Outputs of the sustainable food security modeling framework for West Africa

Debbora Leip

May 18, 2021

1 Outputs of `FoodSecurityProblem()`

- `settings: dict`
  Input settings to the model framework.
  - `PenMet: 'prob'` or `'penalties'`
    `'prob'` if desired probabilities are given and penalties are to be calculated accordingly. `'penalties'` if input penalties are to be used directly. The default is `'prob'`.
  - `probF: float` or `None`
    Demanded probability for keeping the food demand constraint (only relevant if `PenMet` is `'prob'`). The default is 0.99.
  - `probS: float` or `None`
    Demanded probability for keeping the solvency constraint (only relevant if `PenMet` is `'prob'`). The default is 0.95.
  - `rhoF: float` or `None`
    If `PenMet` is `'penalties'`, this is the value that will be used for `args['rhoF']`. if `PenMet` is `'prob'`, `args['rhoF']` will be calculated to according to the probability. If `rhoF` is not `None`, this value will then be used as an initial guess for the correct penalty. If it is `None`, the penalty resulting from a run with the same settings but lower sample size will be used as initial guess (if available). The default is None.
  - `rhoS: float` or `None`
    If `PenMet` is `'penalties'`, this is the value that will be used for `args['rhoS']`. if `PenMet` is `'prob'`, `args['rhoS']` will be calculated to provide the demanded probability. If `rhoS` is not `None`, this value will then be used as an initial guess for the correct penalty. If it is `None`, the penalty resulting from a run with the same settings but lower sample size will be used as initial guess (if available). The default is None.
  - `k: int`
    Number of clusters in which the area is to be divided. The default is `9`.
  - `k_using: 'all'` or `list` of `ints` $i \in 1, ..., k$
    Specifies which of the clusters are to be considered by the model. The default is the representative cluster `[3]`.
  - `num_crops: int`
    The number of crops that are used. The default is `2`.
  - `yield_projection: str` (`'fixed'` or `'trend'`)
    If `'fixed'`, the yield distributions of the year prior to the first year of simulation are used for all years. If `'trend'`, the mean of the yield distributions follows the linear trend. The default is `'fixed'`.
  - `sim_start: int`
    The first year of the simulation. The default is `2017`.

- `pop_scenario: str`
  Specifies which population scenario should be used. `'fixed'` uses the population of the year prior to the first year of the simulation for all years. The other options are `'Medium'`, `'High'`, `'Low'`, `'ConstantFertility'`, `'InstantReplacement'`, `'ZeroMigration'`, `'ConstantMortality'`, `'NoChange'`, `'Momentum'`, referring to the different United Nations World Population Prospects scenarios. All scenarios have the same estimates up to (including) 2019, scenario-specific predictions start from 2020 The default is `'fixed'`.
- `risk: int`
  The risk level that is covered by the government. E.g. if risk is `0.05`, yields in the lower 5% quantile of the yield distributions will be considered as catastrophic. The default is `0.05`.
- `N: int`
  Number of yield samples to be used to approximate the expected value in the original objective function. The default is `10000`.
- `validation_size: int` or `None`
  If not `None`, the objevtice function will be re-evaluated for validation with a higher sample size which is given by this parameter. The default is None.
- `T: int`
  Number of years to cover in the simulation. The default is `20`.
- `seed: int`
  Seed to allow for reproduction of the results. The default is `201120`.
- `tax: float`
  Tax rate to be payed on farmers profits. The default is `0.01`.
- `perc_guaranteed: float`
  The percentage that determines how high the guaranteed income will be depending on the expected income of farmers in a scenario excluding the government. The default is `0.9`.
- `ini_fund: float`
  Initial size of the fund. The default is `0`. Given in $[10^9\,\$]$.
- `import: float`
  Food import that will be subtracted from demand of every year. The default is 0. Given in $[10^{12}\,\text{kcal}]$.

- `args: dict`
  Additional model inputs calculated based on the `settings`.
  - `k: int`
    Number of clusters in which the area is to be divided.
  - `k_using: 'all'` or `list` of `ints` $i \in 1, ..., k$
    Specifies which of the clusters are to be considered by the model.
  - `num_crops: int`
    The number of crops that are used. The default is `2`.
  - `N: int`
    Number of yield samples to be used to approximate the expected value in the original objective function. The default is `10000`.
  - `cat_clusters: np.array` of `ints` of shape `(N, T, len(k_using))`
    Indicating clusters with yields labeled as catastrophic with `1`, clusters with "normal" yields with `0`.
  - `terminal_years: np.array` of `ints` of shape `(N, )`
    Indicating the first year with a catastrophic cluster for each sample. `-1` for samples without catastrophe.

- `ylds: np.array` of `floats` of shape `(N, T, num_crops, len(k_using))`
  Yield samples according to the presence of catastrophes. Given in in [t/ha].
- `costs: np.array` of `floats` of shape `(num_crops, len(k_using))`
  Cultivation costs for each crop in each cluster (currently the same for each cluster). Given in $[10^9\,\$/10^6\,\text{ha}]$.
- `demand: np.array` of `floats` of shape `(T, )`
  Total food demand for each year based on population. Given in $[10^{12}\,\text{kcal}]$.
- `ini_fund: float` or `int`
  Initial size of the fund. Currently not used as a input to the model. Given in $[10^9\,\$]$.
- `import: float` or `int`
  Food import which will be subtracted from the demand. Currently not used as a input to the model. Given in $[10^{12}\,\text{kcal}]$.
- `tax: float`
  Tax rate to be payed on farmers profits (and losses).
- `prices: np.array` of `floats` of shape `(num_crops, len(k_using))`
  Farm gate prices farmers earn for each crop in each cluster (currently the same for each cluster). Given in $[10^9\,\$/10^6\,\text{t}]$.
- `T: int`
  Number of years to cover in the simulation.
- `guaranteed_income: np.array` of `ints` of shape `(T, len(k_using))`
  The income guaranteed by the government for each year and cluster in case of catastrophe. Given in $[10^9\,\$]$.
- `crop_cal: np.array` of `floats` of shape `(num_crops, )`
  Calorie content of each crop. Given in $[10^{12}\,\text{kcal}/10^6\,\text{t}]$.
- `max_areas: np.array` of `floats` of shape `(len(k_using), )`
  Upper limit of area available for agricultural cultivation in each cluster. Given in $[10^6\,\text{ha}]$.
- `rhoF: float`
  Penalty for shortcomings of the food demand as used in the final model run (either calculate according to the demanded probability or directly given through the settings).
- `rhoS: float`
  Penalty for insolvency of the government fund as used in the final model run (either calculate according to the demanded probability or directly given through the settings).

- `AddInfo_CalcParameters: dict`
  Additional information gather when calculating the guaranteed income and the penalties.
  - `expceted_incomes: np.array` of `floats` of shape `(len(k_using), )`
    Expected income of farmers in each cluster in a scenario excluding the government, calculated by solving the model for the corresponding scenario (`probF = 0.99` and `probS = 0.95` in the year before `sim_start`). Used to calculated the `guaranteed_income`. Given in $[10^9\,\$]$.
  - `probF_onlyF: float`
    The probability for food security when including only the food security constraint.
  - `probS_onlyS: float`
    The probability for solvency when including only the solvency constraint.
  - `import_onlyF: float`
    Average of yearly necessary import (averaged over all samples) when including only the food security constraint.
  - `debt_onlyS: float`
    Average necessary debt over all samples when only including the solvency constraint.

- `yield_information: dict`
  Additional information on the yield distributions for the considered settings.
  - `slopes: np.array` of `floats` of shape `(num_crops, len(k_using))`
    Slope coefficients of the linear yield trends per crop and cluster.
  - `constants: np.array` of `floats` of shape `(num_crops, len(k_using))`
    Constant term of the linear yield trends per crop and cluster.
  - `yld_means: np.array` of `floats` of shape `(T, num_crops, len(k_using))`
    Means of the yield distributions per year, crop, and cluster. Given in [t/ha].
  - `residual_stds: np.array` of `floats` of shape `(num_crops, len(k_using))`
    Standard deviation of the residuals when subtracting yield trends from yield observations; per crop and cluster.
  - `prob_cat_year: float`
    Probability for a year to be catastrophic given the covered risk level and number of considered clusters.
  - `share_no_cat: float`
    Share of samples that don't have a catastrophe within the considered time-frame.
  - `y_profit: np.array` of `floats` of shape `(num_crops, len(k_using))`
    Minimal yield per crop and cluster needed to not have losses (currently the same for each cluster). Given in [t/ha].
  - `share_rice_np: float`
    Share of cases where rice yields are too low to provide profit.
  - `share_maize_np: float`
    Share of cases where maize yields are too low to provide profit.
  - `exp_profit: np.array` of `floats` of shape `(T, num_crops, len(k_using))`
    Expected profits per year, crop, cluster. Given in $[10^9\,\$/10^6\,\text{ha}]$.
- `population_information: dict`
  Additional information on the population for the considered settings.
  - `population: np.array` of `floats` of shape `(T, )`
    Population in the considered area in each simulated year, according to given population scenario.
  - `total_pop_scen: np.array` of `floats` of shape `(T, )`
    Estimates of population in West Africa from simulation start to end for given population scenario.
  - `pop_cluster_ratio2015: np.array` of `floats` of shape `(len(k_using), )`
    Ratio between estimated population in West Africa and estimated population in each considered cluster in 2015.
- `status: int`
  Output-status of the solver `GurobiPy` (optimal: 2)
- `crop_alloc: np.array` of `floats` of shape `(T, num_crops, len(k_using))`
  The optimal crop areas for all years, crops and clusters based on the current model inputs. Given in $[10^6\,\text{ha}]$. Order of crops: rice, maize.
- `meta_sol: dict`
  Additional information based on the objective function evaluated for the resulting `crop_alloc`.
  - `exp_tot_costs: float`
    Final value of objective function, i.e. sum of cultivation and penalty costs. Given in $[10^9\,\$]$.
  - `fix_costs: np.array` of `floats` of shape `(N, )`
    Cultivation costs for each yield sample (depends only on the final year of simulation for each sample, see `terminal_years`). Given in $[10^9\,\$]$.

- `yearly_fixed_costs: np.array` of `floats` of shape `(N, T, len(k_using))`
  Total cultivation costs in each cluster for each year and each sample. Given in $[10^9\,\$]$.
- `fd_penalty: np.array` of `floats` of shape `(N, T)`
  Penalty payed because of food shortages for each year and sample. Given in $[10^9\,\$]$.
- `avg_fd_penalty: np.array` of `floats` of shape `(T, )`
  Average penalty payed because of food shortages in each year over all samples. Given in $[10^9\,\$]$.
- `sol_penalty: np.array` of `floats` of shape `(N, )`
  Penalty payed because of insolvency in each sample. Given in $[10^9\,\$]$.
- `shortcomings: np.array` of `floats` of shape `(N, T)`
  Shortcoming of the food demand for each year in each sample. Cases with surplus production are reported with zero shortcomings. Given in $[10^{12}\,\text{kcal}]$.
- `exp_shortcomings: np.array` of `floats` of shape `(T, )`
  Average shortcoming of the food demand in each year. Given in $[10^{12}\,\text{kcal}]$.
- `expected_incomes: np.array` of `floats` of shape `(T, len(k_using))`
  Average profits of farmers for each year in each cluster. Given in $[10^9\,\$]$.
- `profits: np.array` of `floats` of shape `(N, T, len(k_using))`
  Profits (or losses) of farmers per cluster and year for each sample. Given in $[10^9\,\$]$.
- `num_years_with_losses: int`
  Number of occurrences where farmers of a cluster have negative profits.
- `payouts: np.array` of `floats` of shape `(N, T, len(k_using))`
  Payouts from the government to farmers in case of catastrophe for each sample, year and cluster. Given in $[10^9\,\$]$.
- `final_fund: np.array` of `floats` of shape `(N, )`
  The fund size after final year of simulation (i.e. after payouts in case of a catastrophe, or after `T` years without catastroohe) for each sample. Given in $[10^9\,\$]$.
- `probF: float`
  Resulting probability for meeting the food demand.
- `probS: float`
  Resulting probability for solvency of the government fund at the end of the simulation (i.e. after payouts in case of a catastrophe, or after `T` years without catastroohe).
- `avg_nec_import: float`
  Average additional import (first over all samples and then over all years) that is necessary to cover food demand in final model run. If the model input `args['import']` is positive this is additional import. Given in $[10^{12}\,\text{kcal}]$.
- `avg_nec_debt: float`
  Average debt that is necessary to cover payouts in the final model run. Given in $[10^9\,\$]$.

- `crop_alloc_vss: np.array` of `floats` of shape `(T, num_crops, len(k_using))`
  Optimal crop areas in a deterministic case assuming average yield values.
- `meta_sol_vss: dict`
  Additional information based on the objective function evaluated for the resulting `crop_alloc_vss`.
- `VSS_value: float`
  The difference between total costs using deterministic solution for crop allocation and stochastic solution for crop allocation.
- `validation_values: dict`
  - `sample_size: int`
    Sample size used for validation.

- `total_costs: float`
  Expected total costs (value of objective function) for the original sample size.
- `total_costs_val: float`
  Expected total costs (value of objective function) for the validation sample size.
- `fd_penalty: float`
  Average food demand penalties payed over the full time for the original sample size.
- `fd_penalty_val: float`
  Average food demand penalties payed over the full time for the validation sample size.
- `sol_penalty: float`
  Average solvency penalties payed for the original sample size.
- `sol_penalty_val: float`
  Average solvency penalties payed for the validation sample size.
- `total_penalties: float`
  Average total penalties payed for the original sample size.
- `total_penalties_val: float`
  Average total penalties payed for the original sample size.
- `deviation_penalties: float`
  1 − `total_penalties`/`total_penalties_val`.

• `durations: dict`
Information on the duration of running the model framework.

- `GetExpectedIncome: float`
  Time spent to calculate the expected income.
- `GetPenalties: float`
  Time spent to calculate the correct penalties.
- `MainModelRun: float`
  Time spent on the main model run.
- `VSS: float`
  Time spent on calculating the Value of Stochastic Solution.
- `Validation: float`
  Time spent for validation with a higher sample size.
- `TotalTime: float`
  Total time spent to run the full model framework.

• `fn: str`
All relevant settings combined to a single file name to save/load results.

2 Additional explanations

The `settings` correspond to the parameters that are set by the user when calling `FoodSecurityProblem`. Some additional technical parameters can be set:

- `console_output: boolean` or `None`
  Specifying whether the progress should be documented through console outputs. If `None` the default as defined in `ModelCode/GeneralSettings` is used. The default is `None`.
- `logs_on: boolean` or `None`
  Specifying whether the progress should be documented in a log file. If `None` the default as defined in `ModelCode/GeneralSettings` is used. The default is `None`.
- `save: boolean`
  Whether the model outputs should be saved. The default is True.

- `plotTitle: str` or `None`
  If a `str` is given, the resulting crop allocations over time are plotted using `plotTitle` for the plot title (adding information on the considered clusters) and the figure is saved. The default is `None`.
- `close_plots: boolean`
  Whether plots should be closed directly after plotting. If `None` the default as defined in `ModelCode/GeneralSettings` is used. The default is `None`.
- `panda_file: str`
  Filename of the csv that should be used to save additional model results. The default is `'current_panda'`.

In `AddInfo_CalcParameters` information that were gathered while calculating the expected income and penalties but are not needed within `ModelCore` are saved.

The arguments for the `ModelCore` function are given by `args`. As some of the parameters needed within `ModelCore` need to be directly set by the user, some parameters given in `settings` are repeated in `args`.

Some other parameters seem to be repeated as well, while actually giving slightly different information:

- `rhoS/rhoF:` In `args` these are the penalty values used in the final model run, while in `settings` these can be either the final penalties if `PenMet` is `'penalties'`, or an initial guess for finding the correct penalties or `None` if `PenMet` is `'prob'`.
- `import:` The parameter `args['import']` is externally set import that will be used to reduce the food demand correspondingly. `AddInfo_CalcParameters['import_onlyF']` result from calculating the correct penalties and how much (additional) import is needed on average to meet the food demand when only looking at the food security constraint. `meta_sol['avg_nec_import']` is the average additional amount of food that needs to be import (in addition to `args['import']`) to meet the food demand in the final model run (i.e. including the solvency constraint). This can be higher than `AddInfo_CalcParameters ['necessary_import']` if there is a trade-off between the food demand constraint and the solvency constraint.
- `debt:` `AddInfo_CalcParameters['debt_onlyS']` refers to the average necessary debt to provide payouts when only including the solvency constraint (i.e. `rhoF = 0`). `meta_sol['avg_nec_debt']` refers to the avergae necessary debt to provide payouts in the final model run (i.e. including the correct `rhoF`).
- `probF/probS:` While `settings['probF']` and `args['probF']` are the same, as well as `settings['probS']` and `args['probS']`, in `meta_sol` the probabilities resulting in the final model run are given, which can differ from the probabilities given as input to the model framework in case of trade-offs between the food security constraint and the solvency constraint (and due to the limited accuracy of the penalty calculation).
- `expected_incomes:` `AddInfo_CalcParameters['expected_incomes']` refers to the expected income per cluster in a scenario without government in the year before `sim_start` and is used to calculated the guaranteed income. `AddInfo_CalcParameters['expected_incomes']` refers to the average income per cluster and year in the final model run.

3 Outputs to csv table

Settings:

- Penalty method
- Input probability food security
- Input probability solvency
- Number of crops

- Number of clusters
- Used clusters
- Yield projection
- Simulation start
- Population scenario
- Risk level covered
- Tax rate
- Share of income that is guaranteed
- Initial fund size
- Sample size
- Sample size for validation
- Number of covered years

Resulting penalties and probabilities:

- Penalty for food shortage
- Penalty for insolvency
- Resulting probability for food security
- Resulting probability for solvency
- Max. possible probability for food security (excluding solvency constraint)
- Max. possible probability for solvency (excluding food security constraint)

Yield information:

- Probability for a catastrophic year
- Share of samples with no catastrophe
- Share of years/clusters with unprofitable rice yields
- Share of years/clusters with unprofitable maize yields

Population information:

- Share of West Africa's population that is living in total considered region (2015)
- Share of West Africa's population that is living in considered clusters (2015)

Crop areas:

- On average cultivated area per cluster
- Average yearly total cultivated area
- Total cultivation costs

Food demand and needed import:

- Import (given as model input)
- Average food demand
- Food demand per capita
- Average total necessary import (over samples and then years)
- Average necessary add. import (over samples and then years)
- Average necessary add. import excluding solvency constraint (over samples and then years)
- Average necessary add. import per capita (over samples and then years)
- Average necessary add. import per capita (over samples and then years, only cases that need import)

Expected income, final fund and needed debt:

- Expected income (to calculate guaranteed income)
- Number of occurrences per cluster where farmers make losses
- Number of samples with negative final fund
- Average final fund (over all samples)
- Average income per cluster in final run (over samples and then years)
- Average income per cluster in final run scaled with capita (over samples and then years)
- Aggregated average government payouts per cluster (over samples)
- Average necessary debt (excluding food security constraint)
- Average necessary debt
- Average necessary debt (over all samples with negative final fund)
- Average necessary debt per capita (over all samples with negative final fund)
- Average necessary debt per capita (over all samples)

Different cost items in objective function:

- Average food demand penalty (over samples and then years)
- Average solvency penalty (over samples)
- Average total cultivation costs
- Expected total costs

VSS:

- Value of stochastic solution
- VSS as share of total costs (sto. solution)
- VSS as share of total costs (det. solution)
- Resulting probability for food security for VSS
- Resulting probability for solvency for VSS

Technincal variables:

- Validation value (deviation of total penalty costs)
- Seed (for yield generation)
- Filename for full results