# Package 'rfcipCalibrate'

October 12, 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', 'rmaADM', and 'rfcipCalcPass' packages.

**License** GPL-3 + file LICENSE

**URL** https://github.com/you/rfcipCalibrate

**BugReports** https://github.com/you/rfcipCalibrate/issues

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**VignetteBuilder** knitr

**Depends** R (>= 4.1.0)

**Imports** rfcip, data.table, matrixStats, crayon, moments, DEoptim, numDeriv, utils, usmap, devtools, purrr, readr, stringr

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

**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



---

`adjust_revenue_to_yield_equivalence` *Adjust Revenue Protection to Yield-Equivalent Levels*

---

## Description

Adjusts gross revenue protection (GRP) policy values so that they are equivalent to their corresponding yield protection policies, as is done by USDA RMA (Coble et al., 2010).

## Usage

```
adjust_revenue_to_yield_equivalence(
  insurance_plan_code,
  liability_amount,
  indemnity_amount,
  projected_price,
  harvest_price
)
```

## Arguments

`insurance_plan_code`
: Integer vector. Plan codes for the insurance policy-codes 2 and 3 indicate revenue plans that require adjustment.

`liability_amount`
: Numeric vector. The original liability amounts under the revenue policy.

`indemnity_amount`
: Numeric vector. The original indemnity payments calculated under the revenue policy.

`projected_price`
: Numeric vector. The projected price used in the revenue guarantee.

`harvest_price`
: Numeric vector. The actual harvest price used to settle revenue policies.

## Details

For revenue plans (codes 2 and 3), the liability is scaled by $projected\_price/harvest\_price$. For pure yield plans (code 1), no adjustment is made. The loss guarantee for code 2 is further adjusted to use the higher of projected or harvest price. The "production to count" is then the difference between that guarantee and the original indemnity. Finally, the indemnity is re-computed on the yield-equivalent basis and floored at zero to avoid negative payments.

## Value

A named list with two numeric vectors:

**liability_amount_adj** Liability amounts adjusted so that the revenue policy is equivalent to a yield policy.

**indemnity_amount_adj** Indemnity amounts adjusted for yield- equivalence, floored at zero.

## References

Coble, K. H., Heimlich, R., & Jolly, R. (2010). *Comprehensive Review of RMA Loss Adjustment Procedures* (USDA RMA). https://legacy.rma.usda.gov/pubs/2009/comprehensivereview.pdf

## See Also

Other Yield/Revenue Calibration: `calibrate_yield()`, `prepare_yield_calibration_data()`, `rate_yield_optimizer()`, `rma_500_revenue_draw()`

## Examples

```
# No adjustment for plan 1 (yield policy):
adjust_revenue_to_yield_equivalence(
  insurance_plan_code = 1,
  liability_amount     = 1000,
  indemnity_amount     = 200,
  projected_price      = 5.50,
  harvest_price        = 5.00
)

# Adjustment for plan 2 or 3 (revenue policies):
adjust_revenue_to_yield_equivalence(
```

```
  insurance_plan_code = c(2, 3),
  liability_amount     = c(1000, 1000),
  indemnity_amount     = c(150, 100),
  projected_price      = c(5.50, 5.50),
  harvest_price        = c(6.00, 4.25)
)
```

---

`adm_commodities` *Simulator Helper Datasets*

---

## Description

A combined dataset for adm_commodities

## Usage

```
data(adm_commodities)
```

## Format

A data frame with 1641 rows and 5 columns covering 2011-2025.

## Source

USDA-RMA, Actuarial Data Master (ADM)

---

`adm_counties` *adm_counties*

---

## Description

A combined dataset for adm_counties

## Usage

```
data(adm_counties)
```

## Format

A data frame with 3194 rows and 5 columns covering Inf–Inf.

## Source

USDA-RMA, Actuarial Data Master (ADM)

`adm_production_practices` *adm_production_practices*

## Description

A combined dataset for adm_production_practices

## Usage

```
data(adm_production_practices)
```

## Format

A data frame with 17660 rows and 12 columns covering 2011-2025.

## Source

USDA-RMA, Actuarial Data Master (ADM)

`adm_states` *adm_states*

## Description

A combined dataset for adm_states

## Usage

```
data(adm_states)
```

## Format

A data frame with 50 rows and 5 columns covering Inf–Inf.

## Source

USDA-RMA, Actuarial Data Master (ADM)

## adm_types *adm_types*

### Description

A combined dataset for adm_types

### Usage

```
data(adm_types)
```

### Format

A data frame with 8579 rows and 6 columns covering 2011-2025.

### Source

USDA-RMA, Actuarial Data Master (ADM)

## apply_controls *Ensure specified fields exist in a data.table, filling in defaults*

### Description

Given a set of field names and a data.frame or data.table, this function checks for any missing columns and adds them, pulling default values from a control list (e.g. created by `rfcipCalibrate_controls`).

### Usage

```
apply_controls(fields, data, control = rfcipCalibrate_controls())
```

### Arguments

| | |
|---|---|
| `fields` | Character vector. Names of columns that must be present in `data`. |
| `data` | A `data.frame` or `data.table` to operate on. Converted by reference. |
| `control` | A named list of default column values; typically from `rfcipCalibrate_controls`. |

### Value

The input `data` as a `data.table`, with any missing `fields` added and populated by the corresponding element in `control`.

### See Also

Other helpers: `rfcipCalibrate_controls()`

## Examples

```
## Not run:
# Sample data with only one column
df <- data.frame(id = 1:3)

# Ensure insurance fields exist
out <- apply_controls(
  fields = c("insured_share_percent", "damage_area_rate"),
  data   = df,
  control = rfcipCalibrate_controls(insured_share_percent = 0.8)
)


## End(Not run)

# Now out has id, insured_share_percent, and damage_area_rate
```

---

# calculate_mode *Calculate the Statistical Mode*

---

## Description

Returns the element that occurs most frequently in a vector.

## Usage

```
calculate_mode(x, na.rm = TRUE)
```

## Arguments

| | |
|---|---|
| x | A vector of any atomic type (numeric, character, factor,). |
| na.rm | Logical; should missing values be ignored? Defaults to TRUE. If FALSE and x contains any NAs, the function returns NA. |

## Details

Internally the function:

1. Optionally removes NAs (na.rm = TRUE).
2. Builds a lookup table of unique values via unique(x).
3. Counts the frequency of each unique value with tabulate(match(x, ux)).
4. Returns the value with the maximum count.

Because it relies on base R functions, the implementation is vectorised and generally fast for typical data-frame column sizes.

## Value

A single value giving the modal element of x. If two or more values are tied for the highest frequency, the first one encountered in x is returned.

`calculate_revenue_moments` *Calculate key distributional moments for a numeric vector of revenues*

## Description

Calculate key distributional moments for a numeric vector of revenues

## Usage

```
calculate_revenue_moments(x)
```

## Arguments

| | |
|---|---|
| x | Numeric vector of revenues (e.g. per-draw revenues) |

## Value

A one-row data.frame with:

- mu: mean
- md: median
- sd: standard deviation
- cv: coefficient of variation (sd / mu)
- vr: variance
- sk: skewness
- ku: kurtosis
- lapv: mean squared loss deviations below the mean
- lrpv: lapv / probability of loss
- nlapv: lapv normalized by mu
- nlrpv: lrpv normalized by mu
- lres2: same as lapv (for clarity)
- cdf: empirical probability of loss (P(X < mu))

`calibrate_preferences` *Calibrate Agent Preferences via Merit Specification - lite 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(
  agent_data,
  functional_form = ~b0 + b1 * log(ret) + b2 * log(risk) + b3 * log(ret)^2 + b4 *
    log(risk)^2 + b5 * log(ret) * log(risk),
  control = rfcipCalibrate_controls()
)
```

## Arguments

| | |
|---|---|
| agent_data | A data.table or data frame where each row is an agent and includes list-columns:<br>menu_mu List of expected returns for each menu option.<br>menu_risk_measure List of risk measures matching risk_measure.<br>no_insurance Logical list indicating the no insurance option.<br>observed Logical list indicating the observed choice option.<br>menu_covlevl List of coverage levels for each option.<br>menu_plan List of plan identifiers for each option.<br>producer_id Identifier used to group agents. |
| functional_form | A one-sided formula (e.g. ~ b0 + b1 * ret + b2 * risk) defining the merit function whose coefficients will be estimated. |
| control | a list of control parameters; see rfcipCalibrate_controls(). 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

## See Also

Other Producer Preference Calibration: `merit_function()`, `merit_objective_function()`

---

`calibrate_yield` *Yield calibration workhorse*

---

## Description

Calibrates the Federal Crop Insurance Program (FCIP) yield for a single crop year by finding the optimal production history length (`T_0`, `T_1`) and yield offset `dy_Opt` that minimize the squared deviation between observed rates and the model-predicted rate in the following year. Uses a data.table-based grid search over integer `T_0[2,10]`, `T_1[3,10]`, and an `optimize()` call for `dy_Opt`.

## Usage

```
calibrate_yield(
  sobtpu_current,
  sobtpu_subsequent,
  control = rfcipCalibrate_controls()
)
```

## Arguments

| | |
|---|---|
| `sobtpu_current` | The target year prepared sobtpu, typically returned by `prepare_yield_calibration_data`. |
| `sobtpu_subsequent` | The prepared sobtpu for the year immediately following the target year, typically returned by `prepare_yield_calibration_data`. |
| `control` | a list of control parameters;`continuous_integration_session` set via control to determine the ADM source. see `rfcipCalibrate_controls`. |

## Value

A data.table containing, for each insurance-pool-election record with a calibrated yield

## References

Tsiboe, Francis, Dylan Turner, and Jisang Yu. (2025) Utilizing large-scale insurance data sets to calibrate sub-county level crop yields. Journal of Risk and Insurance, 92(1), 139-165.

## See Also

Other Yield/Revenue Calibration: adjust_revenue_to_yield_equivalence(), prepare_yield_calibration_data(), rate_yield_optimizer(), rma_500_revenue_draw()

---

construct_menu_option_fcip *Construct Basic FCIP Menu Options*

---

## Description

Generates a menu of insurance options (none, group/area, individual/APH) for a given farm dataset, year, and desired protection levels.

## Usage

```
construct_menu_option_fcip(
  farmdata,
  farm_identifiers = NULL,
  year = 2020,
  protection_level = c("individual", "group", "none"),
  premium_subsidy_control = NULL,
  control = rfcipCalibrate_controls()
)
```

## Arguments

| | |
|---|---|
| farmdata | A data.table of farm-level input. contain at least 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. |
| year | Numeric scalar giving the target insurance crop year (e.g. 2024). Default is 2020. |
| protection_level | Character vector specifying which menus to include. Can include any of "individual", "group", and "none". Default is all. |
| premium_subsidy_control | A named list of adjustment functions, typically created using rfcipCalcPass::premium_subsidy_c If NULL, the default adjustment functions are used. |
| control | a list of control parameters passed to rfcipCalcPass::fcip_calculator(); see rfcipCalibrate_controls and rfcipCalcPass::fcip_calculator() |

## Details

Internally this:

1. Ensures farmdata is a data.table and tags it with commodity_year.
2. Creates a producer_id if no identifiers provided.
3. Checks for required columns (rate_yield, approved_yield, FCIP_INSURANCE_POOL) and errors out if missing.

4. Builds the none menu (zero premium & full subsidy).
5. If "group" requested, pulls area-based rating data via {rfcipCalcPass::get_adm_data_extended()} (using different datasets for 2011-2016 vs. 2017+), averages rates, merges in index values and subsidy percentages, and computes final premium & subsidy.
6. If "individual" requested, pulls APH eligibility & rate-differential tables, reshapes to unit structures (OU/BU/EU), then calls rfcipCalcPass::fcip_calculator() per election to compute premium & subsidy.
7. Binds all selected menus together and returns the combined table.

## Value

A data.table: Each row corresponds to one farm & unit structure, insurance plan, coverage type, and coverage level combination, with columns for premium rate, subsidy rate, user defind farm identifier in farm_identifiers and all fields in rfcipCalcPass::FCIP_INSURANCE_POOL and rfcipCalcPass::FCIP_INSURANCE_ELECTION.

## See Also

Other FCIP Menu Calibration: simulate_menu_revenue_fcip()

---

# full_calibration_dispatcher

*Full Calibration Dispatcher for FCIP Yield and Revenue Models*

---

## Description

This function runs the full Federal Crop Insurance Program (FCIP) calibration pipeline for a given year and crop/location. It downloads data, prepares it for calibration, performs yield calibration, simulates revenue outcomes, constructs insurance menus, and calibrates producer preferences. All outputs are saved in a structured project-specific directory for reproducibility and reuse.

## Usage

```
full_calibration_dispatcher(
  project_name = NULL,
  calibration_name = NULL,
  year = 2020,
  crop = NULL,
  state = NULL,
  county = NULL,
  fips = NULL,
  clear_adm_cache = FALSE,
  clear_project_cache = FALSE,
  control = rfcipCalibrate_controls()
)
```

## Arguments

| | |
|---|---|
| `project_name` | Character. A name used to organize and cache outputs from this calibration run (e.g., "corn_iowa_2024"). |
| `calibration_name` | Character. A short label used to name the saved `.rds` files within the project directory. |
| `year` | Integer. The commodity year to retrieve data for. Default is 2020. |
| `crop` | Character or integer vector. Crop name(s) or code(s). Accepts either commodity names or numeric crop codes. Use `rfcip::get_crop_codes()` or `get_crop_codes_extended()` to view valid crops. |
| `state` | Character or integer. State abbreviation (e.g., "IA"), full state name (e.g., "Iowa"), or 2-digit FIPS code. |
| `county` | Character or integer. County name (e.g., "Story") or 5-digit FIPS code. If using the county name, the `state` parameter must also be provided. |
| `fips` | Integer. 5-digit county FIPS code (alternative to using `state` and `county` together). |
| `clear_adm_cache` | Logical. If `TRUE`, clears cached ADM data using `rfcip::clear_rfcip_cache()` and `rfcipCalcPass::clear_rfcipCalcPass_cache()`. |
| `clear_project_cache` | Logical. If `TRUE`, deletes all saved files for the given project before running the calibration. |
| `control` | A list of control parameters passed to all stages. Includes optimizer settings and toggles like `continuous_integration_session`. See `rfcipCalibrate_controls` for details. |

## Details

The following outputs are saved in the project cache directory:

- `*_data_sob_current.rds`, `*_data_cal_current.rds`: SOB and calibration inputs for year `t`
- `*_data_sob_subsequent.rds`, `*_data_cal_subsequent.rds`: Optional year `t+1` data if not running in CI mode
- `*_cal_yield.rds`: Yield calibration output
- `*_revenue_draw.rds`: 500 simulated revenue draws
- `*_cal_revenue.rds`: Merged yield and revenue simulation data
- `*_menu_option.rds`: Constructed insurance menu options
- `*_cal_menu.rds`: Simulated menu-level revenue outcomes
- `*_cal_preference.rds`: Estimated producer preference parameters

## Value

This function is called for its side effects: downloading, processing, calibrating, and saving `.rds` files to disk. Returns `NULL` invisibly.

## get_crop_codes_extended

*Lookup Crop Codes for FCIP Commodities (Extended Version)*

### Description

This function provides an extended interface to retrieve crop codes and names used in the Federal Crop Insurance Program (FCIP). It attempts to use the `rfcip::get_crop_codes()` function to pull data by year and crop. If that fails, it defaults to using the internal `adm_commodities` dataset.

### Usage

```
get_crop_codes_extended(
  year = as.numeric(format(Sys.Date(), "%Y")),
  crop = NULL,
  control = rfcipCalibrate_controls()
)
```

### Arguments

| | |
|---|---|
| `year` | A single numeric year or a numeric vector of years for which to retrieve crop information. Defaults to the current year. |
| `crop` | A vector of crop identifiers, either as character strings (commodity names, case-insensitive) or numeric crop codes. If `NULL`, all crop codes for the specified year(s) are returned. |
| `control` | A list of control parameters. `continuous_integration_session` can be set via `control` to determine the ADM data source. See `rfcipCalibrate_controls`. |

### Details

This function wraps around `rfcip::get_crop_codes()` and provides a fallback to `adm_commodities` if the primary function call fails. This ensures more robust behavior in cases where external data access fails.

### Value

A tibble containing crop codes and names for the specified year(s). The tibble typically includes at least the year, crop code, and crop name.

## get_fcip_calibrated_data

*Get calibrated FCIP data*

### Description

Downloads (and caches) Farm Bill FCIP calibration artifacts from the USFarmSafetyNetLab GitHub Releases, returning a filtered `data.table`. Supports vectorized queries over years and common geographic & crop filters.

## Usage

```
get_fcip_calibrated_data(
  year = NULL,
  dataset = NULL,
  crop = NULL,
  state = NULL,
  county = NULL,
  fips = NULL,
  force = FALSE
)
```

## Arguments

| | |
|---|---|
| year | Numeric scalar or vector. One or more ADM years (e.g., 2020 or 2018:2022). If multiple years are provided, each year is downloaded/read and row-bound into a single result. Default NULL (must be supplied). |
| dataset | Character (length 1). Which dataset to retrieve. One of:<br>• "revenue_draw" - 500 simulated revenue draws for each observed FCIP aggregated transaction, generated using the RMA M-13 framework with ADM parameters, ensuring all menu alternatives are evaluated under the same stochastic scenarios.<br>• "calibrated_revenue" - Combined dataset of calibrated yields and 500 simulated revenue draws for each observed FCIP aggregated transaction.<br>• "menu_option" - All available Basic FCIP menu options for each observed FCIP aggregated transaction.<br>• "calibrated_menu" - For each observed FCIP aggregated transaction, the feasible FCIP menu is constructed for the study year/location, with revenues simulated for every alternative across the same 500 correlated scenarios.<br>• "calibration_data", "calibrated_yield", "calibrated_preference" - additional calibration artifacts when available. |
| crop | Character, numeric, or vector. Crop identifier(s). Accepts common names (e.g., "corn", "soybeans") or commodity codes (e.g., 41). Names are converted to codes via rfcip::get_crop_codes(). Use rfcip::get_crop_codes() to discover available crops/codes. |
| state | Character, numeric, or vector. U.S. state(s) as two-letter abbreviations (e.g., "ND"), full names (e.g., "North Dakota"), or FIPS state codes (e.g., 38). Values are normalized to FIPS via usmap::fips(). |
| county | Character, numeric, or vector. County filter as county name(s) or 5-digit county FIPS code(s). If supplying county names, you must also supply state. When county names are used with state, they are resolved to 3-digit county FIPS via usmap::fips(). |
| fips | Numeric or character, length 5 or vector. Full 5-digit county FIPS code(s). When provided, fips takes precedence and state/county are derived from it. |
| force | Logical. If TRUE, attempts to re-download each requested file even if a cached copy exists (falls back to cache on download failure with a warning). Default FALSE. |

## Details

For each year, the function constructs a release asset name <dataset>_<year>.rds under the GitHub tag <dataset> and downloads it to the cache (or reads from cache), then applies optional filters: commodity_code (crop), state_code (2-digit, char), county_code (3-digit, char).

## Value

A data.table containing the requested dataset row-bound across all requested years and filtered by the supplied crop/state/county/fips. The column schema depends on dataset.

## Caching

Files are stored under tools::R_user_dir("rfcipCalibrate", "cache"). Subsequent calls reuse cached .rds files unless force = TRUE.

## Filtering precedence

1. If fips is provided, it is normalized via clean_fips() and used to derive state (2-digit) and county (3-digit).
2. Otherwise, if state and county are both provided:
   - If county looks like a 5-digit FIPS, split into state+county.
   - Else, resolve county names within the given state via usmap::fips().
3. If only state is provided, filter at the state level.

## Examples

```
## Not run:
# State-level calibrated revenue for corn (by name)
dt <- get_fcip_calibrated_data(
  year = 2020,
  dataset = "calibrated_revenue",
  crop = "corn",
  state = "ND"
)

# Multiple years and multiple crops (by codes) with county FIPS filtering
dt2 <- get_fcip_calibrated_data(
  year = 2019:2021,
  dataset = "revenue_draw",
  crop = c(41, 81),              # corn, soybeans
  fips = c("38017", "38035")     # Cass, Grand Forks (ND)
)

# Force-refresh cached files
dt3 <- get_fcip_calibrated_data(
  year = c(2020, 2021),
  dataset = "menu_option",
  force = TRUE
)

## End(Not run)
```

`get_sob_data_extended` *Download and Import Summary of Business (Extended Version)*

## Description

This function provides an extended interface to retrieve the Risk Management Agency's (RMA) Summary of Business (SOB) data. It first attempts to pull the data using `rfcip::get_sob_data()`. If that fails, it falls back to downloading a pre-processed `.rds` version from the USFarmSafetyNet-Lab GitHub releases.

## Usage

```
get_sob_data_extended(
  year = 2020,
  crop = NULL,
  state = NULL,
  county = NULL,
  fips = NULL,
  control = rfcipCalibrate_controls()
)
```

## Arguments

| | |
|---|---|
| `year` | Integer. The commodity year to retrieve data for. Default is 2020. |
| `crop` | Character or integer vector. Crop name(s) or code(s). Accepts either commodity names or numeric crop codes. Use `rfcip::get_crop_codes()` or `get_crop_codes_extended()` to view valid crops. |
| `state` | Character or integer. State abbreviation (e.g., "IA"), full state name (e.g., "Iowa"), or 2-digit FIPS code. |
| `county` | Character or integer. County name (e.g., "Story") or 5-digit FIPS code. If using county name, the `state` parameter must also be provided. |
| `fips` | Integer. 5-digit county FIPS code (alternative to using `state` and `county` together). |
| `control` | A list of control parameters. `continuous_integration_session` can be set via `control` to determine the ADM data source. See `rfcipCalibrate_controls`. |

## Details

If the call to `rfcip::get_sob_data()` fails (e.g., due to service unavailability), the function downloads a cached `.rds` version of the data from GitHub. Data is then filtered based on the provided `crop`, `state`, `county`, or `fips` values. Only policies with coverage type `"A"` and insurance plans in `{1, 2, 3, 90}` are retained. Negative indemnities are coerced to zero.

## Value

A `data.table` containing the filtered Summary of Business data. Includes numeric and categorical columns such as liability, indemnity, state/county codes, and coverage type.

## 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_function *Symbolically Analyze and Evaluate a Merit Function*

### Description

This function takes a symbolic merit function with coefficients (e.g., `b0`, `b1`, ...) and computes its symbolic first- and second-order derivatives with respect to `ret` and `risk`. If numeric values for parameters and variables are provided, it evaluates the merit and all derivatives at those points.

## Usage

```
merit_function(
  functional_form,
  parameter_values = NULL,
  ret = NULL,
  risk = NULL
)
```

## Arguments

| | |
|---|---|
| functional_form | A one-sided formula object (e.g., ~ b0 + b1 * ret + b2 * risk + ...) representing the merit function to analyze. |
| parameter_values | Optional numeric vector of parameter values. If unnamed, the function will assign names as b0, b1, etc. Must match the number and order of coefficients in the formula. |
| 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. |

## Details

If parameter_values is unnamed, the function will assign names automatically as b0, b1, ..., in the order they appear.

## Value

A named list containing:

equation The original merit function as a symbolic expression.

dU_dret First derivative with respect to ret (symbolic).

dU_drisk First derivative with respect to risk (symbolic).

d2U_dret2 Second derivative with respect to ret (symbolic).

d2U_drisk2 Second derivative with respect to risk (symbolic).

d2U_dretdrisk Mixed second derivative with respect to ret and risk (symbolic).

values (Optional) A list of evaluated numeric vectors for the merit and its derivatives.

## See Also

Other Producer Preference Calibration: calibrate_preferences(), merit_objective_function()

## Examples

```
f <- ~ b0 + b1 * ret + b2 * risk + 0.5 * b3 * ret^2 + 0.5 * b4 * risk^2 + b5 * ret * risk
theta <- c(1, 2, 3, 4, 5, 6)  # unnamed vector is fine
ret <- c(1, 2, 3, 5, 9, 7)
risk <- c(0.6, 0.8, 0.6, 0.5, 0.8, 1)
result <- merit_function(f, parameter_values = theta, ret = ret, risk = risk)
result$equation
result$values$U
```

`merit_objective_function` *Merit-based DEoptim Objective Function - lite 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(parameter_values, ret, risk, functional_form)
```

## 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. |

## 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 to infer profit maximization.
- Sign/equality constraints on marginal utility of risk to infer risk minimization
- Quadratic penalty if the implied marginal rate of substitution exceeds 10.

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()`, `merit_function()`

Other Producer Preference Calibration: `calibrate_preferences()`, `merit_function()`

`prepare_yield_calibration_data` *Aggregate Data for FCIP Yield Calibration*

## Description

This function processes Summary of Business (SOB) and Actuarial Data Master (ADM) data for a single year and computes the variables necessary to calibrate the Federal Crop Insurance Program (FCIP) yield model, as described in Tsiboe et al. (2025).

Internally, the function:

- Filters SOB data to include only coverage type "A" and insurance plans 1, 2, 3, and 90.
- Joins with ADM actuarial parameters for the relevant insurance pools and elections.
- Computes adjusted liabilities and indemnities for revenue policies to yield equivalents.
- Derives approved yields and extrapolates implied rate yields via optimization.
- Computes key rating factors like alpha, delta, and base premium rates.

## Usage

```
prepare_yield_calibration_data(sobtpu, control = rfcipCalibrate_controls())
```

## Arguments

| | |
|---|---|
| `sobtpu` | A `data.table` of filtered Summary of Business data, typically returned by `get_sob_data_extended`. |
| `control` | A list of control parameters for model calibration. Passed to the `DEoptim` optimizer and other internals. Use `rfcipCalibrate_controls` to create and configure this list. |

## Value

A `data.table` with one row per unique insurance contract (defined by year, pool, and election), containing:

- `commodity_year`, plus all variables in `FCIP_INSURANCE_POOL` and `FCIP_INSURANCE_ELECTION`
- `rate_yield`: The optimized yield input used in rating.
- `approved_yield`: Yield used to compute liability.
- `yield_indem`: Per-acre indemnified yield.
- `reference_amount`, `reference_rate`, `fixed_rate`, `exponent_value`
- `rate_differential_factor`, `unit_residual_factor`, `alpha`, `delta`
- `base_premium_rate_adj`, `simulation_weight`, and `revealed_budget`

Observations with missing or invalid optimization results are dropped.

## References

Tsiboe, Francis, Dylan Turner, and Jisang Yu. (2025). Utilizing large-scale insurance data sets to calibrate sub-county level crop yields. *Journal of Risk and Insurance*, 92(1), 13-165.

Coble, K. H., Knight, T. O., Goodwin, B. K., Miller, M. F., Rejesus, R. M., & Duffield, G. (2010). A comprehensive review of the RMA APH and COMBO rating methodology final report. *USDA Risk Management Agency*.

## See Also

Other Yield/Revenue Calibration: adjust_revenue_to_yield_equivalence(), calibrate_yield(), rate_yield_optimizer(), rma_500_revenue_draw()

## Examples

```
## Not run:
sob <- get_sob_data_extended(year = 2024, crop = "Corn", state = "IA")
sob_filtered <- sob[coverage_type_code == "A" & insurance_plan_code %in% c(1, 2, 3, 90)]
result <- prepare_yield_calibration_data(sobtpu = sob_filtered)

## End(Not run)
```

---

# rate_yield_optimizer *Objective Function for Rate Yield Optimization*

---

## Description

Computes the squared difference between the base premium rate implied by a candidate rate_yield and the observed base_premium_rate_expected. Intended for use with optimization routines (e.g., DEoptim) to invert the RMA premium-rate pipeline.

## Usage

```
rate_yield_optimizer(
  rate_yield,
  base_premium_rate_expected,
  reference_amount,
  reference_rate,
  fixed_rate,
  rate_differential_factor,
  unit_residual_factor,
  exponent_value,
  prior_year_reference_amount,
  prior_year_reference_rate,
  prior_year_fixed_rate,
  prior_year_rate_differential_factor,
  prior_year_unit_residual_factor,
  prior_year_exponent_value,
  rate_yield_control = rfcipCalibrate_controls()
)
```

## Arguments

| | |
|---|---|
| rate_yield | Numeric. Candidate yield per acre evaluated by the optimizer. |
| base_premium_rate_expected | Numeric. Target base premium rate to match. |
| reference_amount | Numeric. Current-year reference yield for ratio calculation. |
| reference_rate | Numeric. Rate component for current-year base rate calculation. |

`fixed_rate` Numeric. Fixed increment added after exponentiation (current year).

`rate_differential_factor`
Numeric. Differential factor for current-year base premium.

`unit_residual_factor`
Numeric. Unit residual factor for current-year base premium.

`exponent_value` Numeric. Exponent used on current-year yield ratio.

`prior_year_reference_amount`
Numeric. Reference yield for the prior year.

`prior_year_reference_rate`
Numeric. Rate component for prior-year base rate.

`prior_year_fixed_rate`
Numeric. Fixed increment for prior-year calculation.

`prior_year_rate_differential_factor`
Numeric. Differential factor for prior-year premium.

`prior_year_unit_residual_factor`
Numeric. Unit residual factor for prior-year premium.

`prior_year_exponent_value`
Numeric. Exponent used on prior-year yield ratio.

`rate_yield_control`
List. Control settings from `rfcipCalcPass_control()` forwarded to `calc_base_premium_rate()`.

## Value

Numeric. Squared error between computed and expected base premium rates; a large penalty (1e10) is returned if the result is not finite.

## See Also

Other Yield/Revenue Calibration: `adjust_revenue_to_yield_equivalence()`, `calibrate_yield()`, `prepare_yield_calibration_data()`, `rma_500_revenue_draw()`

---

# `rfcipCalibrate_controls` *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

```
rfcipCalibrate_controls(
  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,
  adm_decoy_state_abb = "ND"
)
```

## 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()`

`adm_decoy_state_abb`
: tate abbreviation indicating which state's decoy ADM to use (default is ND).

## Value

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

## See Also

Other helpers: `apply_controls()`

## Examples

```
## Not run:
# Use all defaults:
ctrl <- rfcipCalibrate_controls()
```

```
# Override a couple of factors:
ctrl2 <- rfcipCalibrate_controls(
  free_acres_factor = 0.15,
  liability_adjustment_factor = 0.9
)


## End(Not run)
```

---

`rma_500_revenue_draw` *Generate 500 Simulated Revenue Draws for FCIP*

---

## Description

Given a calibrated producer data set, this function:

1. Creates a placeholder `lookup rate` by averaging `rate_yield` and `calibrated_yield`, then running the FCIP calculator `rfcipCalcPass::fcip_calculator()` to obtain `revenue_lookup_rate`.
2. Joins in ADM parameters (beta IDs, price volatility factors, harvest prices, combo revenue factors).
3. Computes adjusted mean and standard-deviation of yield, and log-mean of price.
4. Retrieves 500 RMA beta draws and uses them to generate 500 simulated yields and prices per farm.
5. Returns a data.table with one row per farm (grouped by `farm_identifiers`) containing lists of draws.

## Usage

```
rma_500_revenue_draw(
  farmdata,
  farm_identifiers,
  control = rfcipCalibrate_controls()
)
```

## Arguments

| | |
|---|---|
| `farmdata` | A data.frame or data.table of observed or/and calibrated producer records. Must include at least:<br>• `rate_yield` and `calibrated_yield` (for lookup rate)<br>• `approved_yield`, `mean_quantity`, `standard_deviation_quantity`<br>• `commodity_year` and all fields in `rfcipCalcPass::FCIP_INSURANCE_POOL` |
| `farm_identifiers` | column name(s) that uniquely identify each record/farm/producer. If `NULL`, a "producer_id" column is created. |
| `control` | a list of control parameters;`continuous_integration_session` set via control to determine the ADM source. see `rfcipCalibrate_controls`. |

## Value

A `data.table` with one row per unique farm identifier. Columns:

- `rma_draw_lookup_rate`: Mean of the simulated lookup rate (should equal the pre-draw lookup rate)
- `rma_draw_number`: List of 500 sequence numbers (1-500).
- `rma_draw_yield`: List of 500 simulated yields (numeric).
- `rma_draw_price`: List of 500 simulated prices (numeric).

## See Also

Other Yield/Revenue Calibration: `adjust_revenue_to_yield_equivalence()`, `calibrate_yield()`, `prepare_yield_calibration_data()`, `rate_yield_optimizer()`

## Examples

```
## Not run:
library(data.table)
farmdata <- readRDS("data-raw/calibrated_producer.rds")
# Suppose each record has a column "producer_id"
results <- rma_500_revenue_draw(farmdata, "producer_id")
str(results)

## End(Not run)
```

---

`simulate_menu_revenue_fcip`

*Simulate Basic FCIP Menu Revenues*

---

## Description

Generates a `menu` of insurance options (none, group/area, individual/APH) for a given farm dataset, year, and desired protection levels.

## Usage

```
simulate_menu_revenue_fcip(
  year = 2011,
  farmdata,
  menu_option,
  farm_identifiers = NULL,
  control = rfcipCalibrate_controls()
)
```

## Arguments

| | |
|---|---|
| `year` | Numeric scalar giving the target insurance crop year (e.g. `2024`). Default is `2011`. |
| `farmdata` | A `data.table` of farm-level input. contain at least all columns required by `rfcipCalcPass::fcip_calculator()` |
| `menu_option` | A data.frame or data.table Basic FCIP Menu Options constructed with `construct_menu_option_fci` |

`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

Internally this:

1. Ensures `farmdata` is a `data.table` and tags it with `commodity_year`.
2. Creates a `producer_id` if no identifiers provided.
3. Checks for required columns (`rate_yield`, `approved_yield`, `FCIP_INSURANCE_POOL`) and errors out if missing.
4. Builds the `none` menu (zero premium & full subsidy).
5. If `"group"` requested, pulls area-based rating data via `rfcipCalcPass::get_adm_data_extended()` (using different datasets for 2011-2016 vs. 2017+), averages rates, merges in index values and subsidy percentages, and computes final premium & subsidy.
6. If `"individual"` requested, pulls APH eligibility & rate-differential tables, reshapes to unit structures (OU/BU/EU), then calls `rfcipCalcPass::fcip_calculator()` per election to compute premium & subsidy.
7. Binds all selected menus together and returns the combined table.

## Value

A `data.table`: Each row corresponds to one farm & unit structure, insurance plan, coverage type, and coverage level combination, with columns for premium rate, subsidy rate, user defind farm identifier in `farm_identifiers` and all fields in `FCIP_INSURANCE_POOL` and `FCIP_INSURANCE_ELECTION`.

## See Also

Other FCIP Menu Calibration: `construct_menu_option_fcip()`

# Index