



# Python Package for Agro-ecological zoning

Strengthening Asia-Pacific agriculture practices with Agro-Ecological Zonation

## User Guide for PyAEZ (v2.0.0)

Published by
the Food and Agriculture Organization of the United Nations
and
Asian Institute of Technology

Required citation:

FAO and AIT. 2022. *Python Package for Agro-ecological zoning. Strengthening Asia-Pacific agriculture practices with Agro-Ecological Zonation.* Rome. Full DOI link.

# Contents

Contents ..... 2
Introduction ..... 6
1.1 Background ..... 6
1.2 What is PyAEZ? ..... 7
Data requirement and preparation ..... 10
1.3 Python dependencies ..... 10
1.4 Data preparation ..... 10
1.4.1 Climate data ..... 10
1.4.2 Soil ..... 11
1.4.3 Geographical location and terrain data ..... 13
Module 1: Climate Regime ..... 14
1.5 Introduction ..... 14
1.6 Object Class Creation (Mandatory) ..... 15
1.7 Setting up Geographical and Terrain Data (Mandatory) ..... 15
1.8 Read the climate data and load into the Class (Mandatory) ..... 15
1.9 Setting Study Area Inputs (Optional) ..... 16
1.10 Thermal Climate ..... 16
1.11 Thermal zones ..... 18
1.12 Thermal Length of Growing Period ($LGP_t$) ..... 19
1.13 Temperature summations (TS) ..... 19
1.14 Temperature profiles ..... 20
1.15 Length of Growing Period (LGP) ..... 21
1.15.1 Reference evapotranspiration (ETo) ..... 21
1.15.2 Maximum evapotranspiration (ETm) ..... 21
1.15.3 Actual evapotranspiration (ETa) ..... 21
1.15.4 LGP calculation ..... 23
1.15.5 LGP Equivalent ..... 23
1.16 Multiple cropping zones classification ..... 25
1.17 Fallow period requirements ..... 26
1.18 Permafrost evaluation ..... 26
1.19 Agro-ecological Zones classification ..... 27

Module 2: Crop Simulation ..... 28
1.20 Introduction ..... 28
1.21 Setting up inputs for Module 2 ..... 30
1.21.1 Geographical and terrain input ..... 30
1.21.2 Climate data input ..... 30
1.21.3 Setting Study Area Inputs (Optional) ..... 31
1.21.4 Crop parameters input ..... 31
1.21.5 Soil water parameter input ..... 31
1.21.6 Crop cycle parameter input ..... 32
1.21.7 Thermal screening input ..... 32
1.21.8 Adjustment for Perennial Crop (optional) ..... 34
1.22 Calculations and outputs ..... 34
1.22.1 Crop cycle simulation ..... 34
1.22.2 Estimated Maximum Yield ..... 35
1.22.3 Optimum Crop Calendar ..... 35
Module 3: Climate Constraints ..... 37
1.23 Introduction ..... 37
1.24 Setting up parameter files ..... 38
1.25 Applying climate constraints ..... 39
Module 4: Soil Constraints ..... 40
1.26 Introduction ..... 40
1.27 Setting up parameter files ..... 41
1.28 Calculate soil qualities ..... 45
1.29 Calculate soil ratings ..... 46
1.30 Extracting soil qualities ..... 46
1.31 Extracting soil ratings ..... 46
1.32 Applying soil constraints ..... 47
Module 5: Terrain Constraints ..... 48
1.33 Introduction ..... 48
1.34 Setting up parameter files ..... 48
1.35 Setting up inputs ..... 49
1.35.1 Climate and terrain inputs ..... 49

1.35.2 Calculate Fournier Index ..... 49
1.35.3 Extract Fournier Index ..... 50
1.36 Applying terrain constraints ..... 50
Module 6: Economic Suitability Analysis ..... 51
1.37 Introduction ..... 51
1.38 Crop parameters inputs ..... 51
1.39 Net Revenue ..... 52
1.40 Classified Net Revenue ..... 52
1.41 Normalized Net Revenue ..... 53
Utility Calculations ..... 54
1.42 Introduction ..... 54
1.43 Monthly-to-daily interpolation ..... 54
1.44 Daily-to-monthly aggregation ..... 55
1.45 Create latitude map ..... 55
1.46 Classify the final crop yield ..... 56
1.47 Saving GeoTIFF rasters ..... 56
1.48 Averaging raster files ..... 57
1.49 Calculate wind speed at 2m altitude ..... 57
References ..... 58

# List of Figures

Figure 1.Overview of PyAEZ workflow....9
Figure 2. Example of the Topsoil Characteristic input format (CSV file). ....12
Figure 3. Overview of Module 1 (Climate Regime) workflow. ....14
Figure 4. Overview of Module 2 (Crop Simulation) workflow. ....29
Figure 5. Overview of Crop Simulation routine....29
Figure 6. Estimation of Maximum Attainable Yield (rainfed and irrigated) and Optimum Starting Date (Module 2). ....36
Figure 7. Overview of Module 3 (Climate Constraints) workflow. ....38
Figure 8. Overview of Module 4 (Soil Constraints) workflow. ....41
Figure 9. Overview of Module 5 (Terrain Constraints) workflow. ....49
Figure 10. Overview of Module 6 (Economic Suitability) workflow. ....51

# List of Tables

Table 1. Input Climatic parameters. ....11
Table 2. Soil Data preparation....11
Table 3. Input Soil parameters of topsoil and sub-soil properties. Please pay special attention to the abbreviations when used in the CSV file as PyAEZ reads the data using these headers. ....12
Table 4. Geographical and terrain data preparation. ....13
Table 5. Classification of Thermal Climate classes according to rainfall and temperature seasonality....17
Table 6. Classification of Thermal Zone classes according to rainfall and temperature seasonality....18
Table 7. Temperature Profile classes ....20
Table 8. Kc values used in Module 1 for the calculation of the maximum evapotranspiration (Etm)....24
Table 9. Delineation of multiple cropping zones. ....25
Table 10. Net Revenue Classification....53
Table 11. Yield suitability classification. ....56

# Introduction

The world population is expected to reach 8.6 billion in 2030 and 9.8 billion in 2050 (UNDESA, 2017). With this eruptive growth in population, an unprecedented increase in demand for food, feed, and fuel is expected, while the agricultural land needed for production continues to shrink in many parts of the world.

The accelerating pace of climate change, combined with global population growth, threatens food security globally. Higher temperatures eventually reduce yields of desirable crops while encouraging weed and pest proliferation and changes in precipitation patterns increase the likelihood of short-run crop failures and long-run production declines (Nelson et al., 2009).

Yield increases on existing croplands will, therefore, be an essential component to increase food production (Ray et al., 2013). To this end, Agro-Ecological Zoning (AEZ) framework was developed as a tool to analyse the effect of climate on land use and agricultures, as well as helping to optimise the crop cycle to produce the best yield possible. PyAEZ is an open-source Python package which offers AEZ calculations for user to implement for their regional AEZ analyses. This technical document contains detailed descriptions of all the AEZ modules and functions in PyAEZ.

## 1.1 Background

Over the last thirty years, FAO and the International Institute for Applied Systems Analysis (IIASA) have been developing Agro-Ecological Zoning (AEZ). AEZ is a modelling system for land evaluation to support sustainable land use planning, stimulate agricultural investments, monitor the status of agricultural resources, and assess the impacts of climate change on agriculture.

The Agro-Ecological Zoning (AEZ) approach, developed by FAO jointly with IIASA, is based on the principles of land evaluation and defines matching procedures to identify crop-specific limitations of prevailing climate, soil, and terrain resources with simple and robust crop models, under assumed levels of inputs and management conditions. Main outputs are maximum potential and agronomically attainable crop yields and suitability levels for basic land resources units under different agricultural production systems defined by water supply systems and levels of inputs and management circumstances.

While most countries have adopted land evaluation, land suitability assessment, agro-ecological zoning in the past to prepare agricultural investment plans, most of those are outdated. In parallel, various generations of “AEZ Projects” have served as vehicles to consolidate efforts, structure project goals, and promote funding and resources for the further development of these concepts. And while new datasets and technologies are

increasingly becoming available, national capacities to develop, update and use agro-ecological zoning remain limited.

The GAEZ assessment, currently at its fourth update (GAEZ v4), uses seven different modules that are run sequentially to generate agro-climatic and crop-specific information. Each module is made up of a series of FORTRAN routines, documented in GAEZ version 4 model documentation (Fischer et al., 2021), that are run through custom batch scripts. Additional scripts in Delphi are used for specific modelling (to be clearly specified by IIASA). Data preparation, although fully documented on the theoretical concepts, is mostly undocumented when addressing the required process to input new data in the modelling system. Limited knowledge on data preparation and lack of a systematic system to run FORTRAN routines make the capacity to generate outputs, limited to a restricted number of experts at IIASA.

The main strategic shift of GAEZ v5 is to focus on how the strong scientific basis for GAEZ can be made available to national entities to support decision-making. GAEZ v5 will focus on taking the last three decades of knowledge and building a standardized, repeatable, accessible yet extensible approach for countries to implement their own, fit for purpose, nationally-adjusted Agro-Ecological Zoning project(s). Countries need guidance on how to collect relevant local data, what tools they can use to create data if it does not exist, how to engage with farmers, local representatives and other stakeholders, and how to process, manage, host and disseminate results of an analysis

With a growing need to address country-specific agro-ecological zoning modelling, the "Strengthening Agro-climatic Monitoring and Information System (SAMIS)"[1] project in Lao PDR, in collaboration with the Geomatics Unit of the Asian Institute of Technology (AIT) developed a first prototype in Python language, named PyAEZ, that generate national AEZ information. The code, with supporting documentation, and training material is publicly available in the GitHub repository at the https://github.com/gicait/PyAEZ

## 1.2 What is PyAEZ?

PyAEZ is the first step of GAEZ expansion that utilizes Python scripts to develop users'AEZ projects. The PyAEZ package utilizes climate, soil, and terrain conditions relevant to agricultural production and suitability using crop-specific land resource inventory parameters.

The package is developed with several Python routines and is operated with Jupyter Notebooks, which means it has the capability to be uploaded onto Google Colab, an online Jupyter Notebook system. This compatibility with an online platform such as Google Collab

[1] Further information on the SAMIS project can be found on the FAO page: http://www.fao.org/in-action/samis/en/

allowed the development team to host two virtual hands-on trainings where attendants were guided through the scientific concepts of AEZ as well as executing the scripts with country-specific input data, through Google Colab.

PyAEZ has been developed to be used within the tropical region, hence some of the complexity of GAEZ in non-tropical regions (e.g., vernalization requirements, permafrost evaluation) is not accounted for. Moreover, the system has not been tested on larger areas where performances of results may be an issue.

PyAEZ package consists of 6 main AEZ modules and 1 additional utility module (Figure 1):

- **Module 1: Climate Regime** – calculation of agro-climatic indicators for evaluation of climatic suitability of crops
- **Module 2: Crop Simulation** – simulate an optimal crop cycle for the highest attainable yield
- **Module 3: Climate Constraints** – application of agro-climatic constraints to the calculated yield of a particular crop
- **Module 4: Soil Constraints** – application of edaphic constraints to the calculated yield of a particular crop
- **Module 5: Terrain Constraints** – application of terrain constraints to the calculated yield of a particular crop
- **Module 6: Economic Suitability Analysis** – evaluation of economic profitability of a crop based on crop price and the calculated yield
- **Utilities Calculations** – miscellaneous calculation routines used throughout the 6 main AEZ modules.

The package is also equipped with additional calculation routines for:

- Water Balance Calculation and applying of yield reduction factors based on water limitation (FAO CropWat algorithm) (Smith, 1992)
- Biomass Calculation produced by Photosynthesis activities of plants under given radiation conditions.
- Reference Evapotranspiration Calculation using Penman-Monteith algorithm (Allen et al., 1998; Monteith, 1965, 1981)

This documentation provides a step-by-step guideline for anyone looking to develop an AEZ project using PyAEZ package, starting from the installation to the description of the functions in each module, as well as the theoretical concepts behind each function/module.

The code, with supporting documentation, and training material (Jupyter Notebooks and example data) is publicly available and can be downloaded and installed through:

- PyAEZ GitHub repository (https://github.com/gicait/PyAEZ)
- Python package management systems '*pip*' and '*conda*'

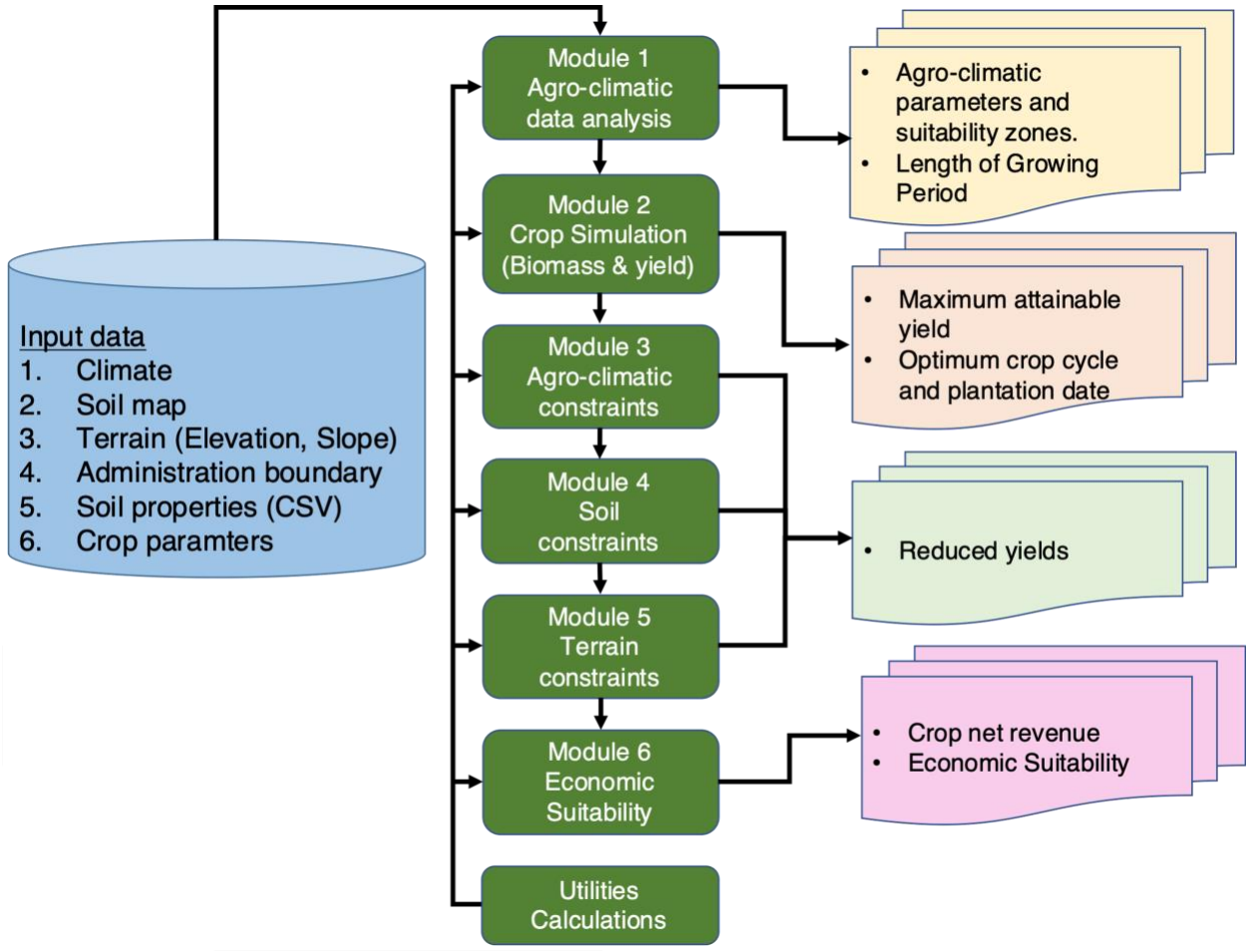


Figure 1.Overview of PyAEZ workflow.

# Data requirement and preparation

This section will cover all the system and data requirements to run PyAEZ. These subsections also act as an essential checklist for the necessary elements to every PyAEZ project initiation.

## 1.3 Python dependencies

PyAEZ package requires the following additional open-source Python packages to be installed and imported for the AEZ calculations to work:

1. NumPy[2] – NumPy array is the format used throughout PyAEZ for pixel-based calculation
2. GDAL[3] – allow the package to utilize and generate geo-referenced output from non-geocoded NumPy arrays
3. SciPy[4] – offers statistical analyses and is interoperable with NumPy array
4. Pandas[5] – allows PyAEZ to read MS Excel sheets with user-defined parameters
5. Numba[6] – NumPy-aware optimizing compiler used to speed up some computationally heavy routines within PyAEZ

## 1.4 Data preparation

Input-data preparation is essential as the current version of PyAEZ requires users to input data of specific format and shape. Depending on the nature of each aspect, one might prepare to transform into 2D or 3D NumPy arrays, in other cases, preparing additional excel sheet information will be required.

### 1.4.1 Climate data

PyAEZ requires 6 climatic parameters (Table 1) to be prepared as 3D NumPy data cube for a single year (row, column, day-of-year). We encourage users to use daily climatic data for more accurate results. If the monthly climate data is used, it will need to be interpolated to daily data. The input climate data can be Historical-type data or Future-projected data. Example of the possible climate data sources are Corpernicus' Climate Data Store[7], European Centre for Medium-Range Weather Forecasts (ECMWF)[8], Google Earth Engine (GEE)[9], and etc. Users can also utilize own country data from their national sensor network/database

[2] NumPy: https://numpy.org/install
[3] GDAL: e.g. https://anaconda.org/conda-forge/gdal
[4] SciPy: https://scipy.org/install
[5] Pandas: https://pandas.pydata.org/docs/getting_started/install.html
[6] Numba: https://numba.readthedocs.io/en/stable/user/installing.html
[7] Corpernicus' Climate Data Store: https://cds.climate.copernicus.eu/
[8] ECMWF: https://www.ecmwf.int/en/forecasts/datasets
[9] Google Earth Engine: https://developers.google.com/earth-engine/datasets

Table 1. Input Climatic parameters.

| Climatic parameter | Data frequency | Unit | Data format |
|---|---|---|---|
| Minimum air temperature (2m above surface) | Daily or monthly | Degree Celsius | 3D NumPy (row, column, time) |
| Maximum air temperature (2m above surface) | Daily or monthly | Degree Celsius | 3D NumPy (row, column, time) |
| Total precipitation | Daily or monthly | mm/day | 3D NumPy (row, column, time) |
| Solar radiation | Daily or monthly | $W/m^2$ | 3D NumPy (row, column, time) |
| Relative humidity | Daily or monthly | Percentage | 3D NumPy (row, column, time) |
| Windspeed (2m above surface) | Daily or monthly | m/s | 3D NumPy (row, column, time) |

During the preparation of climatic data, all NaN values (different climate data tend to have some specified no-value values, e.g., -9999) need to be set to zero to prevent any incomputable errors further down the line.

### 1.4.2 Soil

PyAEZ requires two soil-related data preparations, as follows:

Table 2. Soil Data preparation

| Data | Data source | Data format |
|---|---|---|
| Soil map | • Harmonized World Soil Database (HWSD)[10]<br>• Own local/regional soil map | • 2D NumPy array<br>• Each pixel refers to a unique soil mapping unit |
| Soil characteristics (Table 3) | • Corresponding to the soil map | • CSV file with each soil characteristic parameters as the column headers (Figure 2).<br>• PyAEZ needs 2 CSV files, one for topsoil and another for sub-soil. |

[10] HWSD: https://www.fao.org/soils-portal/data-hub/soil-maps-and-databases/harmonized-world-soil-database-v12/en/

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | CODE | TXT | OC | pH | TEB | BS | CEC_soil | CEC_clay | RSD | SPR | SPH | OSD | DRG | ESP | EC | CCB | GYP | GRC | VSP |
| 2 | 4260 | Coarse | 0.3 | 5.7 | 2.8 | 46 | 5 | 16 | 100 | 0 | Lithic | 0 | MW | 2 | 0 | 0 | 0 | 10 | 0 |
| 3 | 4261 | Coarse | 0.3 | 5.7 | 2.8 | 46 | 5 | 16 | 100 | 0 | Lithic | 0 | MW | 2 | 0 | 0 | 0 | 10 | 0 |
| 4 | 4264 | Medium | 0.3 | 4.7 | 2 | 20 | 12 | 22 | 100 | 0 | Lithic | 0 | P | 1 | 0 | 0 | 0 | 7 | 0 |
| 5 | 4265 | Coarse | 0.38 | 4.8 | 2.3 | 31 | 6 | 17 | 100 | 0 | Lithic | 0 | P | 4 | 0 | 0 | 0 | 8 | 0 |
| 6 | 4267 | Medium | 0.42 | 4.8 | 2.3 | 32 | 6 | 14 | 100 | 0 | Lithic | 0 | MW | 2 | 0 | 0 | 0 | 10 | 0 |
| 7 | 4284 | Medium | 0.42 | 4.8 | 2.3 | 32 | 6 | 14 | 100 | 0 | Lithic | 0 | I | 2 | 0 | 0 | 0 | 10 | 0 |
| 8 | 4325 | Fine | 0.45 | 5.9 | 17.1 | 84 | 23 | 39 | 100 | 0 | Lithic | 0 | VP | 2 | 0 | 0 | 0 | 1 | 0 |
| 9 | 4383 | Medium | 0.45 | 5.9 | 17.1 | 84 | 23 | 39 | 10 | 0 | Lithic | 0 | I | 2 | 0 | 0 | 0 | 1 | 0 |
| 10 | 4408 | Fine | 0.6 | 6.8 | 31.3 | 100 | 20 | 39 | 100 | 0 | Lithic | 0 | P | 2 | 0 | 0 | 0 | 8 | 0 |
| 11 | 4452 | Coarse | 0.3 | 5.7 | 2.8 | 46 | 5 | 16 | 100 | 0 | Lithic | 0 | MW | 2 | 0 | 0 | 0 | 10 | 0 |
| 12 | 4499 | Fine | 0.77 | 4.6 | 17.1 | 46 | 21 | 36 | 100 | 0 | Lithic | 0 | VP | 1 | 0 | 0 | 0 | 1 | 0 |
| 13 | 4544 | Fine | 0.52 | 5.5 | 4.7 | 45 | 10 | 13 | 100 | 0 | Lithic | 0 | MW | 2 | 0 | 0 | 0 | 2 | 0 |
| 14 | 4587 | Fine | 0.63 | 7.4 | 44.5 | 100 | 43 | 71 | 100 | 0 | Lithic | 0 | P | 2 | 0 | 3.4 | 0 | 4 | 0 |
| 15 | 6651 | Fine | 0.46 | 5.7 | 8 | 65 | 13 | 15 | 100 | 0 | Lithic | 0 | MW | 1 | 0 | 0 | 0 | 8 | 0 |
| 16 | 11788 | Fine | 0.52 | 4.9 | 2.1 | 21 | 7 | 11 | 100 | 0 | Lithic | 0 | MW | 1 | 0 | 0 | 0 | 28 | 0 |

Figure 2. Example of the Topsoil Characteristic input format (CSV file).

Table 3. Input Soil parameters of topsoil and sub-soil properties. Please pay special attention to the abbreviations when used in the CSV file as PyAEZ reads the data using these headers.

| Abbreviation | Parameter name | Data type |
|---|---|---|
| CODE | Soil Mapping Unit ID[11] | Numerical |
| TXT | Soil texture | String |
| OC | Soil organic carbon | Numerical |
| pH | Soil pH (0-14) | Numerical |
| TEB | Total exchangeable bases | Numerical |
| BS | Base saturation | Numerical |
| CEC_soil | Cation exchange capacity of soil | Numerical |
| CEC_clay | Cation exchange capacity of clay | Numerical |
| RSD | Effective soil depth | Numerical |
| GRC | Soil coarse material (Gravel) percentage | Numerical |
| DRG | Drainage classes (VP: very poor, P: poor, I: imperfectly, MW: moderately well, W: well, SE: somewhat excessive, E: excessive | String |
| ESP | Exchangeable sodium percentage | Numerical |
| EC | Electricity conductivity [dS/m] | Numerical |
| SPH | Soil phase rating (0 or 1) | Numerical |
| SPR | Soil property rating (0 or 1) | Numerical |
| OSD | Other soil depth/volume related characteristics rating | Numerical |
| CCB | Calcium carbonate content percentage | Numerical |
| GYP | Gypsum content percentage | Numerical |
| VSP | Vertical properties (0 or 1) | Numerical |

[11] Soil Mapping Unit ID as obtained from the soil map

### 1.4.3 Geographical location and terrain data

PyAEZ requires the elevation and slope maps to be prepared (Table 4). Administrative boundary mask is optional, however, is highly encouraged because it can help minimizing the computational time by considering only area/region of interest.

Table 4. Geographical and terrain data preparation.

| Data | Data source | Data format |
|---|---|---|
| Elevation map | • Global elevation map, or<br>• Own national/regional data | • 2D NumPy array<br>• Unit: metre |
| Terrain slope map | • Global slope map, or<br>• Own national/regional data | • 2D NumPy array<br>• Unit: percentage |
| Administrative boundary mask (optional) | • Global mask, or<br>• Own national/regional mask | • 2D NumPy array<br>• Binary: 1 for wanted area, and 0 for unwanted area |

# Module 1: Climate Regime

## 1.5 Introduction

This module performs climate data analysis and compiling general agro-climatic indicators. These general agro-climatic indicators summarize climatic profiles in the study area for each grid. Figure 3 shows the overall workflow of Module 1. The key input data for this module is the climatic data, and the geographical and terrain data. This section offers descriptions of the all the functions within Module 1, with example code snippets.

It is advisable to always run this module first, as several agro-climatic indicators output from Module 1 will get feed into Module 2 (Crop Simulation).

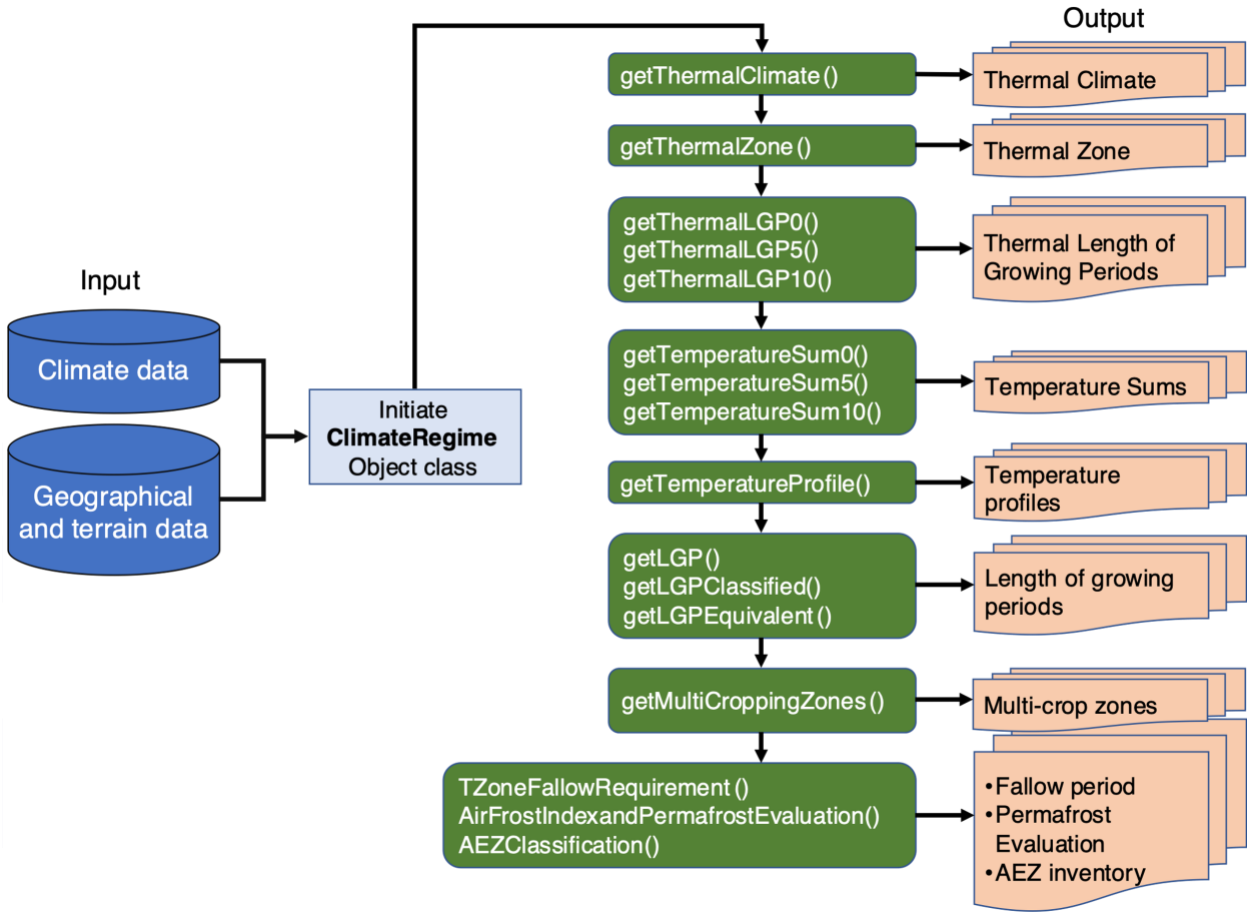


Figure 3. Overview of Module 1 (Climate Regime) workflow.

## 1.6 Object Class Creation (Mandatory)

PyAEZ codes utilizes '*Object-Oriented Programming*' style, meaning that each module has its own *Classes* containing separate attributes and functions. Therefore, it is essential that the necessary object-classes are initiated at the beginning of each module.

For Module 1, the Class that we need is called '`ClimateRegime`', and is imported and initiated as:

```
# Import PyAEZ Module1:ClimateRegime object class
import ClimateRegime
clim_reg = ClimateRegime.ClimateRegime()
```

## 1.7 Setting up Geographical and Terrain Data (Mandatory)

The next mandatory step after object class creation is to input user's elevation and geographic latitude information into the object class by using this function.

```
# Load geographical location and elevation data into the object class
clim_reg.setLocationTerrainData(lat_min, lat_max, elevation)
```

| **Function Arguments** | |
|---|---|
| `lat_min` | a single value corresponding to the minimum latitude (decimal degrees) of the study area |
| `lat_max` | a single value corresponding to the maximum latitude (decimal degrees) of the study area |
| `elevation` | 2D NumPy array, elevation of the study area in metres |
| **Function Returns** | |
| None | |

## 1.8 Read the climate data and load into the Class (Mandatory)

The third and final mandatory step of preparation is to incorporate all the required climatic datasets into the object class. Depending on the temporal dimension of climatic datasets, user can use either one of the following functions: one for daily datasets and the other for monthly.

```
# Load climate data from NPY files
min_temp = np.load('PATH_TO_FILE') # Continue loading the rest of climate data
# Deal with NaN or inappropriate negative values, for example:
rel_humidity[rel_humidity<0] = 0
short_rad[short_rad<0]=0
wind_speed[wind_speed<0]=0
# Use the line below if MONTHLY data are used
clim_reg.setMonthlyClimateData(min_temp, max_temp, precipitation, short_rad,
wind_speed, rel_humidity)
# Use the line below if DAILY climate data are used
clim_reg.setDailyClimateData(min_temp, max_temp, precipitation, short_rad,
wind_speed, rel_humidity)
```

| Function Arguments | |
|---|---|
| min_temp | 3D NumPy array, daily or monthly minimum temperature (°C) |
| max_temp | 3D NumPy array, daily or monthly maximum temperature (°C) |
| precipitation | 3D NumPy array, daily or monthly total precipitation (mm/day) |
| short_rad | 3D NumPy array, daily or monthly solar radiation ($W/m^2$) |
| wind_speed | 3D NumPy array, daily or monthly windspeed at 2m elevation (m/s) |
| rel_humidity | 3D NumPy array, daily or monthly relative humidity (percentage decimal, 0-1) |
| **Function Returns** | |
| None | |

## 1.9 Setting Study Area Inputs (Optional)

This function is set up as an optional step which set up the mask layer as input which reduces the computation time outside the pixels of considerations.

```
1 # Set up mask for the study area (country, regional, or local)
2 clim_reg.setStudyAreaMask(admin_mask, no_data_value=0)
```

| Function Arguments | |
|---|---|
| admin_mask | 2D NumPy array, extracted only region of interest (Binary 0/1) |
| no_data_value | A single value, pixels equal to this value will be omitted during calculation |
| **Function Returns** | |
| None | |

## 1.10 Thermal Climate

The Thermal Climate function calculates and classifies latitudinal thermal climate, which will be used later in Module 2 for the assessment of potential crops and land utilization types (LUT) presence in each grid cell. It is advisable to use an average of multiple years of temperature data (e.g., 30 years) rather than a single-year data, to obtain better representation of the climate for the study region.

Table 5 describes the classification of thermal climates based on (i) the monthly mean temperature (sea-level adjusted[12]), (ii) the ratios between summer/winter rainfall and the reference evapotranspiration (P/ETo), and (iii) the temperature amplitude as a measure of continentality (i.e., the difference between temperatures of warmest and coldest month) (Fischer et al., 2021).

```
1 # Classification of rainfall and temperature seasonality into thermal climate
2 classes
3 tclimate =clim_reg.getThermalClimate ()
```

[12] Sea-level adjusted monthly mean temperature with a fixed lapse rate of **0.55°C/100 metres** of elevation

| Function Arguments | |
|---|---|
| None | |
| **Function Returns** | |
| `tclimate` | 2D NumPy array (map) of Thermal Climate classification |

Table 5. Classification of Thermal Climate classes according to rainfall and temperature seasonality

| Pixel value | Climate | Rainfall and Temperature Seasonality | |
|---|---|---|---|
| **1** | **Tropics** All months with monthly mean sea-level adjusted temperatures > 18°C, and monthly temperature amplitude* < 15°C | **Tropical lowland** | Tropics with actual mean temperatures (Ta) above $20^0$C |
| **2** | | **Tropical highland** | Tropics with actual mean temperatures below $20^0$C |
| **3** | **Subtropics** One or more months with monthly mean temperatures, corrected to sea level, below $18^0$C, but all above $5^0$C, and 8-12 months above $10^0$C | **Low rainfall** | Annual rainfall less than 250 mm |
| **4** | | **Summer rainfall** | Northern hemisphere: P/ETo in April-September ≥ P/ETo in October-March. Southern hemisphere: P/ETo in October-March ≥ P/ETo in April-September |
| **5** | | **Winter rainfall** | Northern hemisphere: P/ETo in April-September ≤ P/ETo in October-March. Southern hemisphere: P/ETo in October-March ≤ P/ETo in April-September |
| **6** | **Temperate** At least one month with monthly mean temperatures, corrected to sea level, below $5^0$C and four or more months above $10^0$C | **Oceanic** | Seasonality less than $20^0$C** |
| **7** | | **Sub-continental** | Seasonality 20-$35^0$C ** |
| **8** | | **Continental** | Seasonality more than $35^0$C** |
| **9** | **Boreal** At least one month with monthly mean temperatures, corrected to sea level, below $5^0$C and 1-3 months above $10^0$C | **Oceanic** | Seasonality less than $20^0$C** |
| **10** | | **Sub-continental** | Seasonality 20-$35^0$C ** |
| **11** | | **Continental** | Seasonality more than $35^0$C** |
| **12** | **Arctic** | **Arctic** | All months with monthly mean temperatures, corrected to sea level, below 10°C |

*Monthly temperature amplitude = monthly maximum temperature – monthly minimum temperature
**Seasonality = the difference in mean temperature of the warmest and coldest month

## 1.11 Thermal zones

The thermal zone is classified based on actual temperature which reflects on the temperature regimes of major thermal climates (Table 6).

```
1  # Classification of thermal zone classes
2  tzone = clim_reg.getThermalZone()
```

| **Function Arguments** | |
|---|---|
| None | |
| **Function Returns** | |
| `tzone` | 2D NumPy array (map) of Thermal Zones classification |

Table 6. Classification of Thermal Zone classes according to rainfall and temperature seasonality

| **Pixel value** | **Climate** | **Thermal zones** | |
|---|---|---|---|
| **1** | **Tropics** All months with monthly mean sea-level adjusted temperatures > 18°C, and monthly temperature amplitude* < 15°C | **Warm** | Tropics with annual mean temperature above $20^0$C |
| **2** | | **Cool/Cold/ Very cold** | Tropics with annual mean temperatures below $20^0$C |
| **3** | **Subtropics** One or more months with monthly mean temperatures, corrected to sea level, below $18^0$C, but all above $5^0$C, and 8-12 months above $10^0$C | **Warm/Moderately cool** | Annual mean temperature above $20^0$C |
| **4** | | **Cool** | At least one month with monthly mean temperatures below $5^0$C and 4 or more months above $10^0$C |
| **5** | | **Cold** | At least one month with monthly mean temperatures below $5^0$C and 1-3 months above $10^0$C |
| **6** | | **Very cold** | All months with monthly mean temperatures below $10^0$C. |
| **7** | **Temperate** At least one month with monthly mean temperatures, corrected to sea level, below $5^0$C and four or more months above $10^0$C | **Cool** | At least one month with monthly mean temperatures below $5^0$C and 4 or more months above $10^0$C |
| **8** | | **Cold** | At least one month with monthly mean temperatures below $5^0$C and 1-3 months above $10^0$C |
| **9** | | **Very cold** | All months with monthly mean temperatures below $10^0$C. |
| **10** | **Boreal** At least one month with monthly mean temperatures, corrected to sea level, below $5^0$C and 1-3 months above $10^0$C | **Cold** | At least one month with monthly mean temperatures below $5^0$C and 1-3 months above $10^0$C |
| **11** | | **Very cold** | All months with monthly mean temperatures below $10^0$C. |
| **12** | **Arctic** | **Arctic** | All months with monthly mean temperatures, corrected to sea level, below 10°C |

*Monthly temperature amplitude = monthly maximum temperature – monthly minimum temperature

## 1.12 Thermal Length of Growing Period ($LGP_t$)

The thermal length of growing period ($LGP_t$) is defined as the number of days in a year during which the daily mean temperature (Ta) is conductive to crop growth and development. PyAEZ utilizes the AEZ three standard temperature thresholds for LGPt:

i. Periods with Ta>0°C ($LGP_{t0}$)
ii. Periods with Ta>5°C ($LGP_{t5}$) – the period conductive to plant growth and development
iii. Periods, and Ta>10°C ($LGP_{t10}$) – a proxy for the period of low risks for late and early frost occurrences and termed 'frost-free period'

```
# Calculate Thermal Length of Growing Period (LGPt)
# 3 temperature thresholds
# LGPt>0 degC
lgpt0 = clim_reg.getThermalLGP0()
# LGPt>5 degC
lgpt5 = clim_reg.getThermalLGP5()
# LGPt>10 degC
lgpt10 = clim_reg.getThermalLGP10()
```

| **Function Arguments** | |
|---|---|
| None | |
| **Function Returns** | |
| `lgpt0` | 2D NumPy arrays [days] |
| `lgpt5` | 2D NumPy arrays [days] |
| `lgpt10` | 2D NumPy arrays [days] |

## 1.13 Temperature summations (TS)

Temperature summation corresponds to the accumulated temperature which represent the crop-/LUT-specific heat requirements (Fischer et al., 2021).

Reference temperature sums (TS) are calculated for each grid-cell by accumulative daily average temperature (Ta) for days when Ta is above the thresholds as follows: (i) 0°C, (ii) 5°C, and (iii) and 10°C.

```
# Calculate temperature summation at 3 temperature thresholds
# Tsum>0 degC
tsum0 = clim_reg.getTemperatureSum0()
# Tsum>5 degC
tsum5 = clim_reg.getTemperatureSum5()
# Tsum>10 degC
tsum10 = clim_reg.TemperatureSum10()
```

| Function Arguments | |
|---|---|
| None | |
| **Function Returns** | |
| `tsum0` | 2D NumPy arrays [°C] |
| `tsum5` | 2D NumPy arrays [°C] |
| `tsum10` | 2D NumPy arrays [°C] |

## 1.14 Temperature profiles

Temperature profiles (Table 7) can be classified into 9 classes of different daily 'temperature ranges' between Ta<-5°C to Ta>30°C. This classification uses 5°C intervals as well as distinguishes the increasing and decreasing temperature trends within a year (Fischer et al., 2021). The output from this classification will be used in Module 2 (Crop Simulation), where these profiles are matched with crop-specific temperature profile requirements to assess the crop-growth suitability for any specific locations.

```
1 # Classification of temperature ranges for temperature profile
2 tprofile = clim_reg.getTemperatureProfile()
```

| Function Arguments | |
|---|---|
| None | |
| **Function Returns** | |
| `tprofile` | 18 2D NumPy arrays [A1-A9, B1-B9] correspond to each Temperature Profile class [days] |

Table 7. Temperature Profile classes

| **Mean daily temperature (°C)** | **Temperature trend** | |
|---|---|---|
| | Increasing | Decreasing |
| **>30** | A1 | B1 |
| **25 - 30** | A2 | B2 |
| **20 - 25** | A3 | B3 |
| **15 - 20** | A4 | B4 |
| **10 - 15** | A5 | B5 |
| **5 - 10** | A6 | B6 |
| **0 - 5** | A7 | B7 |
| **-5 - 0** | A8 | B8 |
| **<-5** | A9 | B9 |

## 1.15 Length of Growing Period (LGP)

The length of growing period (LGP) is defined as the number of days during the year when the temperature regime and moisture supply are conductive to crop growth and development. LGP, therefore, acts as an agro-climatic indicator of the potential productivity of an area of land.

### 1.15.1 Reference evapotranspiration (ETo)

The reference evapotranspiration (ETo) represents evapotranspiration from a defined reference surface, which closely resembles an extensive surface of green, well-watered grass of uniform height (12 cm), actively growing and completely shading the ground. GAEZ calculates ETo from the attributes in the climate database for each grid-cell according to the Penman-Monteith equation (Allen et al., 1998; Monteith, 1965, 1981; Pruitt & Doorenbos, 1977). A description of the implementation of the Penmann-Monteith equations is provided in Appendix 3-1 of Fischer et al., (2021).

### 1.15.2 Maximum evapotranspiration (ETm)

In Module 1, the calculation of maximum evapotranspiration (ETm) for a 'reference crop' assumes that sufficient water is available for uptake in the rooting zone. The value of ETm is related to ETo through applying crop coefficients for water requirement (Kc), reflecting phenological development and leaf area. The Kc values are crop- and climate-specific. They vary generally between 0.3-0.5 at initial crop stages (emergence) to 1.0-1.2 at reproductive stages. PyAEZ utilizes the 'reference crop' whose Kc values depend on the thermal characteristics of a grid cell, as described in Table 8.

$$ETm = Kc \times ETo$$

### 1.15.3 Actual evapotranspiration (ETa)

The actual uptake of water by the 'reference' crop is characterized by the actual evapotranspiration (ETa, mm/day) resulting in the daily calculations of the reference crop water balance. The calculation of ETa differentiates two possible cases depending on the availability of water for plant extraction:

i. Adequate soil water availability (ETa=ETm), and
ii. Limiting soil water availability (ETa<ETm).

Water balance calculation

The calculation of Eta involves daily soil water balance (Wb), which is defined as the volume of water available for plant uptake. The water balance, Wb, accounts for the accumulation of daily water inflow from precipitation (P), snowmelt (Sm), and outflow from the actual evapotranspiration (Eta), and excess water lost due to runoff and deep percolation (amount

of water that exceeds the upper limit of water available to plants, Wx). For the 'j' day of the year, the daily water balance is calculated as:

$$Wb_j = \min\left(Wb_{j-1} + Sm_j + P_j - ETa_j, Wx\right)$$

The upper limit Wx of water available to plants is the product of the available soil water (Sa) and rooting depth (D),

$$Wx = Sa \times D$$

The threshold of readily available soil moisture Wr is, in turn, calculated from Wx and the soil moisture depletion fraction (p),

$$Wr = Wx \times (1 - p)$$

Snow balance calculation

In seasonally cold climates the calculation of a snow balance (Sb, mm) affects the water balance procedure outlined above. The snow balance increases when precipitation falls as snow and decreases with snowmelt and snow sublimation. Precipitation (P) is assumed to fall as snow ($P^{snow}$) when maximum temperature (Tx) is below a certain temperature threshold (Ts).

The snowmelt (Sm) is calculated as a function of daily maximum temperature, the snow melt parameter ($\delta$) and depends on the previously accumulated snow balance. The snow melt factor $\delta$ is set to 5.5 mm/°C

$$Sm = \min\left(\delta \times (Tx - Ts), Sb\right)$$

Further details of the two possible cases of ETa calculation are as follows:

ETa for adequate soil water availability

A condition of 'adequate soil moisture availability' is defined when:

i. Daily precipitation (P) is greater or equal to ETm, and/or
ii. Combination of P and the difference between Wb and the readily-available-water threshold Wr is greater than Etm

$$ETa = ETm, \qquad for \begin{cases} P \geq ETm \\ P + (Wb - Wr) > Etm \end{cases}$$

ETa for limited water availability

When the soil water is limiting, then ETa falls short of ETm. In this case, ETa is calculated as a fraction ρ of ETm, where

$$\rho = Wb/Wr$$

The Eta is then calculated as

$$ETa = P + \rho \times ETm$$

This procedure assumes rainfall is immediately available to plants on the day of precipitation, prior to replenishing soil moisture.

### 1.15.4 LGP calculation

LGP refers to the number of days when average daily temperature is above $5^0$C ($LGP_{t5}$) and ETa of this reference crop exceeds a specified fraction of ETm. In the current GAEZ parameterization, LGP days are considered when ETa≥0.4×ETm, which aims to capture periods when sufficient soil moisture is available that would allow the establishment of the reference crop.

$$LGP = total\ number\ of\ days\ when\ \ ETa/ETm \geq\ 0.4$$

### 1.15.5 LGP Equivalent

Reference LGPs (Section 1.15.4) account for both temperature and soil moisture conditions and do not necessarily account for significant differences in wetness conditions especially within long LGPs (>225 days), for a better reflection of wetness conditions, so-called equivalent LGPs are used. Equivalent LGP is defined based on regression analysis of the reference LGP and the humidity index P/ETo as follows.

A quadratic polynomial is used to express the relationship between the number of growing period days and the annual humidity index. Parameters were estimated using data of all grid cells with essentially year-round temperature growing periods, i.e., with $LGP_{t5}$ = 365.

$$LGP_{eq} = \begin{cases} 14.0 + 293.66 \times (P/ETo) - 61.25 \times (P/ETo)^2, & when\ (P/ETo) \leq 2.4 \\ 366\ , & when\ \ P/ETo) > 2.4 \end{cases}$$

The equivalent LGP is used in the assessment of agro-climatic constraints, which relate environmental wetness with the occurrences of pest and diseases and workability constraints for harvesting conditions and for high moisture content of crop produce at harvest time.

In PyAEZ, the LGP, LGP classification, and LGP Equivalent are obtained through the following function,

```
# Length of Growing Period (LGP)
lgp = clim_reg.getLGP(Sa=100, D=1)
# Classification of LGP
lgp_class = clim_reg.getLGPClassified(lgp)
# LGP Equivalent
lgp_equv = clim_reg.getLGPEquivalent()
```

| **Function Arguments (for `getLGP`)** | |
|---|---|
| `Sa` | A single value or A 2D NumPy array, corresponding to available soil moisture holding capacity (mm=m). Usually, this value varies with soil texture. Hence, Sa can be provided as single value for entire area or 2D NumPy array that represent variation of soil moisture holding capacity depending on soil texture. Default value is 100 mm/m. This is an optional argument. |
| `D` | A single value, corresponding to corresponding rooting depth in meters. Default value is 1. This is an optional argument. |
| **Function Returns** | |
| `lgp` | 2D NumPy arrays of LGP [days] |

Table 8. Kc values used in Module 1 for the calculation of the maximum evapotranspiration (Etm)

| **Daily temperature condition** | **Remarks** | **Kc** |
|---|---|---|
| **Areas with year-round temperature growing period – $LGP_{t5}$=365 days** | | |
| Daily Ta ≥5°C; $LGP_{t5}$=365 days | In areas with year-round $LGP_{t5}$, the Kc value stays at 1 | 1.0 |
| **Areas with dormancy period or cold break – $LGP_{t5}$ less than 365 days** | | |
| Daily Ta ≤ 0°C; Tmax < 0°C | Precipitation falls as snow and is added to snow bucket | 0.0 |
| Daily Ta ≤ 0°C; Tmax≥0°C | Snowmelt takes place (water balance = precipitation + snow melt); minor evapotranspiration | 0.1 |
| 0°C < Daily Ta < 5°C; Ta trend upward | Some biological activities before the start of the growing period | 0.2 |
| Daily Ta ≥ 5°C; $LGP_{t5}$<365 days; Case 1 | Kc used for the days prior the start of the growing period | 0.5 |
| Daily Ta ≥5°C; $LGP_{t5}$<365 days; Case 2 | Kc increases from 0.5 to 1.0 during the first month of LGP | 0.5-1.0 |
| Daily Ta ≥5°C; $LGP_{t5}$<365 days; Case 1 | Kc=1 until the daily Ta falls below 5°C | 1.0 |
| 0°C < Daily Ta < 5°C; Ta trend downward | Reduced biological activities before dormancy | 0.2 |

## 1.16 Multiple cropping zones classification

Multiple cropping zones classification (

Table 9) is an additional agro-climatic indicator, which relates to the possibility of cultivating multiple sequential crops under rain-fed and irrigated conditions.

The PyAEZ's core modules perform calculation for single cropping systems. Additionally, several potential multiple cropping zones have been defined through matching the growth cycle with the temperature requirements based on Thermal Climate, Length of Growing period, Thermal Growing Period ($LGP_{t0}$ and $LGP_{t10}$), and the accumulated temperature summations ($Tsum_{t0}$, $Tsum_{t10}$). For more details on the multiple cropping zones classification please refer to the GAEZv4 documentation (Fischer et al., 2021).

```
# Multiple Cropping Zones classification
multi_crop_zone = clim_reg.getMultiCroppingZones(tclimate, lgp, lgpt5, lgpt10,
tsum0,tsum10)
```

| **Function Arguments** | |
|---|---|
| `tclimate` | 2D NumPy array, Thermal Climate classes (from Section 1.10) |
| `lgp` | 2D NumPy array, Length of Growing Period (from Section 1.15) |
| `lgpt5` | 2D NumPy array, Thermal LGP of Ta>5˚C (from Section 1.12) |
| `lgpt10` | 2D NumPy array, Thermal LGP of Ta>10˚C (from Section 1.12) |
| `tsum0` | 2D NumPy array, Temperature summation for Ta≥0˚C (from Section 1.13) |
| `tsum10` | 2D NumPy array, Temperature summation for Ta≥10˚C (from Section 1.13) |
| **Function Returns** | |
| `multi_crop_zone` | Python List of 2D NumPy arrays, as [multi_crop_rainfed, multi_crop_irrigated]. |

Table 9. Delineation of multiple cropping zones.

| Pixel values | Zone | Description |
|---|---|---|
| 1 | A | Zone of no cropping (too cold or too dry for rain-fed crops) |
| 2 | B | Zone of single cropping |
| 3 | C | Zone of limited double cropping (relay cropping; single wetland rice may be possible) |
| 4 | D | Zone of double cropping (note, in Zone D sequential double cropping including wetland rice is not possible) |
| 5 | E | Zone of double cropping with rice (sequential double cropping with one wetland rice crop is possible in Zone E) |
| 6 | F | Zone of double rice cropping or limited triple cropping (may partly involve relay cropping. A third crop is not possible in case of two wetland rice crops) |
| 7 | G | Zone of triple cropping (sequential cropping of three short-cycle crops; two wetland rice crops are possible in Zone G) |
| 8 | H | Zone of triple rice cropping (sequential cropping of three wetland rice crops is possible) |

## 1.17 Fallow period requirements

Fallow is an agricultural technique that consists of not sowing the arable land during one or more growing seasons. In AEZ framework, the fallow factors have been established by main crop groups and environmental conditions. The crop groups include cereals, legumes, roots and tubers, and a miscellaneous group consisting of long-term annuals/perennials. The fallow factors are expressed as percentage of time during the fallow-cropping cycle the land must be under fallow. PyAEZ determines the fallow requirements using Thermal Zones (Section 1.11). For further information on the fallow period requirement, please refer to Appendix 6-10 of the GAEZv4 documentation (Fischer et al., 2021).

```
# Fallow period requirements
fallow = clim_reg.TZoneFallowRequirement(tzone)
```

| **Function Arguments** | |
|---|---|
| `tzone` | 2D NumPy array, corresponding to Thermal Zone (Section 3.7) |
| **Function Returns** | |
| `fallow` | 2D NumPy array, corresponding to Thermal Zone for Fallow Requirements |

## 1.18 Permafrost evaluation

Occurrence of continuous or discontinuous permafrost conditions are used in the suitability assessment. Permafrost areas are characterized by sub-soil at or below the freezing point for two or more years. In this section, PyAEZ utilizes the air frost index (FI) which is used to characterize climate-derived permafrost condition into 4 classes: (i) Continuous permafrost; (ii) Discontinuous permafrost; (iii) Sporadic permafrost; and (iv) No permafrost. For detailed calculations for air frost index please refer to Chapter 3 of the GAEZv4 documentation (Fischer et al., 2021).

```
# Permafrost Evaluation
permafrost = clim_reg.AirFrostIndexandPermafrostEvaluation()
```

| **Function Arguments** | |
|---|---|
| `None` | |
| **Function Returns** | |
| `permafrost` | Python List of 2D NumPy arrays, as [frost index, permafrost class]. |

## 1.19 Agro-ecological Zones classification

The agro-ecological zones (AEZ) methodology provides a framework for establishing a spatial inventory of land resources compiled from global/national environmental data sets and assembled to quantify multiple spatial characteristics required for the assessments of land productivity under location-specific agro-ecological conditions.

The inventory combines spatial layers of thermal and moisture regimes with broad categories of soil/terrain qualities. It also indicates locations of areas with irrigated soils and shows land with severely limiting bio-physical constraints including very cold and very dry (desert) areas as well as areas with very steep terrain or very poor soil/terrain conditions. For further information on the classification criteria, please refer to Chapter 10 of the GAEZv4 documentation (Fischer et al., 2021).

```
# AEZ classification
aez_class = clim_reg.AEZClassification(tclimate, lgp, lgp_equv, lgpt_5,
soil_terrain_lulc, permafrost)
```

| **Function Arguments** | |
|---|---|
| `tclimate` | 2D NumPy array, Thermal Climate classes (from Section 1.10) |
| `lgp` | 2D NumPy array, Length of Growing Period (from Section 1.15) |
| `lgp_equv` | 2D NumPy array, LGP Equivalent (from Section 1.15.5) |
| `lgpt5` | 2D NumPy array, Thermal LGP of Ta>5˚C (from Section 1.12) |
| `soil_terrain_lulc` | 2D NumPy array, soil/terrain/special land cover classes (8 classes)<br>1: Dominantly very steep terrain<br>2: Dominantly hydromorphic soil<br>3: No or few soil/terrain limitations<br>4: Moderate soil/terrain limitations<br>5: Severe soil/terrain limitations<br>6: Irrigated soils<br>7: Water<br>8: Built-up/Artificial |
| `permafrost` | 2D NumPy array, Permafrost classes (from Section 1.18) |
| **Function Returns** | |
| `aez_class` | 2D NumPy array, 57 classes of AEZ |

# Module 2: Crop Simulation

## 1.20 Introduction

This key module simulates all the possible crop cycles to find the best crop cycle that produces maximum yield for a particular grid (Module 2 overview is shown in Figure 4). During the simulation process for each grid, 365 crop cycle simulations are performed. Each simulation corresponds to cycles that start from each day of the year (starting from Julian date 0 to Julian date o365). Similarly, this process is performed by the program for each grid in the study area.

Schematic representation of this process is shown in Figure 5. The attainable yields under irrigated and rain-fed conditions, during each crop cycle, are calculated with the help of several deterministic and empirical models as follows.

- **Total biomass** (de Wit, 1965): This model calculates total biomass produced by photosynthesis activities of plants under radiation condition of each grid. For more detailed calculations, refer to Chapter 4 in GAEZv4 Documentation (Fischer et al., 2021).
- **Crop yield from total biomass**: Crop yield is obtained as a portion of useful harvest from the total biomass. This portion is defined by an index call *Harvest Index (HI)*. Harvest index is defined as the amount of useful harvest divided by the total above ground biomass. For more detailed calculations, refer to Chapter 4 in GAEZv4 Documentation (Fischer et al., 2021).
- **Effects of water limitation on the crop yield**: This component is carried out for the rain-fed yield calculation only. In the case of irrigated conditions, this component is abandoned, as we are assuming that the water is not a limiting factor for crop growth. To address the water limitation on the crop yield, two major models are considered:
    - Reference Evapotranspiration – the Penman-Monteith equation (Pruitt & Doorenbos, 1977). A description of the implementation of the Penmann-Monteith equations is provided in Appendix 3-1 of Fischer et al., (2021).
    - Water balance calculation, together with applying the yield reduction factors based on water limitation(Smith, 1992)
- **Effects of temperature during crop cycle** and screening of crop cycles based on temperature requirements (termed *Thermal Screening*).

Similar to Module 1, we have to import and initiate the Class for the Crop Simulation module,

```
# import Module 2 Class
import CropSimulation
# create an instance – initiate the Class
aez = CropSimulation.CropSimulation()
```

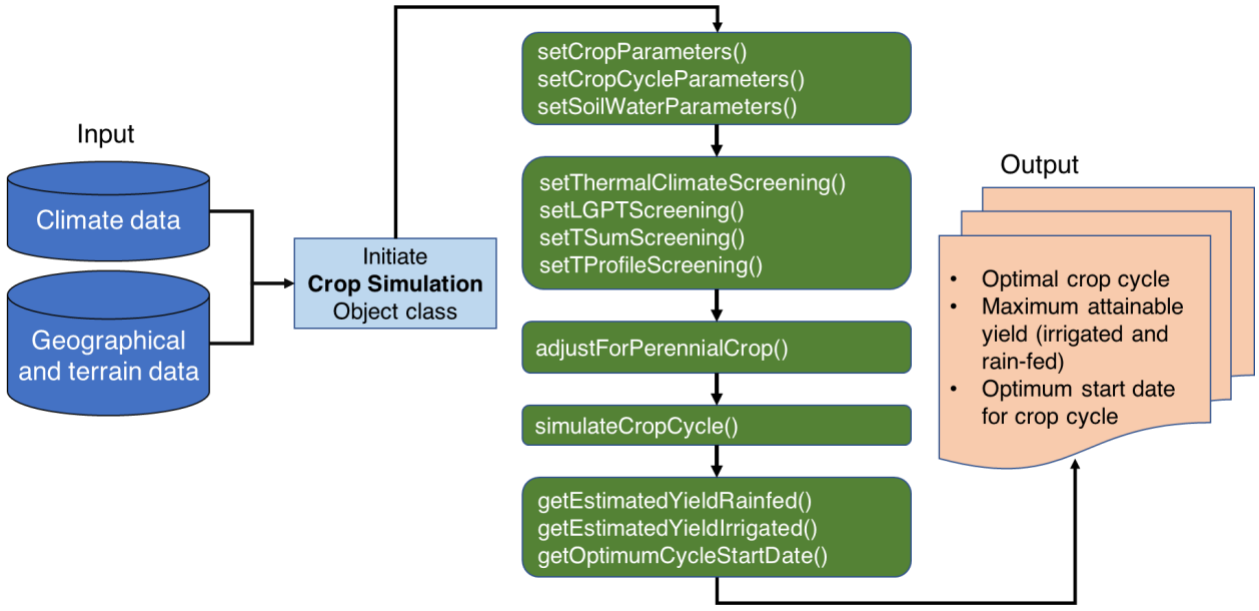


Figure 4. Overview of Module 2 (Crop Simulation) workflow.

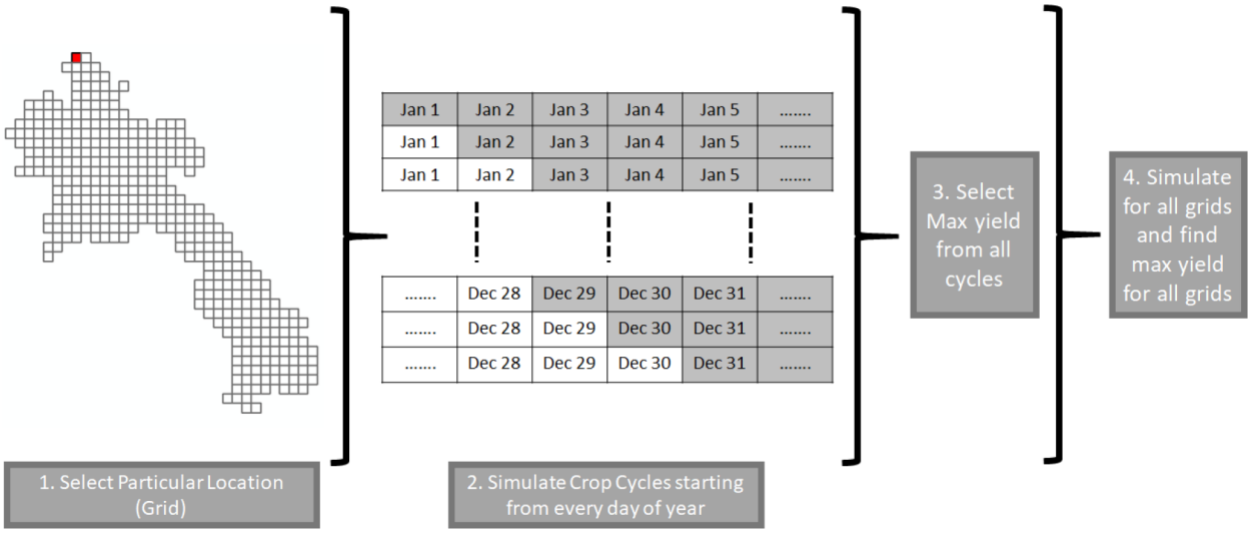


Figure 5. Overview of Crop Simulation routine.

## 1.21 Setting up inputs for Module 2

### 1.21.1 Geographical and terrain input

```
# Load geographical location and elevation data into the object class
aez.setLocationTerrainData(lat_min, lat_max, elevation)
```

| **Function Arguments** | |
|---|---|
| `lat_min` | a single value corresponding to the minimum latitude (decimal degrees) of the study area |
| `lat_max` | a single value corresponding to the maximum latitude (decimal degrees) of the study area |
| `elevation` | 2D NumPy array, elevation of the study area in metres |
| **Function Returns** | |
| None | |

### 1.21.2 Climate data input

First, we have to read and load the climate data (similar to what is done in Section 1.8) into Module 2 Class before proceeding with any calculations.

```
# Use the line below if MONTHLY data are used
aez.setMonthlyClimateData (min_temp, max_temp, precipitation, short_rad,
wind_speed, rel_humidity)
# Use the line below if DAILY data are used
aez.setDailyClimateData (min_temp, max_temp, precipitation, short_rad,
wind_speed, rel_humidity)
```

| **Function Arguments** | |
|---|---|
| `min_temp` | 3D NumPy array, daily or monthly minimum temperature (°C) |
| `max_temp` | 3D NumPy array, daily or monthly maximum temperature (°C) |
| `precipitation` | 3D NumPy array, daily or monthly total precipitation (mm/day) |
| `short_rad` | 3D NumPy array, daily or monthly solar radiation (W/m$^2$) |
| `wind_speed` | 3D NumPy array, daily or monthly windspeed at 2m elevation (m/s) |
| `rel_humidity` | 3D NumPy array, daily or monthly relative humidity (percentage decimal, 0-1) |
| **Function Returns** | |
| None | |

### 1.21.3 Setting Study Area Inputs (Optional)

This function is set up as an optional step which set up the mask layer as input which reduces the computation time outside the pixels of considerations.

```
1 # Set up mask for the study area (country, regional, or local)
2 clim_reg.setStudyAreaMask(admin_mask, no_data_value=0)
```

| **Function Arguments** | |
|---|---|
| admin_mask | 2D NumPy array, extracted only region of interest (Binary 0/1) |
| no_data_value | A single value, pixels equal to this value will be omitted during calculation |
| **Function Returns** | |
| None | |

### 1.21.4 Crop parameters input

This function allows users to set up the main crop parameters necessary for PyAEZ. This step is mandatory for Module 2 calculations.

```
1 # Set crop paramaters input for Module 2
2 aez.setCropParameters(LAI, HI, legume, adaptability, cycle_len, D1, D2)
```

| **Function Arguments** | |
|---|---|
| LAI | A single value, Leaf Area Index |
| HI | A single value, Harvest Index |
| legume | A single binary value, Is the crop is legume? No=0, Yes=1 |
| adaptability | A single value, corresponding to adaptability class of the crop. Hence, value must be either 1, 2, 3, or 4 corresponding to adaptability class of the crop. |
| cycle_len | A single value, corresponding length of crop cycle [days] |
| D1 | A single value, corresponding rooting depth in metres at the beginning of the crop cycle |
| D2 | A single value, corresponding rooting depth in metres after maturity (D1 and D2 can also be same value. In this case, interpolations will not be applied, and same rooting depth will be applying for the entire crop cycle) |
| **Function Returns** | |
| None | |

### 1.21.5 Soil water parameter input

This function allow user to set up the parameters related to the soil water storage.

```
1  # Set soil water parameter
2  aez.setSoilWaterParameters(Sa, pc)
```

| Function Arguments | |
|---|---|
| Sa | A single value or a 2D NumPy array, corresponding to available soil moisture holding capacity [mm/m]. Usually, this value varies with soil texture. Sa can be provided as single value for entire area or 2D NumPy array that represents variation of soil moisture holding capacity depending on soil texture. |
| pc | A single value between 0 and 1, corresponding to soil water depletion fraction below which ETa < ETo. |
| **Function Returns** | |
| None | |

### 1.21.6 Crop cycle parameter input

This function allow user to set up the parameters related to the crop cyles.

```
1 # Set crop cycle parameter
2 aez.setCropCycleParameters(stage_per, kc, kc_all, yloss_f, yloss_f_all )
```

| Function Arguments | |
|---|---|
| stage_per | A 4-element numerical list, corresponding to percentage of each 4 stages of a crop cycle, namely initial (d1), vegetative (d2), reproductive (d3), and maturation stage (d4). Example: stage per=[10, 30, 30, 30] |
| kc | A 3-element numerical list, corresponding crop water requirements for initial, reproductive, the end of the maturation stage. Example: kc=[1.1, 1.2, 1] |
| kc_all | A single value, corresponding to crop water requirements for entire growth cycle. |
| yloss_f | A 4-element numerical list, corresponding to yield loss factors of each 4 stages of a crop cycle, namely initial (d1), vegetative (d2), reproductive (d3), and maturation stage (d4). Example: yloss_f=[1,2,2.5,1] |
| yloss_f_all | A single value corresponding to yield loss factor for entire growth cycle. |
| **Function Returns** | |
| None | |

### 1.21.7 Thermal screening input

The functions in this section will screen the suitability of grid-cells for the possible presence of individual LUTs. The crops' temperature requirements will be matched with the prevailing thermal conditions (Thermal Regime characteristics calculated in Module 1 (Section 1.10-1.14).

**Thermal climate**

PyAEZ 's screening of crop/LUTs about thermal climate results in a 'yes/no' filter for further calculations.

**Thermal growing period**

Growth cycle lengths of crop/LUTs are matched with $LGP_{t5}$ (Section 1.12). The result of the matching provides optimum match when the growth cycle can generously be accommodated within $LGP_{t5}$. Otherwise, the match is considered not suitable.

**Accumulated temperature sum**

The matching of the crop LUT heat unit requirements with the prevailing temperature sum is:

- optimum, when the requirements are within the specified optimum Tsum range, and
- not suitable, when prevailing Tsum range are too high or too low.

**Temperature profile**

Potential crop calendars of each LUT are tested for the match of crop/LUT temperature profile requirements and grid-cell temperature profiles, while considering growth cycle starting days within the length of the growing period for rain-fed conditions, and separately within the year for irrigated conditions.
For all feasible crop calendars within the LGP (rain-fed) or within the year (irrigated), the temperature profile conditions are tested against optimum and suboptimum crop temperature profile requirements and in each case an "optimum" or "not suitable" match is established.

```
1 # Set parameters for Thermal Screening
2 aez.setThermalClimateScreening(tclimate,no_tclimate)
3 aez.setLGPTScreening(no_lgpt, optm_lgpt)
4 aez.setTsumScreening(no_Tsum, optm_Tsum)
5 aez.setTProfileScreening(no_Tprofile, optm_Tprofile)
```

| **Function Arguments** | |
|---|---|
| `tclimate` | 2D NumPy array, corresponding to thermal climate (an output of Module 1). |
| `no_tclimate` | A numerical list, corresponding to pixel values of "not suitable" thermal climate zones. |
| `no_lgpt` | 3-elements numerical list, "not suitable" 3 LGPt conditions (as in Module 1). |
| `optm_lgpt` | 3-elements numerical list, "optimum" 3 LGPt conditions (as in Module 1). |
| `no_Tsum` | 3-elements numerical list, "not suitable" 3 Tsum conditions (as in Module 1). |
| `optm_Tsum` | 3-elements numerical list, "optimum" 3 Tsum conditions (as in Module 1). |
| `no_Tprofile` | 18-elements numerical list, "not suitable" 18 Tprofile conditions (as in Module 1). |
| `optm_Tprofile` | 18-elements numerical list, "optimum" 18 Tprofile conditions (as in Module 1). |
| **Function Returns** | |
| None | |

### 1.21.8 Adjustment for Perennial Crop (optional)

If a perennial crop is introduced, PyAEZ will perform adjustment on the Leaf Area Index (LAI) and the Harvest Index (HI) based on the effective growing period. For detailed calculates and adjustment values, please refer to Chapter 4 in GAEZv4 documentation (Fischer et al., 2021).

```
# Set parameters for adjusting for Perennial Crop
aez.adjustForPerennialCrop(aLAI, bLAI, aHI, bHI)
```

| **Function Arguments** | |
|---|---|
| aLAI | A single value, corresponding to $\alpha_{LAI}$. Example: Arabica coffee $\alpha_{LAI}$=0 |
| bLAI | A single value, corresponding to $\beta_{LAI}$. Example: Arabica coffee $\beta_{LAI}$=270 |
| aHI | A single value, corresponding to $\alpha_{HI}$. Example: Arabica coffee $\alpha_{HI}$=120 |
| bHI | A single value, corresponding to $\beta_{HI}$. Example: Arabica coffee $\beta_{HI}$=120 |
| **Function Returns** | |
| None | |

## 1.22 Calculations and outputs

### 1.22.1 Crop cycle simulation

After setting up all of the related parameters in Section 1.21, we can now run the crop cycle simulations/calculations by executing the function below:

```
# Crop cycle simulation
aez.simulateCropCycle(start_doy=1, end_doy=365, step_doy=1, leap_year=False)
```

| **Function Arguments** | |
|---|---|
| start_doy | A single value, corresponding to crop simulations starting Julian date.<br>This is an optional argument. Default value is 0. |
| end_doy | A single value, corresponding to crop simulations ending Julian date.<br>This is an optional argument. Default value is 365. |
| step_doy | A single value, corresponding to spacing (in days) between 2 adjacent crop simulations.<br>This is an optional argument. Default value is 1. |
| leap_year | • True or False, depending on whether the simulating year is a leap year or not. This allows handing leap and non-leap year differently.<br>• This is only relevant for monthly climate data because this value will be used in interpolation processes.<br>• In case of daily climate data inputs, length of daily climate data vector will be taken as number of days in a year.<br>• This is an optional argument, and the default value is False. |
| **Function Returns** | |
| None | |

### 1.22.2 Estimated Maximum Yield

These functions return the maximum attainable yield under the provided climate conditions in both rain-fed and irrigated conditions. The result's unit is in Kilograms per hectare (kg/ha) (Figure 6).

```
# Estimation of Maximum Yield for Rainfed scenario
yield_map_rain = aez.getEstimatedYieldRainfed()
# Estimation of Maximum Yield for Irrigated scenario
yield_map_irr = aez.getEstimatedYieldIrrigated()
```

| **Function Arguments** | |
|---|---|
| None | |
| **Function Returns** | |
| `yield_map_rain` | 2D NumPy arrays, the maximum attainable yield under the provided climate conditions, under rain-fed conditions [kg/ha] |
| `yield_map_irr` | 2D NumPy arrays, the maximum attainable yield under the provided climate conditions, under irrigated conditions [kg/ha] |

### 1.22.3 Optimum Crop Calendar

```
# Optimum starting date for crop cycle
starting_date = aez.getOptimumCycleStartDate()
```

| **Function Arguments** | |
|---|---|
| None | |
| **Function Returns** | |
| `starting_date` | 2D NumPy arrays. Each pixel value corresponds to the Julian day of the optimum crop cycle starting date. |

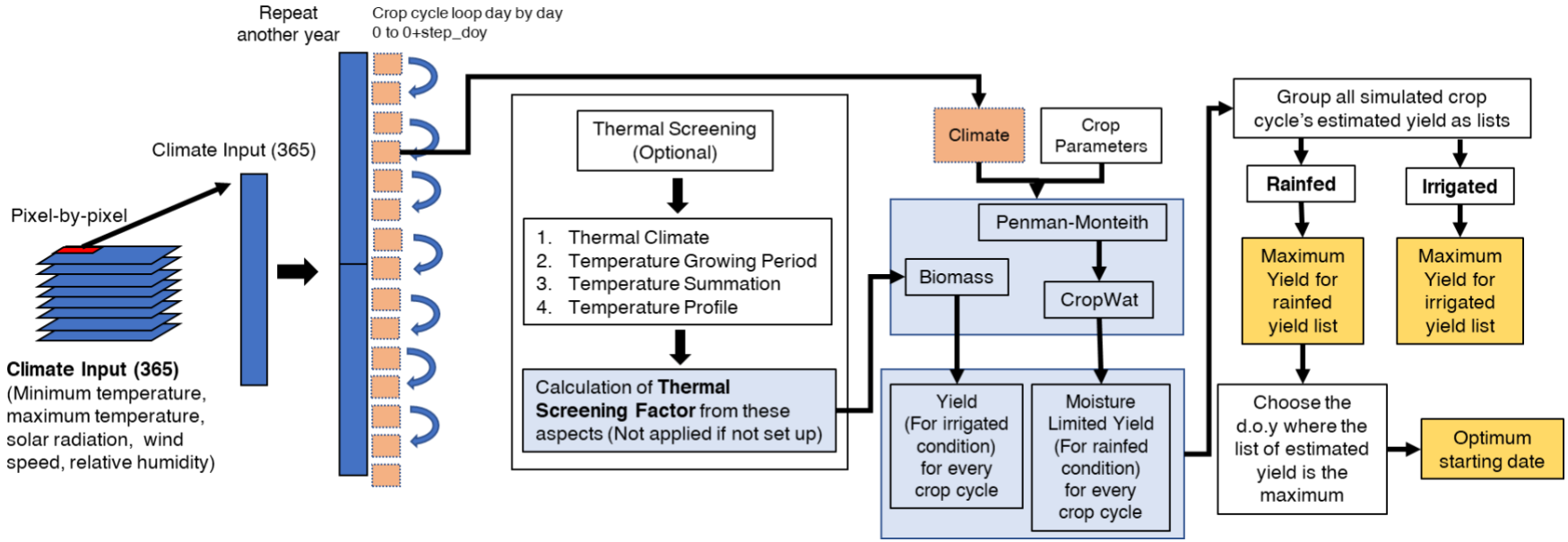


Figure 6. Estimation of Maximum Attainable Yield (rainfed and irrigated) and Optimum Starting Date (Module 2).

# Module 3: Climate Constraints

## 1.23 Introduction

In this module, various yield reduction factors will be applied to the maximum attainable yield estimated from Module 2 (Section 0) to consider the constraining effects which are difficult to simulate during the crop cycle simulation (Figure 7). For example, climatic effects can be pests, diseases, and poor workability due to excess soil moisture. These effects, in turn, depend on the different levels of inputs and LGP Equivalent (Section 1.15.5).

All of the reduction factors used in Modules 3, 4, and 5 are located in 2 parameter files corresponding to irrigated and rain-fed conditions. These files MUST be edited with the reduction factors values corresponding to each crop and input level. Users are strongly encouraged to advised to use specific reduction factors based on national research for national-level analysis.

This module considers types of agro-climatic constrains:

- Long term limitation to crop performance due year-to-year rainfall variability
- Pests, diseases, and weeds damages on plant growth
- Pests, diseases, and weeds damages on produce's quality
- Climatic factors affecting the efficiency of farming operations

See GAEZv4 documentation for further details on the climate constraints (Fischer et al., 2021)

Similar to the previous modules, this module starts with importing and initiating the Class:

```
1 # Import and create a Class instgance
2 import ClimateConstraints
3 obj_constraints = ClimaticConstraints.ClimateConstraints()
```

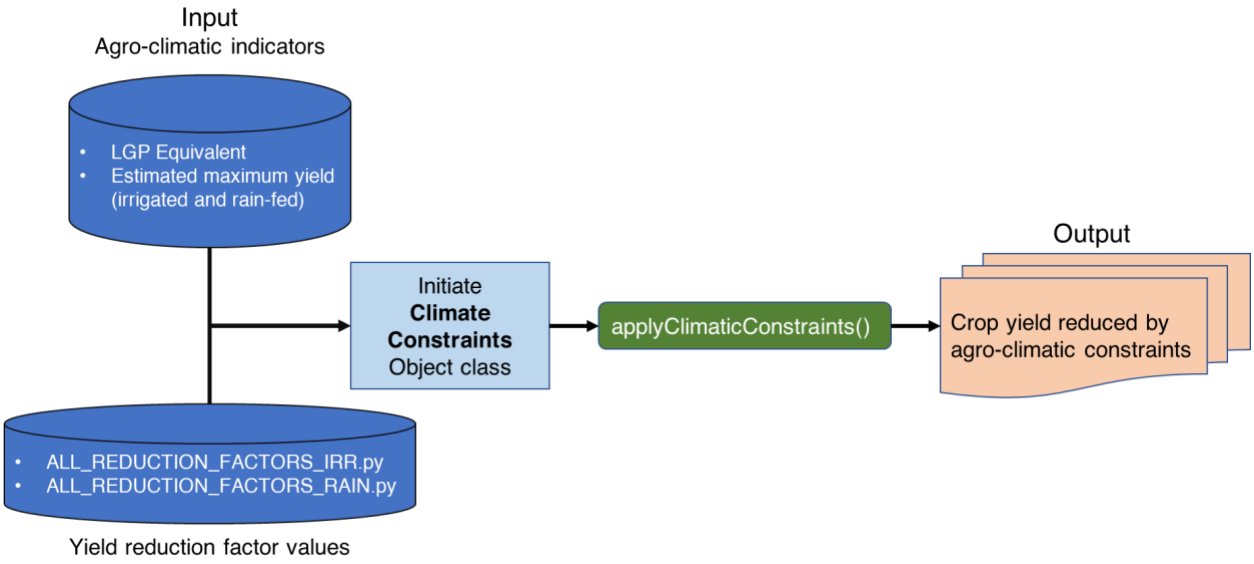


Figure 7. Overview of Module 3 (Climate Constraints) workflow.

## 1.24 Setting up parameter files

The following 2 parameter files contains the reduction factors values to be used in Module 3, 4, and 5:

| File | Remark |
|---|---|
| ALL_REDUCTION_FACTORS_IRR.py | Reduction factors for irrigated conditions |
| ALL_REDUCTION_FACTORS_RAIN.py | Reduction factors for rain-fed conditions |

Within the two parameter files, the values related to climatic constraint are input as:

```
'''-------------------------------------------------------'''
'''Reduction Factors for Climatic Constraints'''
'''-------------------------------------------------------'''
#defining yield reduction factors based of LGP Equivalent class
lgp_eq_class = [[0,29],[30,59],[60,89],[90,119],[120,149],
                [150,179],[180,209],[210,239],[240,269],
                [270,299],[300,329],[330,366]]

lgp_eq_red_fr = [[25,25,25,25,25,25,25,50,50,50,75,75],
                 [100,100,100,100,100,100,100,100,100,100,100,100],
                 [50,50,50,50,50,75,75,100,100,100,100,75],
                 [100,100,100,100,100,100,100,100,100,100,100,75]]
```

| Parameters | |
|---|---|
| lgp_eq_class | 2D List, corresponding to the LGP Equivalent classes [days] |
| lgp_eq_red_fr | 2D List, corresponding to reduction factors.<br>The rows are corresponding to 4 types of agro-climatic constraints which are mentioned in the above section and columns are corresponding to LGP Equivalent classes as in lgp eq class. |

## 1.25 Applying climate constraints

This function applies the climate-related yield reduction factors to produce the reduced yield:

```
1 # Apply climate constraints
2 yield_out = obj_constraints.applyClimaticConstraints(lgp_eq,yield_in,irr_or_rain)
```

| Function Arguments | |
|---|---|
| lpg_eq | 2D NumPy array, corresponding to LGP Equivalent (Section 1.15.5). |
| yield_in | 2D NumPy array, corresponding to the yield before applying the climatic reduction factors |
| irr_or_rain | A single character (string), indicating whether yield_in is under irrigated(I) or rain-fed condition(R). |
| **Function Returns** | |
| yield_out | 2D NumPy array. The yield reduced by climatic factors [same unit as yield_in] |

# Module 4: Soil Constraints

## 1.26 Introduction

After applying the agro-climatic constraints (Section 0) onto the maximum attainable yield, we will now apply the soil constraints (Figure 8).

The combination of 7 soil qualities (SQ), which are based on the soil characteristics of each soil unit, and the input level gives us a single yield reduction factor – *Soil Rating*, which will be applied to the remaining yield. For more details on this calculation, please refer to GAEZv4 documentation (Fischer et al., 2021).

- SQ1: Nutrient availability
- SQ2: Nutrient retention capacity
- SQ3: Rooting conditions
- SQ4: Oxygen availability to roots
- SQ5: Excess salts
- SQ6: Toxicity
- SQ7: Workability (constraining field management)

The soil characteristic of each soil unit must be prepared as outlined in Section 1.4.2 (Table 3). The two CSV files of the topsoil and subsoil characteristics are:

- soil_characteristics_topsoil.csv
- soil_characteristics_subsoil.csv

These two files are located in `./input_data/csv/` folder.

First, we must import Module 4 Class:

```
1 import SoilConstraints
2 soil_constraints = SoilConstraints.SoilConstraints()
```

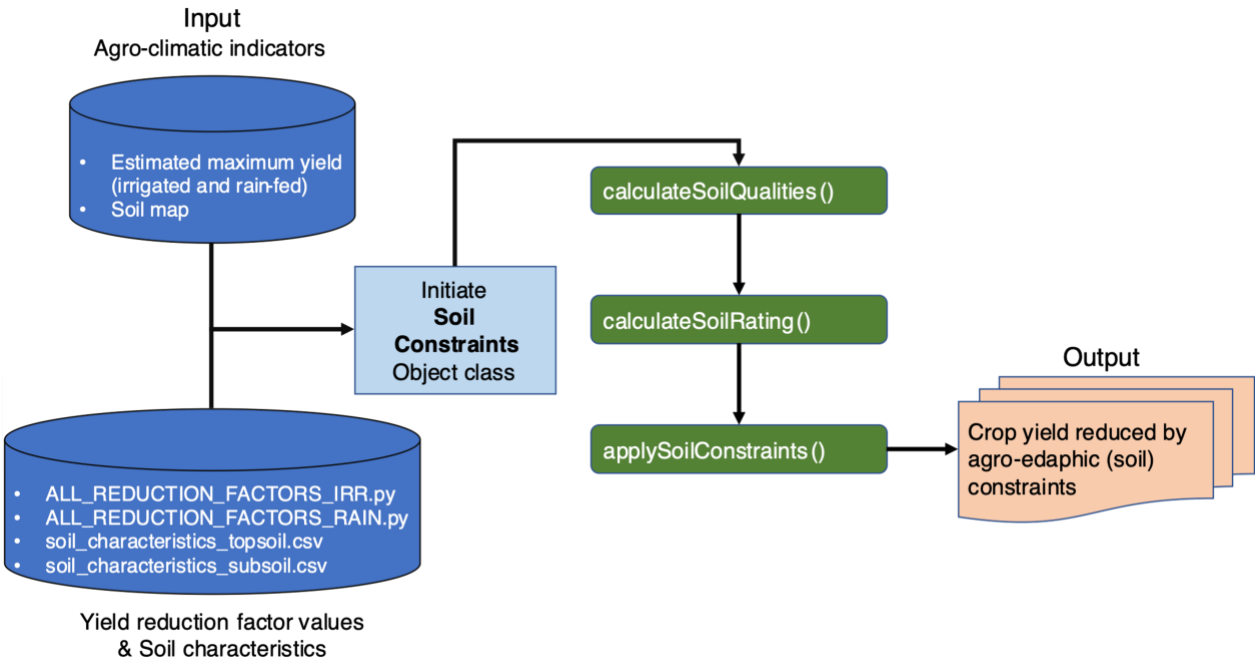


Figure 8. Overview of Module 4 (Soil Constraints) workflow.

## 1.27 Setting up parameter files

The following 2 parameter files contains the reduction factors values to be used in Module 3, 4, and 5:

| File | Remark |
|---|---|
| ALL_REDUCTION_FACTORS_IRR.py | Reduction factors for irrigated conditions |
| ALL_REDUCTION_FACTORS_RAIN.py | Reduction factors for rain-fed conditions |

```
'''Reduction Factors for Soil Constraints'''
# Value - values of soil characteristics (mush be ascending order)
# Factor - yield reduction factors corresponding to each value

# Soil texture for SQ1
TXT1_value = ['Fine', 'Medium', 'Coarse']
TXT1_factor = [90, 70, 30]

# Soil texture for SQ2
TXT2_value = ['Fine', 'Medium', 'Coarse']
TXT2_factor = [90, 70, 30]

```

```
# Soil texture for SQ7
TXT7_value = ['Fine', 'Medium', 'Coarse']
TXT7_factor = [90, 70, 30]

# Soil organic carbon
OC_value = [0, 0.8, 1.5, 2]
OC_factor = [50, 70, 90, 100]

# Soil pH
pH_value = [3.6, 4.1, 4.5, 5, 5.5, 6]
pH_factor = [10, 30, 50, 70, 90, 100]

# Total exchangeable bases
TEB_value = [0, 1.6, 2.8, 4, 6.5]
TEB_factor = [30, 50, 70, 90, 100]

# Base saturation
BS_value = [0, 35, 50, 80]
BS_factor = [50, 70, 90, 100]

# Cation exchange capacity of soil
CECsoil_value = [0, 2, 4, 8, 10]
CECsoil_factor = [30, 50, 70, 90, 100]

# Cation exchange capacity of clay
CECclay_value = [0, 16, 24]
CECclay_factor = [70, 90, 100]

# Effective soil depth
RSD_value = [35, 70, 85]
RSD_factor = [50, 90, 100]

# Soil coarse material (Gravel)
GRC_value = [10, 30, 90] # %
GRC_factor = [100, 35, 10]

# Drainage. VP: very poor, P: Poor, I: Imperfectly, MW: Moderately well, W:
Well, SE: Somewhat Excessive, E: Excessive
DRG_value = ['VP', 'P', 'I', 'MW', 'W', 'SE', 'E']
DRG_factor = [50, 90, 100, 100, 100, 100, 100]

# Exchangeable sodium percentage
ESP_value = [10, 20, 30, 40, 100] # %
ESP_factor = [100, 90, 70, 50, 10]

# Electric conductivity
EC_value = [1, 2, 4, 6, 12, 100] # dS/m
EC_factor = [100, 90, 70, 50, 30, 10]

# Soil phase rating for SQ3
SPH3_value = ['Lithic', 'skeletic', 'hyperskeletic']
SPH3_factor = [100, 50, 30]
```

```
65
66 # Soil phase rating for SQ4
67 SPH4_value = ['Lithic', 'skeletic', 'hyperskeletic']
68 SPH4_factor = [100, 50, 30]
69
70 # Soil phase rating for SQ5
71 SPH5_value = ['Lithic', 'skeletic', 'hyperskeletic']
72 SPH5_factor = [100, 50, 30]
73
74 # Soil phase rating for SQ6
75 SPH6_value = ['Lithic', 'skeletic', 'hyperskeletic']
76 SPH6_factor = [100, 50, 30]
77
78 # Soil phase rating for SQ7
79 SPH7_value = ['Lithic', 'skeletic', 'hyperskeletic']
80 SPH7_factor = [100, 50, 30]
81
82 # Other soil depth/volume related characteristics rating
83 OSD_value = [0]
84 OSD_factor = [100]
85
86 # Soil property rating - vertic or not
87 SPR_value = [0, 1]
88 SPR_factor = [100, 90]
89
90 # Calcium carbonate
91 CCB_value = [3, 6, 15, 25, 100] # %
92 CCB_factor = [100, 90, 70, 50, 10]
93
94 # Gypsum
95 GYP_value = [1, 3, 10, 15, 100] # %
96 GYP_factor = [100, 90, 70, 50, 10]
97
98 # Vertical properties
99 VSP_value = [0, 1]
100 VSP_factor = [100, 90]
```

| **Parameters** | |
|---|---|
| **Soil texture** | |
| `TXT1_value` | List of Strings, corresponding to soil texture types for SQ1. |
| `TXT1_factor` | List of numerical values, corresponding to respective reduction factors to TXT1 value. |
| `TXT2_value` | List of Strings, corresponding to soil texture types for SQ2. |
| `TXT2_factor` | List of numerical values, corresponding to respective reduction factors to TXT2 value. |
| `TXT7_value` | List of Strings, corresponding to soil texture types for SQ7. |
| `TXT7_factor` | List of numerical values, corresponding to respective reduction factors to TXT7 value. |
| **Soil organic carbon** | |
| `OC_value` | List of numerical values, corresponding to soil organic carbon.<br>Values must me in ascending order. |

| | |
|---|---|
| OC_factor | List of numerical values, corresponding to respective reduction factors to OC_value. |
| **Soil pH** | |
| pH_value | List of numerical values, corresponding to soil pH. Values must me in ascending order. |
| pH_factor | List of numerical values, corresponding to respective reduction factors to pH_value. |
| **Total exchangeable bases** | |
| TEB_value | List of numerical values, corresponding to total exchangeable bases. Values must me in ascending order. |
| TEB_factor | List of numerical values, corresponding to respective reduction factors to TEB_value. |
| **Base saturation** | |
| BS_value | List of numerical values, corresponding to base saturation. Values must me in ascending order. |
| BS_factor | List of numerical values, corresponding to respective reduction factors to BS_value. |
| **Cation exchange capacity of soil** | |
| CECsoil_value | List of numerical values, corresponding to cation exchange capacity of soil. Values must me in ascending order. |
| CECsoil_factor | List of numerical values, corresponding to respective reduction factors to CECsoil_value. |
| **Cation exchange capacity of clay** | |
| CECclay_value | List of numerical values, corresponding to cation exchange capacity of clay. Values must me in ascending order. |
| CECclay_factor | List of numerical values, corresponding to respective reduction factors to CECclay_value. |
| **Effective soil depth** | |
| RSD_value | List of numerical values, corresponding to effective soil depth. Values must me in ascending order. |
| RSD_factor | List of numerical values, corresponding to respective reduction factors to RSD_value. |
| **Soil coarse material (Gravel)** | |
| GRC_value | List of numerical values, corresponding to soil coarse material (Gravel) content as percentage. Values must me in ascending order. |
| GRC_factor | List of numerical values, corresponding to respective reduction factors to GRC_value. |
| **Drainage class** | |
| DRG_value | List of Strings, corresponding to drainage class (VP: very poor, P: Poor, I: Imperfectly, MW: Moderately well, W: Well, SE: Somewhat Excessive, E: Excessive). |
| DRG_factor | List of numerical values, corresponding to respective reduction factors to DRG_value. |
| **Exchangeable Sodium percentage** | |
| ESP_value | List of numerical values, corresponding to exchangeable sodium percentage. Values must me in ascending order. |
| ESP_factor | List of numerical values, corresponding to respective reduction factors to ESP_value. |
| **Electric conductivity** | |
| EC_value | List of numerical values, corresponding to electric conductivity. Values must me in ascending order. |
| EC_factor | List of numerical values, corresponding to respective reduction factors to EC_value. |
| **Soil phase rating – stagnic or gleyic, present or not** | |
| SPH3_value | List of Strings, corresponding to soil phase class for SQ3. |

| | |
|---|---|
| SPH3_factor | List of numerical values, corresponding to respective reduction factors to SPH3_value. |
| SPH4_value | List of Strings, corresponding to soil phase class for SQ4. |
| SPH4_factor | List of numerical values, corresponding to respective reduction factors to SPH4_value. |
| SPH5_value | List of Strings, corresponding to soil phase class for SQ5. |
| SPH5_factor | List of numerical values, corresponding to respective reduction factors to SPH5_value. |
| SPH6_value | List of Strings, corresponding to soil phase class for SQ6. |
| SPH6_factor | List of numerical values, corresponding to respective reduction factors to SPH6_value. |
| SPH7_value | List of Strings, corresponding to soil phase class for SQ7. |
| SPH7_factor | List of numerical values, corresponding to respective reduction factors to SPH3_value. |
| OSD_value | List of numerical values, corresponding to other soil depth/volume related characteristics rating. |
| OSD_factor | List of numerical values, corresponding to respective reduction factors to OSD_value. |
| **Soil property rating – vertic or not** | |
| SPR_value | List of numerical values, corresponding to soil property rating. Values in the list can be either 0 or 1 depending on availability of particular soil phases.<br>Values must me in ascending order. |
| SPR_factor | List of numerical values, corresponding to respective reduction factors to SPR_value. |
| **Calcium carbonate** | |
| CCB_value | List of numerical values, corresponding to calcium carbonate content as percentage.<br>Values must me in ascending order. |
| CCB_factor | List of numerical values, corresponding to respective reduction factors to CCB_value. |
| **Gypsum** | |
| GYP_value | List of numerical values, corresponding to gypsum content as percentage.<br>Values must me in ascending order. |
| GYP_factor | List of numerical values, corresponding to respective reduction factors to GYP_value. |
| **Vertical properties** | |
| VSP_value | List of numerical values, corresponding to vertical properties.<br>Values in the list can be either 0 or 1 depending on availability of vertical properties.<br>Values must me in ascending order. |
| VSP_factor | List of numerical values, corresponding to respective reduction factors to VSP_value. |

## 1.28 Calculate soil qualities

This function calculates 7 soil qualities for each soil unit based on the input soil characteristics.

```
1 # Soil Qualities
2 soil_constraints.calculateSoilQualities(irr_or_rain)
```

| **Function Arguments** | |
|---|---|
| irr_or_rain | Single character String, indicating calculations are considered under either rain-fed condition or irrigated condition. 'R' is for rain-fed condition, and 'I' is for irrigated condition. |
| **Function Returns** | |
| None | |

## 1.29 Calculate soil ratings

This function calculates soil ratings for each soil unit, combining 7 soil qualities based on input level.

```
1 # Soil rating
2 soil_constraints.calculateSoilRating(input_level)
```

| **Function Arguments** | |
|---|---|
| `input_level` | Single character String, corresponding to input level. 'L' is for Low input level, 'I' is for Intermediate input level, and 'H' is for Hight input level |
| **Function Returns** | |
| None | |

## 1.30 Extracting soil qualities

This function returns 7 soil qualities calculated for each soil unit based on the input soil characteristics.

```
1 # Extracting soil qualities
2 soil_qualities = soil_constraints.getSoilQualities()
```

| **Function Arguments** | |
|---|---|
| None | |
| **Function Returns** | |
| `soil_qualities` | 2D NumPy array. Each row corresponds to soil units. The first column corresponds to soil unit code. Column 2-8 correspond to the 7 soil qualities. |

## 1.31 Extracting soil ratings

This function returns 7 soil qualities calculated for each soil unit based on the input soil characteristics.

```
1 # Extracting soil qualities
2 soil_ratings = soil_constraints.getSoilRatings()
```

| **Function Arguments** | |
|---|---|
| None | |
| **Function Returns** | |
| `soil_ratings` | 2D NumPy array. Each row corresponds to soil units. The first column corresponds to soil unit code. The second |

## 1.32 Applying soil constraints

This function applies all soil-related yield reduction factors.

```
1 # Soil Constraints
2 yield_out = soil_constraints.applySoilConstraints(soil_map, yield_in)
```

| **Function Arguments** | |
|---|---|
| `soil_map` | 2D NumPy array, corresponding to soil unit. Each pixel value must be soil unit code. This code is used to match the soil rating with the input yield. |
| `yield_in` | 2D NumPy array, corresponding to the yield before applying the soil reduction factors |
| **Function Returns** | |
| `yield_out` | 2D NumPy array. The yield reduced by soil-related factors [same unit as `yield_in`] |

# Module 5: Terrain Constraints

## 1.33 Introduction

This section introduces the yield reduction due to terrain slope, soil erosion, and Fournier Index (FI) (Figure 9). The FI is based on the monthly precipitation (climate-related). These yield reduction factors will be applied to the maximum attainable yield. For detailed calculations for this section, please refer to the GAEZv4 documentation (Fischer et al., 2021).

First, we must import Module 5 Class:

```
import TerrainConstraints
terrain_constraints = TerrainConstraints.TerrainConstraints()
```

## 1.34 Setting up parameter files

The following 2 parameter files contains the reduction factors values to be used in Module 3, 4, and 5:

| File | Remark |
|---|---|
| ALL_REDUCTION_FACTORS_IRR.py | Reduction factors for irrigated conditions |
| ALL_REDUCTION_FACTORS_RAIN.py | Reduction factors for rain-fed conditions |

```
'''Reduction Factors for Terrain Constraints'''
'''-------------------------------------------------------'''
# Classes of slopes (Percentage Slope)
Slope_class = [[0,0.5],[0.5,2],[2,5],[5,8],[8,16],[16,30],[30,45],[45,100]]
# Classes of Fournier index
FI_class=[[0,1300],[1300,1800],[1800,2200],[2200,2500],[2500,2700],[1700,100000]]
# Sample data are for irrigated-intermediate input-wetland rice
# Rows corresponding to FI class and columns corresponding to slope class
Terrain_factor =[[100,   100,    75,     50,     25,     0,      0,      0],
                 [100,   100,    100,    100,    100,    75,     0,      0],
                 [100,   100,    100,    100,    75,     25,     0,      0],
                 [100,   100,    100,    100,    50,     0,      0,      0],
                 [100,   100,    100,    100,    25,     0,      0,      0],
                 [100,   100,    100,    100,    25,     0,      0,      0]]
```

| Parameters | |
|---|---|
| slope_class | 2D List, corresponding to slope classes. Slope unit must me Percentage Slope. |
| FI_class | 2D List, corresponding to Fournier index (FI) classes. |
| Terrain_factor | 2D List, corresponding to reduction factors. The rows are corresponding to FI classes and the columns correspond to slope classes. |

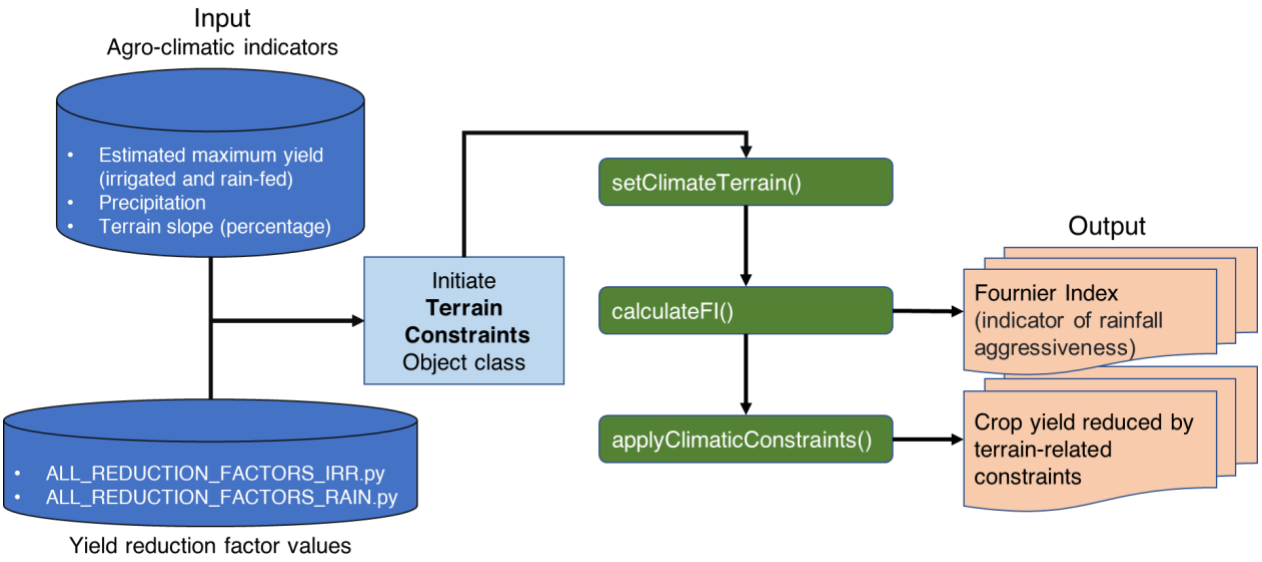


Figure 9. Overview of Module 5 (Terrain Constraints) workflow.

## 1.35 Setting up inputs

### 1.35.1 Climate and terrain inputs

This function allows users to set up the monthly precipitation and terrain slope data. This is a mandatory step before executing further calculations.

```
1 # Set up climate and slope data
2 terrain_constraints.setClimateTerrainData(precipitation, slope)
```

| **Function Arguments** | |
|---|---|
| `precipitation` | 3D NumPy array corresponding to monthly precipitation. Unit of monthly precipitation can be of any unit, since Fournier Index (FI) is a ratio, unit conversion factors will be cancelled out. |
| `slope` | 2D NumPy array, corresponding to terrain slope. [Percentage Slope] |
| **Function Returns** | |
| None | |

### 1.35.2 Calculate Fournier Index

This function calculates Fournier Index (FI) based on the input monthly precipitation. FI is a simple index that indicates the potential of soil erosion based on monthly precipitation.

```
1 # Calculate Fournier Index
2 terrain_constraints.calculateFI()
```

| Function Arguments | |
|---|---|
| None | |
| **Function Returns** | |
| None | |

### 1.35.3 Extract Fournier Index

This function returns Fournier Index (FI), which is based on the input monthly precipitation. This is an optional function. FI can be extracted with this function if required.

```
1 # Extract Fournier Index
2 fi = terrain_constraints.getFI()
```

| Function Arguments | |
|---|---|
| None | |
| **Function Returns** | |
| `fi` | 2D NumPy array, corresponding to Fournier Index (FI) based on the input monthly precipitation |

## 1.36 Applying terrain constraints

This function applies the terrain-related yield reduction factors.

```
1 # Apply Terrain Constraints
2 yield_out = terrain_constraints.applyTerrainConstraints(yield_in, irr_or_rain)
```

| Function Arguments | |
|---|---|
| `yield_in` | 2D NumPy array, corresponding to the yield before applying the terrain-related reduction factor. This can be the yield under either irrigated or rain-fed conditions from Module 4 (Section 0). |
| `irr_or_rain` | single character String, indicating yield in is in either rain-fed or irrigated condition. 'R' is for rain-fed condition, and 'I' is for irrigated condition. |
| **Function Returns** | |
| `yield_out` | 2D NumPy array. The yield reduced by soil-related factors [same unit as `yield_in`] |

# Module 6: Economic Suitability Analysis

## 1.37 Introduction

Economical Suitability Analysis Module is the most recent addition to AEZ framework (Figure 10). This module converts AEZ's final crop suitability (a result of the previous 5 modules) into an economic suitability. Additionally, all crops of interest are compared to the umbrella crop (crop with the highest economical potential) in order to indicate and map out its comparative advantage in terms of an attainable net revenue relative to the best available option. For more detailed calculations, refer to Module 6 chapter in National Agro-Economic Zoning for Major Crops in Thailand (NAEZ) report(FAO-UN, 2017).

First, we have to import the Module 6 Class and create an instance of that Class as below,

```
1 import EconomicSuitability
2 econ_su = EconomicSuitability.EconomicSuitability()
```

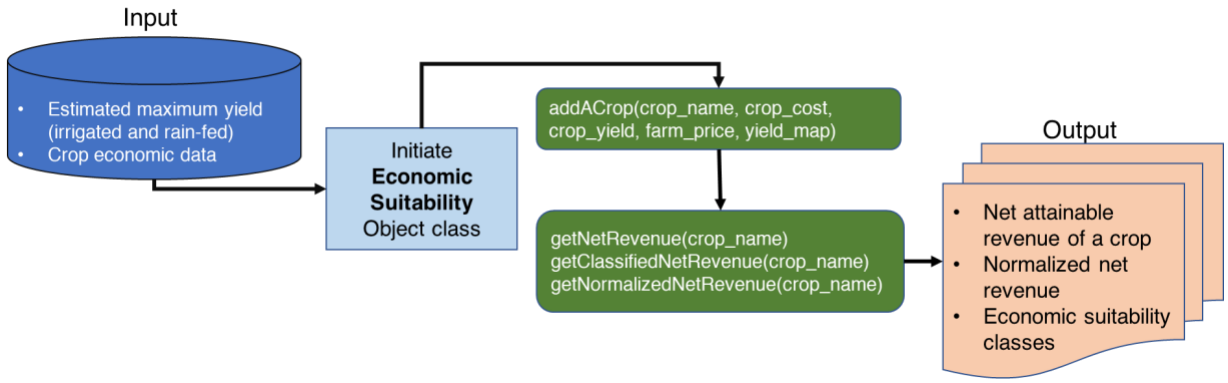


Figure 10. Overview of Module 6 (Economic Suitability) workflow.

## 1.38 Crop parameters inputs

This function allows users to set up the crop parameters for an economic analysis. The key inputs for Module 6 are the crop yield information generated from the previous 5 modules in PyAEZ, market prices, and the costs of the crop.

This function will be called multiple times as it performs a comparative economic analysis. This is a mandatory function to run before any further calculations.

```
1 # Crop parameter input
2 econ_su.addACrop(crop_name, crop_cost, crop_yield, farm_price, yield_map)
```

| Function Arguments | |
|---|---|
| `crop_name` | A single string value, corresponding to the crop name that you are adding.<br>This name will be used later to extract output for each crop. |
| `crop_cost` | • 1D NumPy array, corresponding to the cost of production for each yield values in crop yield variable.<br>• Values of `crop_cost` and `crop_yield` must be corresponding to each other, and they must be in ascending order.<br>• Units of this variable must be in **cost per hectare**.<br>• All the costs and prices in this module must be in same currency. |
| `crop_yield` | • 1D NumPy array, corresponding to the yield values.<br>• Values of `crop_cost` and `crop_yield` must be corresponding to each other, and they must be in ascending order.<br>• Units of this variable must be in **tonnes per hectare**. |
| `farm_price` | • 1D NumPy array, corresponding to the historical crop price that farmers sell.<br>• The price array is used to calculate distribution (mean) of prices.<br>• Unit: price (same currency throughout unit per tonne) |
| `yield_map` | 2D NumPy array, corresponding to yield map of the crop. Unit: **tonnes per hectare** |
| **Function Returns** | |
| None | |

## 1.39 Net Revenue

This function returns net revenue from the crop identified with 'crop_name'.

```
# Get the Net Revenue
crop_rev = econ_su.getNetRevenue(crop_name)
```

| Function Arguments | |
|---|---|
| `crop_name` | A single string value, corresponding to the crop name. |
| **Function Returns** | |
| `crop_rev` | 2D NumPy array, net revenue of the input `crop_name`. Unit: **`revenue per hectare`** |

## 1.40 Classified Net Revenue

This function returns classified net revenue for the crop 'crop_name'. The classification scheme for crop net revenue is outlined in Table 10.

```
# Net revenue classification
crop_rev_class = econ_su.getClassifiedNetRevenue(crop_name)
```

| Function Arguments | |
|---|---|
| `crop_name` | A single string value, corresponding to the crop name. |
| **Function Returns** | |
| `crop_rev_class` | 2D NumPy array, classified net revenue of the input `crop_name`. |

Table 10. Net Revenue Classification.

| Pixel value | Net Revenue Class | Description |
|---|---|---|
| 0 | Not suitable | Net revenue less than 0% |
| 1 | Very marginal | Net revenue between 0% and 10% |
| 2 | Marginal | Net revenue between 10% and 20% |
| 3 | Moderate | Net revenue between 20% and 30% |
| 4 | Medium | Net revenue between 40% and 50% |
| 5 | Good | Net revenue between 50% and 63% |
| 6 | High | Net revenue between 63% and 75% |
| 7 | Very high | Net revenue is equivalent to 75% or more than the overall maximum |

## 1.41 Normalized Net Revenue

This function returns the normalized net revenue for the crop 'crop_name'. The normalization is done, firstly, by assigning the highest possible net revenue, among crops passed through the module, to 1 (i.e., an umbrella crop). Secondly, the net revenue values of other crops are normalized as a portion of the umbrella crop (0-1 scale). This normalization process is performed separately for each pixel.

```
1  # Normalized net revenue
2  crop_rev_norm = econ_su.getNormalizedNetRevenue(crop_name)
```

| **Function Arguments** | |
|---|---|
| `crop_name` | A single string value, corresponding to the crop name. |
| **Function Returns** | |
| `crop_rev_norm` | 2D NumPy array, normalized net revenue of the input `crop_name`. Output values between `0` and `1`. |

# Utility Calculations

## 1.42 Introduction

This section will outline the additional calculation routines used throughout the PyAEZ's 6 main modules. These functions are contained within a Class called 'UtilitiesCalc'.

The functions are as follows:

| Functions in UtilitiesCalc | Description |
|---|---|
| `interpMonthlyToDaily` | Perform monthly-to-daily interpolation for climate data |
| `averageDailyToMonthly` | Aggregate daily climate data into monthly data |
| `generateLatitudeMap` | Generate latitude map as 2D NumPy array, by linearly interpolating the bottom and top latitudes of the study area |
| `classifyFinalYield` | Classify yield estimation and produce suitability map according to AEZ's classification scheme |
| `saveRaster` | Saving 2D NumPy arrays as GeoTIFF raster files |
| `averageRasters` | Averaging a list of rasters in the time-dimension |
| `windSpeedAt2m` | Convert windspeed from a particular altitude to 2m above the surface |

To use this UtilitiesCalc Class, we first must import and create a Class instance:

```
1  import UtilitiesCalc
2  obj_utilities = UtilitiesCalc.UtilitiesCalc()
```

## 1.43 Monthly-to-daily interpolation

This function performs interpolation of monthly climate data into daily climate data with quadratic spline interpolation as recommended in AEZ framework. The interpolation is performed between `cycle_begin` and `cycle_end` Julian dates.

```
1 # Monthly-to-daily interpolation
2 daily_vector = obj_utilities.interpMonthlyToDaily(monthly_vector, cycle_begin,
3 cycle_end, no_minus_values=False)
```

| Function Arguments | |
|---|---|
| `monthly_vector` | 1D NumPy array with 12 elements corresponding to the monthly climate data |
| `cycle_begin` | A single value corresponding to the beginning Julian date of the crop cycle |
| `cycle_end` | A single value corresponding to the ending Julian date of the crop cycle |

| | |
|---|---|
| `no_minus_values` | True or False. If this argument is True, negative values will be forced to be zero. This helps getting rid of any unrealistic negative interpolated values in the climate parameters such as precipitation data. If this argument is False, then negative values are allowed. By default, this argument is set as False and it's not a mandatory argument to pass. |
| **Function Returns** | |
| `daily_vector` | 1D NumPy array, corresponding to the output daily climate data between `cycle_begin` and `cycle_end` Julian dates. |

## 1.44 Daily-to-monthly aggregation

This function aggregates daily climate data into monthly climate data. The aggregation is done by averaging the data in each month.

```
# Daily-to-monthly aggregation
monthly_vector = obj_utilities.averageDailyToMonthly(daily_vector)
```

| **Function Arguments** | |
|---|---|
| `daily_vector` | 1D NumPy array with 365 elements corresponding to the daily climate data |
| **Function Returns** | |
| `monthly_vector` | 1D NumPy array with 12 elements corresponding to the aggregated monthly climate data |

## 1.45 Create latitude map

The latitude map is created by linearly interpolating the bottom and the top latitude values of the study area, as defined by the user's input.

```
# Generate latitude map
lat_map = obj_utilities.generateLatitudeMap(lat_min, lat_max, im_height, im_width)
```

| **Function Arguments** | |
|---|---|
| `lat_min` | A single value corresponding to the minimum latitude as decimal degree |
| `lat_max` | A single value corresponding to the maximum latitude as decimal degree |
| `im_height` | A single value corresponding to height of resulting latitude map as number of pixels. |
| `im_width` | A single value corresponding to width of resulting latitude map as number of pixels. |
| **Function Returns** | |
| `lat_map` | 2D NumPy array, corresponding to latitude map. The resulting dimension of the latitude map will be `im_height` and `im_width` respectively. |

## 1.46 Classify the final crop yield

This function classifies yield estimations and produces suitability maps according to classification scheme defined in AEZ framework. The classification scheme consists of 5 classes (very suitable, suitable, moderately suitable, marginally suitable, and not suitable) (Table 11).

```
# Classification of yield estimation
est_yield_class = obj_utilities.classifyFinalYield(est_yield)
```

| **Function Arguments** | |
|---|---|
| `est_yield` | 2D NumPy array corresponding to the estimated yield |
| **Function Returns** | |
| `est_yield_class` | 2D NumPy array, corresponding to the suitability map after yield classification |

Table 11. Yield suitability classification.

| Pixel value | Suitability Class | Description |
|---|---|---|
| 1 | Not suitable | yields between 0% and 20% of the overall maximum yield |
| 2 | Marginally suitable | yields between 20% and 40% of the overall maximum yield |
| 3 | Moderately suitable | yields between 40% and 60% of the overall maximum yield |
| 4 | Suitable | yields between 60% and 80% of the overall maximum yield |
| 5 | Very suitable | yields are equivalent to 80% or more of the overall maximum yield |

## 1.47 Saving GeoTIFF rasters

This function allows saving 2D numpy array as GeoTIFF raster file. This function can be used to save any output of this PyAEZ package as a GeoTIFF raster file.

```
# Save 2D NumPy to GeoTIFF
obj_utilities.saveRaster(ref_raster_path, out_path, numpy_raster)
```

| **Function Arguments** | |
|---|---|
| `ref_raster_path` | String, locating reference raster. This must be GeoTIFF raster file. Projection information is copied from this raster to final raster. Any input GeoTIFF raster to PyAEZ package with Projection information can be passed for this argument. |
| `out_path` | String, the desired location to save the output GeoTIFF file (with .tif extension) to. |
| `numpy_raster` | 2D NumPy array, corresponding to the raster that user wants to save. |
| **Function Returns** | |
| None | |

## 1.48 Averaging raster files

This function averages list of raster files in time dimension. Some calculations in the AEZ framework are recommended to perform with averaged climate data for 30 years. This function can be used for such calculations.

```
1 # Averaging raster files
2 avg_raster = obj_utilities.averageRaster(raster_3d)
```

| **Function Arguments** | |
|---|---|
| raster_3d | 3D NumPy array, corresponding to any climate data. The averaging will be done by the time dimension (across the years). |
| **Function Returns** | |
| avg_raster | 2D NumPy array, the averaged climate data – into 'one year' worth of data. |

## 1.49 Calculate wind speed at 2m altitude

This function converts wind speed from a particular altitude to wind speed at 2m altitude. All of the wind speed values used in PyAEZ calculations are at 2m altitude, however, it is common for climate data services to offer the wind speed at 10m altitude, hence this conversion.

```
1 # Converting to wind speed at 2m altitude
2 wind_speed_2m = obj_utilities.windSpeedAt2m(wind_speed, altitude)
```

| **Function Arguments** | |
|---|---|
| wind_speed | A NumPy array (can be 1D, 2D or 3D), corresponding to wind speed. |
| altitude | A single value corresponding to the altitude (above ground)[m] |
| **Function Returns** | |
| wind_speed_2m | Converted wind speed at 2m altitude as a NumPy array. Units will be same as unit of wind_speed. |

## References


Allen, R. G., Pereira, L. S., Raes, D., & Smith, M. (1998). Crop Evapotranspiration (guidelines for computing crop water requirements). *FAO Irrigation and Drainage Paper*, *56*.

de Wit, C. T. (1965). Photosynthesis of leaf canopies. *Agricultural Research Reports*, *663*.

FAO-UN. (2017). *National Agro-Economic Zoning for Major Crops in Thailand (NAEZ) (Project TCP/THA/3403) : NAEZ model implementation and results : final report*

FAO. (2017). The future of food and agriculture: trends and challenges. In D. Godoy, J. Dewbre, C. J. Amegnaglo, Y. Y. Soglo, A. F. Akpa, M. Bickel, S. Sanyang, S. Ly, J. Kuiseu, S. Ama, B. P. Gautier, R. Eberlin, E. Oduro-ofori, P. Aboagye Anokye, N. E. A. Acquaye, V. M. Dandelar, & J. Mineo (Eds.), *The future of food and agriculture: trends and challenges* (Vol. 4, Issue 4). Food and Agricultural Organisation of the United Nations.

Fischer, G., Nachtergaele, F., Velthuizen, H. van, Chiozza, F., Franceschini, G., Henry, M., Muchoney, D., & Tramberend, S. (2021). *Global Agro-Ecological Zones v4 – Model documentation*. Food and Agricultural Organisation of the United Nations. https://doi.org/10.4060/cb4744en

Kim, C. (2010). The Impact of Climate Change on the Agricultural Sector : Implications of the Agro - Industry for Low Carbon , Green Growth Strategy and Roadmap for the East Asian Region Table of Contents. *Low Carbon Green Growth Roadmap for Asia and the Pacific*, 1–51.

Nelson, G. C., Rosegrant, M. W., Koo, J., Robertson, R. D., Sulser, T., Zhu, T., Ringler, C., Msangi, S., Palazzo, A., Batka, M., Magalhaes, M., Valmonte-Santos, R., Ewing, M., & Lee, D. R. (2009). *Climate change: Impact on agriculture and costs of adaptation*. https://doi.org/10.2499/0896295354

Ray, D. K., Mueller, N. D., West, P. C., & Foley, J. A. (2013). Yield Trends Are Insufficient to Double Global Crop Production by 2050. *PLOS ONE*, *8*(6), e66428.

UNDESA. (2017). *World population projected to reach 9.8 billion in 2050, and 11.2 billion in 2100.* United Nations Department of Economic and Social Affairs.

WEC. (2013). *World Energy Scenarios Composing energy futures to 2050*.