, compute indemnity amounts.

`subsidy_calculator`
: Function or `NULL`. If `NULL`, it defaults to `fcip_premium_subsidy_schedule()`.

`control`
: a list of control parameters passed to fcip_calculator. see `rfcipCalibrateExtended_control()`

Value

A data.table of summary of business by type, practice, and unit structure for insured acres, liability, premiums, subsidies, and indemnities for a given prediction year.

See Also

Other FCIP Demand Predictors: `get_official_baseline_outcome()`, `predict_final_summary_of_business_crop`

---

rfcipCalibrateExtended_control

Create a control list of adjustment factors for PASS Calculators

---

Description

This function initializes a named list of control parameters (adjustment factors) used throughout the farm policy simulation pipeline. Each element has a sensible default but can be overridden to customize behavior.

Usage

```
rfcipCalibrateExtended_control(
  revenue_lookup_adjustment_factor = 1,
  unit_structure_discount_factor = 1,
  additive_optional_rate_adjustment_factor = 0,
  multiplicative_optional_rate_adjustment_factor = 1,
  capped_revenue_add_on_factor = 0,
  liability_adjustment_factor = 1,
  multiple_commodity_adjustment_factor = 1,
  free_acres_factor = 10/100,
  area_chosen_max_factor = 1/100,
  reported_acres = 1,
  insured_share_percent = 1,
  price_election_percent = 1,
  damage_area_rate = 1,
  rma_rounding = TRUE,
  yield_ratio_cup_and_cap = TRUE,
  set_seed = NULL,
  DEoptim_strategy = 2,
  DEoptim_itermax = 250,
  DEoptim_trace = FALSE,
  continuous_integration_session = FALSE
)
```

Arguments

`revenue_lookup_adjustment_factor`
: Numeric scalar. Multiplier applied to revenue look ups. (Default = 1)

`unit_structure_discount_factor`
: Numeric scalar. Discount factor for unit structure. (Default = 1)

`additive_optional_rate_adjustment_factor`
: Numeric scalar. Additive adjustment to optional rates. (Default = 0)

`multiplicative_optional_rate_adjustment_factor`
: Numeric scalar. Multiplicative adjustment to optional rates. (Default = 1)

`capped_revenue_add_on_factor`
: Numeric scalar. Add-on factor applied to capped revenue. (Default = 0)

`liability_adjustment_factor`
: Numeric scalar. Multiplier applied to liability coverage. (Default = 1)

`multiple_commodity_adjustment_factor`
: Numeric scalar. Adjustment factor when multiple commodities are insured. (Default = 1)

`free_acres_factor`
: Numeric scalar. Proportion of free acres allowed (e.g. 0.10 for 10%). (Default = 0.10) When a policy change results in a 100% premium subsidy or causes an agent to drop coverage entirely, the per-acre out-of-pocket cost becomes zero. To approximate the chosen acres consistent with the revealed budget (the pre-policy total out-of-pocket expenditure), the simulator applies this factor in place of a zero cost.

`area_chosen_max_factor`
: Numeric scalar. Maximum area chosen as a fraction of total (e.g. 0.01 for 1%). (Default = 0.01) Defines the maximum allowable increase in chosen acres when a policy change yields zero out-of-pocket cost (e.g., 100% premium subsidy or dropped coverage). After computing provisional acres via `` `free_acres_factor` ``, the simulator caps the result at:

$$max_acres = (1 + `area_chosen_max_factor`) \times `revealed_acres`$$

where `` `revealed_acres` `` is the pre-policy observed acres (optionally weighted by the number of competing post-policy alternatives). Prevents unrealistically large acre estimates when cost per acre drops to zero.

`reported_acres`
: Numeric scalar. Number of acres reported for insurance purposes. (Default = 1)

`insured_share_percent`
: Numeric scalar. Share of the crop insured (0-1). (Default = 1)

`price_election_percent`
: Numeric scalar. Proportion of the elected price used (0-1). (Default = 1)

`damage_area_rate`
: Numeric scalar. Rate applied to damage-area calculation.(Default= 1)

`rma_rounding`
: logical(1) or numeric(1) If FALSE, rounds only to integer. Otherwise multiplies the number of digits by this factor (mimics round(x, n * rma_rounding)). Defaults to TRUE.

`yield_ratio_cup_and_cap`
: logical(1) If TRUE, enforces a 0.50-1.50 `cup` & `cap` on yield ratios. Defaults to TRUE.

`set_seed`
: Optional integer. Seed for reproducible optimization; defaults to `20250630`.

`DEoptim_strategy`
: defines the Differential Evolution strategy used in the optimization procedure: 1: DE / rand / 1 / bin (classical strategy) 2: DE / local-to-best / 1 / bin (default) 3: DE / best / 1 / bin with jitter 4: DE / rand / 1 / bin with per-vector-dither 5: DE / rand / 1 / bin with per-generation-dither 6: DE / current-to-p-best / 1 any value not above: variation to DE / rand / 1 / bin: either-or-algorithm. Default strategy is currently 2. See *Details* in `DEoptim::DEoptim.control()` documentation.

`DEoptim_itermax`
: the maximum iteration (population generation) passed to `DEoptim::DEoptim.control()` Default is `200`

`DEoptim_trace`
: Positive integer or logical value indicating whether printing of progress occurs at each iteration. Passed to `DEoptim::DEoptim.control()` The default value is TRUE. If a positive integer is specified, printing occurs every trace iterations.

`continuous_integration_session`
: logical(1). If TRUE, a small deterministic subset of the Actuarial Data Master (ADM) YTD ZIP archive is used. This is designed to be safe and fast for use in continuous integration sessions. see `build_min_adm()`

Value

A named list of all control parameters, ready to be passed to other simulation functions.

See Also

Other helpers: `merit_assumption_control()`

Index