BestiaPop

Release 1.0

Jun 07, 2021

Contents

1 How to use BestiaPop **3**
- 1.1 Examples . . . 3
  - 1.1.1 Generate Climate File . . . 3
  - 1.1.2 Download NetCDF4 File . . . 4
- 1.2 PARALLEL COMPUTING . . . 5
  - 1.2.1 EXAMPLE: Generate MET output files from SILO Cloud API for Radiation, Min Temperature, Max Temperature and Daily Rain for years 2015 to 2016 . . . 5

2 API Reference **7**
- 2.1 Modules . . . 7
- 2.2 Base Classes . . . 11

3 BestiaPop: A python package to automate the extraction and processing of climate data for crop modelling **15**
- 3.1 Authors . . . 15
  - 3.1.1 Core Contributors . . . 15
  - 3.1.2 Acknowledgements . . . 16
  - 3.1.3 Other Contributors . . . 16
  - 3.1.4 More information . . . 16

4 Crop Modelling Software **17**
- 4.1 APSIM: Agricultural Production Systems Simulator . . . 17
  - 4.1.1 What is APSIM? . . . 17
  - 4.1.2 What is a MET file? . . . 17
  - 4.1.3 Can I use this script to generate climate files for other process-based crop models? . . . 18
- 4.2 DSSAT: ??? . . . 18

5 About SILO **19**
- 5.1 NetCDF and API data variations from SILO . . . 19
- 5.2 More information . . . 20

6 Installation **21**
- 6.1 Using Anaconda (preferred) . . . 21
- 6.2 Using pip . . . 21

7 Main References **23**
- 7.1 Package references . . . 23

Python Module Index **25**

Index **27**

Date: Jun 07, 2021 **Version**: 3.0

CHAPTER 1

How to use BestiaPop

BestiaPop has three primary commands that you can pass in with the `-a` option:

1. `generate-climate-file`: this command will generate an input file for crop modelling software depending on the output type (`-ot`) being `met` or `wth`. When `csv` is selected, a file containing all years in the sequence, with all requested variables, will be produced for each lat/lon combination.
2. `download-nc4-file`: this command downloads NetCDF4 files from SILO or NASAPOWER
3. `convert-nc4`: *currently not implemented*, this command will allow you to convert NetCDF4 files to other formats like `json` or `csv`.

1.1 Examples

1.1.1 Generate Climate File

Generate MET output files using SILO cloud API, for Radiation, Min Temperature, Max Temperature and Daily Rain for years 2015 to 2016

NOTE:

- the resulting MET files will be placed in the output directory specified by "-o"
- the "-ot" parameter specifies the output type: met, dssat, csv or json.

```
python bestiapop.py -a generate-climate-file -s silo -y "2015-2016" -c "radiation max_
↪temp min_temp daily_rain" -lat "-41.15 -41.05" -lon "145.5 145.6" -o␣
↪C:\some\output\folder\ -ot met
```

Generate WTH (for DSSAT) output files using SILO cloud API, for Radiation, Min Temperature, Max Temperature and Daily Rain for years 2015 to 2016

```
python bestiapop.py -a generate-climate-file -s silo -y "2015-2016" -c "radiation max_
↪temp min_temp daily_rain" -lat "-41.15 -41.05" -lon "145.5 145.6" -o␣
↪C:\some\output\folder\ -ot wth
```

Generate MET output files using NASAPOWER cloud API, for Radiation, Min Temperature, Max Temperature and Daily Rain for years 2003 to 2016

```
python bestiapop.py -a generate-climate-file -s nasapower -y "2015-2016" -c
↪"radiation max_temp min_temp daily_rain" -lat "-41.15 -41.05" -lon "145.5 145.6" -o␣
↪C:\some\output\folder\ -ot met
```

Generate MET output files from Local Disk for Radiation, Min Temperature, Max Temperature and Daily Rain for years 1990 to 2010

NOTE:

- all the required NetCDF files should be placed in a single directory which is then referenced with the *–input* parameter. The directory should have the following structure:

```
C:\input\folder:
    \__ 1990.daily_rain.nc
    \__ 1990.max_temp.nc
    \__ 1990.min_temp.nc
    \__ 1990.radiation.nc
    \__ 1991.daily_rain.nc
    \__ 1991.max_temp.nc
    \__ 1991.min_temp.nc
    \__ 1991.radiation.nc
    \__ ...
    \__ 2010.daily_rain.nc
    \__ 2010.max_temp.nc
    \__ 2010.min_temp.nc
    \__ 2010.radiation.nc
```

```
python bestiapop.py -a generate-climate-file -y "1990-2010" -c "radiation max_temp␣
↪min_temp daily_rain" -lat "-41.15 -41.05" -lon "145.5 145.6" -i␣
↪C:\some\input\folder\with\all\netcdf\files\ -o C:\some\output\folder\ -ot met
```

1.1.2 Download NetCDF4 File

Download SILO climate files for years 2010 to 1028 and the variables daily_rain and max_temp

This command will **only** download the file from AWS, it won't perform any further processing.

NOTE: *a year range must be separated by a dash, whereas multiple climate variables are separated by spaces*

```
python bestiapop.py -a download-nc4-file --data-source silo -y "2010-2018" -c "daily_
↪rain max_temp" -o C:\some\output\folder
```

1.2 PARALLEL COMPUTING

BestiaPop as of version 2.5 comes with parallel processing for multicore systems by leveraging python's multiprocessing library. Not all actions have implemented this functionality yet but they will be added progressively. To enable multiprocessing just pass in the `-m` flag to the `bestiapop.py` command. By default it will leverage **all your cores** (whether physical or logical).

Parallelization is done based on the coordinate variable (whether *latitude* or *longitude*) that has the widest spread (highest data points count). This means, you will rip the benefits of multiple cores when your datasets reference wide ranges of *lat* or *lon* variables.

> **NOTE**: When generating *MET* files *from locally available NetCDF4 files* based on SILO data, you might experience mixed results since SILO provides NetCDF4 files split into `year-variable` and *MET* files require multiple *years* in the same MET file. SILO has created the NetCDF4 files (as of 2020) to perform better when extracting spatial data points rather than time-based data points. This effectively means that it is slower to extract data **for all days of the year** from NetCDF4 files, for a single combination of lat/lon, than it is to extract data for all combinations of lat/lon **for a single day**. Since SILO NetCDF4 files are split into `year-variable` units you will always have to extract data from different files when using multiple years.

1.2.1 EXAMPLE: Generate MET output files from SILO Cloud API for Radiation, Min Temperature, Max Temperature and Daily Rain for years 2015 to 2016

```
python bestiapop.py -a generate-climate-file -s silo -y "2008-2016" -c "radiation max_
↪temp min_temp daily_rain" -lat "-41.15 -41.05" -lon "145.5 145.6" -o␣
↪C:\some\output\folder\ -m
```

Here, the `-m` at the end will engage multiple cores to process the tasks. If you have 8 available cores it will create 8 separate processes to download the data from the cloud and will then use 8 separate processes to generate the output files.

CHAPTER 2

API Reference

2.1 Modules

class common.bestiapop_utils.**MyUtilityBeast**(*input_path=None*)

This class will provide methods to perform generic or shared operations on data

Parameters **logger** (*str*) – A pointer to an initialized Argparse logger

download_nc4_file_from_cloud(*year*, *climate_variable*, *output_path=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/bestiapop/cl*
data_source='silo', *skip_certificate_checks=False*)

Downloads a file from AWS S3 bucket or other cloud API

Parameters

- **year** (*int*) – the year we require data for. SILO stores climate data as separate years like so: daily_rain.2018.nc
- **climate_variable** (*str*) – the climate variable short name as per SILO nomenclature, see https://www.longpaddock.qld.gov.au/silo/about/climate-variables/
- **output_path** (*str, optional*) – The target folder where files should be downloaded. Defaults to Path().cwd().
- **skip_certificate_checks** (*bool, optional*) – ask the requests library to skip certificate checks, useful when attempting to download files behind a proxy. Defaults to False.

generate_climate_dataframe_from_disk(*year_range*, *climate_variables*, *lat_range*, *lon_range*, *input_dir*, *data_source='silo'*)

This function generates a dataframe containing (a) climate values (b) for every variable requested (c) for every day of the year (d) for every year passed in as argument. The values will be sourced from Disk. :param year_range: a numpy array with all the years for which we are seeking data. :type year_range: numpy.ndarray :param climate_variables: the climate variable short name as per SILO or NASAPOWER nomenclature. For SILO check https://www.longpaddock.qld.gov.au/silo/about/climate-variables/. For NASAPOWER check: XXXXX. :type climate_variables: str :param lat_range: a numpy array of latitude values to extract data from :type lat_range: numpy.ndarray :param lon_range: a numpy array of longitude

values to extract data from :type lon_range: numpy.ndarray :param input_dir: when selecting the option to generate Climate Data Files from local directories, this parameter must be specified, otherwise data will be fetched directly from the cloud either via an available API or S3 bucket. :type input_dir: str

Returns a tuple consisting of (a) the final dataframe containing values for all years, latitudes and longitudes for a particular climate variable, (b) the curated list of longitude ranges (which excludes all those lon values where there were no actual data points). The tuple is ordered as follows: (final_dataframe, final_lon_range)

Return type tuple

load_cdf_file(*sourcepath*, *data_category*, *year=None*)

This function loads a NetCDF4 file either from the cloud or locally

Parameters

- **sourcepath** (*str*) – when loading a NetCDF4 file locally, this specifies the source folder. Only the "folder" must be specified, the actual file name will be further qualified by BestiaPop grabbing data from the year and climate variable paramaters passed to the SILO class.
- **data_category** (*str*) – the short name variable, examples: daily_rain, max_temp, etc.
- **year** (*int, optional*) – the year we want to extract data from, it is used to compose the final AWS S3 URL or to qualify the full path to the local NetCDF4 file we would like to load. Defaults to None.

Returns a dictionary containing two items, "value_array" which is a xarray DataSet object and "data_year" which is the year that the NetCDF4 file contains data for, extracted by looking at the contents of the NetCDF4 file itself.

Return type dict

class connectors.silo_connector.**SILOClimateDataConnector**(*climate_variables*, *data_source='silo'*, *input_path=None*)

This class will provide methods that query and parse data from SILO climate database

Parameters

- **logger** (*str*) – A pointer to an initialized Argparse logger
- **data_source** (*str*) – The climate database where the values are being extracted from: SILO or NASAPOWER

generate_climate_dataframe_from_silo_cloud_api(*year_range*, *climate_variables*, *lat_range*, *lon_range*, *input_dir*)

This function generates a dataframe containing (a) climate values (b) for every variable requested (c) for every day of the year (d) for every year passed in as argument. It will leverage SILO API to do it.

Parameters

- **year_range** (*numpy.ndarray*) – a numpy array with all the years for which we are seeking data.
- **climate_variables** (*str*) – the climate variable short name as per SILO or NASAPOWER nomenclature. For SILO check https://www.longpaddock.qld.gov.au/silo/about/climate-variables/. For NASAPOWER check: XXXXX.
- **lat_range** (*numpy.ndarray*) – a numpy array of latitude values to extract data from
- **lon_range** (*numpy.ndarray*) – a numpy array of longitude values to extract data from

- **input_dir** (*str*) – when selecting the option to generate Climate Data Files from local directories, this parameter must be specified, otherwise data will be fetched directly from the cloud either via an available API or S3 bucket.

Returns a tuple consisting of (a) the final dataframe containing values for all years, latitudes and longitudes for a particular climate variable, (b) the curated list of longitude ranges (which excludes all those lon values where there were no actual data points). The tuple is ordered as follows: (final_dataframe, final_lon_range)

Return type tuple

get_yearly_data(*lat*, *lon*, *value_array*, *year*, *year_range*, *climate_variable*)

Extract values from an API endpoint in the cloud or a xarray.Dataset object

Parameters

- **lat** (*float*) – the latitude that values should be returned for
- **lon** (*float*) – the longitude that values should be returned for
- **value_array** (*xarray.Dataset*) – the xarray Dataset object to extract values from
- **year** (*string*) – the year of the file
- **variable_short_name** (*string*) – the climate variable short name as per SILO nomenclature, see https://www.longpaddock.qld.gov.au/silo/about/climate-variables/

Raises ValueError – if there was "NO" data available for all days under a particular combination of lat & lon, then the total values collected should equal "0" (meaning, there was no data for that point in the grid). If this is the case, then the function will simply return with a "no_values" message and signal the calling function that it should ignore this particular year-lat-lon combination.

Returns a dataframe containing 5 columns: the Julian day, the grid data value for that day, the year, the latitude, the longitude.

Return type pandas.core.frame.DataFrame

class connectors.nasapower_connector.**NASAPowerClimateDataConnector**(*climate_variables*, *data_source='silo'*, *input_path=None*)

This class will provide methods that query and parse data from NASA POWER climate database

Parameters

- **logger** (*str*) – A pointer to an initialized Argparse logger
- **data_source** (*str*) – The climate database where the values are being extracted from: SILO or NASAPOWER

generate_climate_dataframe_from_nasapower_cloud_api(*year_range*, *climate_variables*, *lat_range*, *lon_range*, *input_dir*)

This function generates a dataframe containing (a) climate values (b) for every variable requested (c) for every day of the year (d) for every year passed in as argument. It will leverage NASAPOWER API to do it.

Parameters

- **year_range** (*numpy.ndarray*) – a numpy array with all the years for which we are seeking data.

- **climate_variables** (*str*) – the climate variable short name as per SILO nomenclature. For SILO check https://www.longpaddock.qld.gov.au/silo/about/climate-variables/. Variable names are automatically translated from SILO to NASAPOWER codes.
- **lat_range** (*numpy.ndarray*) – a numpy array of latitude values to extract data from
- **lon_range** (*numpy.ndarray*) – a numpy array of longitude values to extract data from
- **input_dir** (*str*) – when selecting the option to generate Climate Data Files from local directories, this parameter must be specified, otherwise data will be fetched directly from the cloud either via an available API or S3 bucket.

Returns a tuple consisting of (a) the final dataframe containing values for all years, latitudes and longitudes for a particular climate variable, (b) the curated list of longitude ranges (which excludes all those lon values where there were no actual data points). The tuple is ordered as follows: (final_dataframe, final_lon_range)

Return type tuple

get_yearly_data(*lat*, *lon*, *value_array*, *year*, *year_range*, *climate_variable*)

Extract values from an API endpoint in the cloud or a xarray.Dataset object

Parameters

- **lat** (*float*) – the latitude that values should be returned for
- **lon** (*float*) – the longitude that values should be returned for
- **value_array** (*xarray.Dataset*) – the xarray Dataset object to extract values from
- **year** (*string*) – the year of the file
- **variable_short_name** (*string*) – the climate variable name

Raises ValueError – if there was "NO" data available for all days under a particular combination of lat & lon, then the total values collected should equal "0" (meaning, there was no data for that point in the grid). If this is the case, then the function will simply return with a "no_values" message and signal the calling function that it should ignore this particular year-lat-lon combination.

Returns a dataframe containing 5 columns: the Julian day, the grid data value for that day, the year, the latitude, the longitude.

Return type pandas.core.frame.DataFrame

The NASA POWER database is a global database of daily weather data specifically designed for agrometeorological applications. The spatial resolution of the database is 0.5x0.5 degrees (as of 2018). For more information on the NASA POWER database see the documentation at: http://power.larc.nasa.gov/common/AgroclimatologyMethodology/Agro_Methodology_Content.html The *NASAPowerClimateDataConnector* is used by BestiaPop to retrieve data from NASA POWER database and provides functions to parse and extract relevant information from it. Important NOTE: as per https://power.larc.nasa.gov/docs/services/api/v1/temporal/daily/, any latitude-longitude combinations within a 0.5x0.5 degrees grid box will yield the same weather data. Thus, there is no difference for data returned between lat/lon -41.5/145.3 and lat/lon -41.8/145.7. When BestiaPop requests data from NASA Power, it will automatically create coordinate series wiht 1 degree jumps. So if you pass in *-lat "-41.15 -55.05"* the resulting series will be: [-55.05, -54.05, -53.05, -52.05, -51.05, -50.05, -49.05, -48.05, -47.05, -46.05, -45.05, -44.05, -43.05, -42.05, -41.05]. Please bear in mind that there is no difference between -41.15 and -41.05.

2.2 Base Classes

class producers.output.**DATAOUTPUT**(*data_source*)

This class will provide different methods for data output from climate dataframes

Parameters

- **logger** (*str*) – A pointer to an initialized Argparse logger
- **data_source** (*str*) – The climate database where the values are being extracted from: SILO or NASAPOWER

Returns A class object with access to DATAOUTPUT methods

Return type *DATAOUTPUT*

generate_met(*outputdir*, *met_dataframe*, *lat*, *lon*)

Generate APSIM MET File

Parameters

- **outputdir** (*str*) – the folder where the generated MET files will be stored
- **met_dataframe** (*pandas.core.frame.DataFrame*) – the pandas dataframe slice to convert to MET file
- **lat** (*float*) – the latitude for which this MET file is being generated
- **lon** (*float*) – the longitude for which this MET file is being generated

generate_output(*final_daily_df*, *lat_range*, *lon_range*, *outputdir=None*, *output_type='met'*)

Generate required Output based on Output Type selected

Parameters

- **final_daily_df** (*pandas.core.frame.DataFrame*) – the pandas daframe containing all the values that are going to be parsed into a specific output
- **lat_range** (*numpy.ndarray*) – an array of latitude values to select from the final_daily_df
- **lon_range** (*numpy.ndarray*) – an array of longitude values to select from the final_daily_df
- **outputdir** (*str*) – the folder that will be used to store the output files
- **output_type** (*str, optional*) – the output type: csv (not implemented yet), json(not implemented yet), met. Defaults to "met".

generate_wth(*outputdir*, *wth_dataframe*, *lat*, *lon*)

Generate WTH File

Parameters

- **outputdir** (*str*) – the folder where the generated WTH files will be stored
- **wth_dataframe** (*pandas.core.frame.DataFrame*) – the pandas dataframe slice to convert to WTH file
- **lat** (*float*) – the latitude for which this WTH file is being generated
- **lon** (*float*) – the longitude for which this WTH file is being generated

class connectors.silo_connector.**SILOClimateDataConnector**(*climate_variables*, *data_source='silo'*, *input_path=None*)

This class will provide methods that query and parse data from SILO climate database

Parameters

- **logger** (*str*) – A pointer to an initialized Argparse logger
- **data_source** (*str*) – The climate database where the values are being extracted from: SILO or NASAPOWER

generate_climate_dataframe_from_silo_cloud_api(*year_range*, *climate_variables*, *lat_range*, *lon_range*, *input_dir*)

This function generates a dataframe containing (a) climate values (b) for every variable requested (c) for every day of the year (d) for every year passed in as argument. It will leverage SILO API to do it.

Parameters

- **year_range** (*numpy.ndarray*) – a numpy array with all the years for which we are seeking data.
- **climate_variables** (*str*) – the climate variable short name as per SILO or NASAPOWER nomenclature. For SILO check https://www.longpaddock.qld.gov.au/silo/about/climate-variables/. For NASAPOWER check: XXXXX.
- **lat_range** (*numpy.ndarray*) – a numpy array of latitude values to extract data from
- **lon_range** (*numpy.ndarray*) – a numpy array of longitude values to extract data from
- **input_dir** (*str*) – when selecting the option to generate Climate Data Files from local directories, this parameter must be specified, otherwise data will be fetched directly from the cloud either via an available API or S3 bucket.

Returns a tuple consisting of (a) the final dataframe containing values for all years, latitudes and longitudes for a particular climate variable, (b) the curated list of longitude ranges (which excludes all those lon values where there were no actual data points). The tuple is ordered as follows: (final_dataframe, final_lon_range)

Return type tuple

get_yearly_data(*lat*, *lon*, *value_array*, *year*, *year_range*, *climate_variable*)

Extract values from an API endpoint in the cloud or a xarray.Dataset object

Parameters

- **lat** (*float*) – the latitude that values should be returned for
- **lon** (*float*) – the longitude that values should be returned for
- **value_array** (*xarray.Dataset*) – the xarray Dataset object to extract values from
- **year** (*string*) – the year of the file
- **variable_short_name** (*string*) – the climate variable short name as per SILO nomenclature, see https://www.longpaddock.qld.gov.au/silo/about/climate-variables/

Raises ValueError – if there was "NO" data available for all days under a particular combination of lat & lon, then the total values collected should equal "0" (meaning, there was no data for that point in the grid). If this is the case, then the function will simply return with a "no_values" message and signal the calling function that it should ignore this particular year-lat-lon combination.

Returns a dataframe containing 5 columns: the Julian day, the grid data value for that day, the year, the latitude, the longitude.

Return type pandas.core.frame.DataFrame

class connectors.nasapower_connector.**NASAPowerClimateDataConnector**(*climate_variables*, *data_source='silo'*, *input_path=None*)

This class will provide methods that query and parse data from NASA POWER climate database

Parameters

- **logger** (*str*) – A pointer to an initialized Argparse logger
- **data_source** (*str*) – The climate database where the values are being extracted from: SILO or NASAPOWER

generate_climate_dataframe_from_nasapower_cloud_api(*year_range*, *climate_variables*, *lat_range*, *lon_range*, *input_dir*)

This function generates a dataframe containing (a) climate values (b) for every variable requested (c) for every day of the year (d) for every year passed in as argument. It will leverage NASAPOWER API to do it.

Parameters

- **year_range** (*numpy.ndarray*) – a numpy array with all the years for which we are seeking data.
- **climate_variables** (*str*) – the climate variable short name as per SILO nomenclature. For SILO check https://www.longpaddock.qld.gov.au/silo/about/climate-variables/. Variable names are automatically translated from SILO to NASAPOWER codes.
- **lat_range** (*numpy.ndarray*) – a numpy array of latitude values to extract data from
- **lon_range** (*numpy.ndarray*) – a numpy array of longitude values to extract data from
- **input_dir** (*str*) – when selecting the option to generate Climate Data Files from local directories, this parameter must be specified, otherwise data will be fetched directly from the cloud either via an available API or S3 bucket.

Returns a tuple consisting of (a) the final dataframe containing values for all years, latitudes and longitudes for a particular climate variable, (b) the curated list of longitude ranges (which excludes all those lon values where there were no actual data points). The tuple is ordered as follows: (final_dataframe, final_lon_range)

Return type tuple

get_yearly_data(*lat*, *lon*, *value_array*, *year*, *year_range*, *climate_variable*)

Extract values from an API endpoint in the cloud or a xarray.Dataset object

Parameters

- **lat** (*float*) – the latitude that values should be returned for
- **lon** (*float*) – the longitude that values should be returned for
- **value_array** (*xarray.Dataset*) – the xarray Dataset object to extract values from
- **year** (*string*) – the year of the file
- **variable_short_name** (*string*) – the climate variable name

Raises `ValueError` – if there was "NO" data available for all days under a particular combination of lat & lon, then the total values collected should equal "0" (meaning, there was no data for that point in the grid). If this is the case, then the function will simply return with a "no_values" message and signal the calling function that it should ignore this particular year-lat-lon combination.

Returns a dataframe containing 5 columns: the Julian day, the grid data value for that day, the year, the latitude, the longitude.

Return type pandas.core.frame.DataFrame

The NASA POWER database is a global database of daily weather data specifically designed for agrometeorological applications. The spatial resolution of the database is 0.5x0.5 degrees (as of 2018). For more information on the NASA POWER database see the documentation at: http://power.larc.nasa.gov/common/AgroclimatologyMethodology/Agro_Methodology_Content.html The *NASAPowerClimateDataConnector* is used by BestiaPop to retrieve data from NASA POWER database and provides functions to parse and extract relevant information from it. Important NOTE: as per https://power.larc.nasa.gov/docs/services/api/v1/temporal/daily/, any latitude-longitude combinations within a 0.5x0.5 degrees grid box will yield the same weather data. Thus, there is no difference for data returned between lat/lon -41.5/145.3 and lat/lon -41.8/145.7. When BestiaPop requests data from NASA Power, it will automatically create coordinate series wiht 1 degree jumps. So if you pass in *-lat "-41.15 -55.05"* the resulting series will be: [-55.05, -54.05, -53.05, -52.05, -51.05, -50.05, -49.05, -48.05, -47.05, -46.05, -45.05, -44.05, -43.05, -42.05, -41.05]. Please bear in mind that there is no difference between -41.15 and -41.05.

CHAPTER 3

BestiaPop: A python package to automate the extraction and processing of climate data for crop modelling

Climate data is an essential input for crop models to predict crop growth and development using site-specific (point) or gridded climate data. However, *Crop Modelling* software expects input data to be encapsulated in custom file formats (`MET`, `WTH`, etc.) which don't conform to a common standard and require various customizations, depending on the prediction engine that generates crop models. Moreover, source data providers like SILO or NASA POWER are usually neutral in the type of data output files they provide as part of their API services which leads to a gap between source *raw* data and *processed* data required by crop modelling suites to develop their models. We developed **BestiaPop** (a spanish word that translates to *pop beast*), a Python package which allows model users to automatically download SILO's (Scientific Information for Land Owners) or NASAPOWER gridded climate data and convert this data to files that can be ingested by *Crop Modelling* software like APSIM or DSSAT.

The package offers the possibility to select a range of grids (5 km × 5 km resolution) and years producing various types of output files: CSV, MET (for APSIM), WTH (for DSSAT) and soon JSON (which will become part of BestiaPop's API in the future).

Curently, the code downloads data from two different climate databases:

1. SILO
2. NASA POWER

3.1 Authors

3.1.1 Core Contributors

- **Data Analytics Specialist & Code Developer**: Diego Perez (@darkquassar / https://linkedin.com/in/diegope)
- **Data Scientist & Agricultural Systems Modeller**: Jonathan Ojeda (@JJguri / https://www.jojeda.com/)

3.1.2 Acknowledgements

- This work was supported by the JM Roberts Seed Funding for Sustainable Agriculture 2020 and the Tasmanian Institute of Agriculture, University of Tasmania.

- SILO (Scientific Information for Land Owners), see: https://www.longpaddock.qld.gov.au/silo/about/
- NASAPOWER, see: https://power.larc.nasa.gov

3.1.3 Other Contributors

- Drew Holzworth ([helping](https://github.com/APSIMInitiative/ApsimX/issues/5423) integrate BestiaPop into APSIM, kudos!)

3.1.4 More information

- https://www.jojeda.com/project/project-6/

CHAPTER 4

Crop Modelling Software

There are two major crop modelling suites in use by the scientific community in Australia: APSIM and DSSAT.

[yoni to expand]

4.1 APSIM: Agricultural Production Systems Simulator

4.1.1 What is APSIM?

The Agricultural Production Systems sIMulator (APSIM) is internationally recognised as a highly advanced platform for modelling and simulation of agricultural systems. It contains a suite of modules that enable the simulation of systems for a diverse range of crop, animal, soil, climate and management interactions. APSIM is undergoing continual development, with new capability added to regular releases of official versions. Its development and maintenance is underpinned by rigorous science and software engineering standards. The APSIM Initiative has been established to promote the development and use of the science modules and infrastructure software of APSIM.

4.1.2 What is a MET file?

The APSIM MET module provided daily meteorological information to all modules within an APSIM simulation. The APSIM Met Module requires parameters to specify the climate of the site for each APSIM time step. This information is included in a MET file.

APSIM MET files consist of a section name, which is always *weather.met.weather*, several constants consisting of *name* = *value*, followed by a headings line, a units line and then the data. Spacing in the file is not relevant. Comments can be inserted using the ! character.

At a minimum three constants must be included in the file: **latitude**, **tav** and **amp**. The last two of these refer to the annual average ambient temperature and annual amplitude in mean monthly temperature. Full details about tav and amp can be found here: tav_amp.

The MET file must also have a year and day column (or date formatted as *yyyy/mm/dd*), solar radiation (*MJ/m2*), maximum temperature (*°C*), minimum temperature (*°C*) and rainfall (*mm*). The column headings to use for

these are year and day (or date), radn, maxt, mint, rain. Other constants or columns can be added to the file. These then become available to APSIM as variables that can be reported or used in manager script.

While *point* data is usually available in MET format at the SILO webpage, *gridded data* is provided in NetCDF file format which is difficult to store and convert to an input file readable by APSIM or other crop models. **BestiaPop** takes care of generating the required input files for APSIM.

4.1.3 Can I use this script to generate climate files for other process-based crop models?

[yoni to expand/fix]

So far, the code is producing CSV or MET files to be directly used by APSIM, however, it also could be applied to produce input climate data for other crop models such as DSSAT and STICS. Decision Support System for Agrotechnology Transfer (DSSAT) is a software application program that comprises dynamic crop growth simulation models for over 40 crops. DSSAT is supported by a range of utilities and apps for weather, soil, genetic, crop management, and observational experimental data, and includes example data sets for all crop models. The STICS (Simulateur mulTIdisciplinaire pour les Cultures Standard, or multidisciplinary simulator for standard crops) model is a dynamic, generic and robust model aiming to simulate the soil-crop-atmosphere system.

4.2 DSSAT: ???

[yoni to expand] –> we need a section similar to what we wrote for APSIM but for DSSAT I guess...

CHAPTER 5

About SILO

SILO is a database of Australian climate data from 1889 to the present. It provides daily meteorological datasets for a range of climate variables in ready-to-use formats suitable for biophysical modelling, research and climate applications.

5.1 NetCDF and API data variations from SILO

There are particular years where some data may be different when you read the data from a `NetCDF SILO file`, as opposed to reading the same data from the `SILO API`. Below is a detailed description of the SILO team as to why this might happen.

When you request point data at grid cell locations, the data is extracted from the relevant grid cell in the NetCDF files. This data is then passed through a simple filter that checks if each datum is within an acceptable range:

- daily rainfall: 0 – 1300 mm
- maximum temperature: -9 - 54 C
- minimum temperature: -20 - 40 C
- class A pan evaporation: 0 - 35 mm
- solar radiation: 0 - 35 MJ/m2
- vapour pressure: 0 - 43.2 hPa
- maximum temperature > minimum temperature

If a given datum (or pair of data values for the Tmax > Tmin check) fails the check, it/they may be erroneous so SILO provides the long term daily mean(s) instead. This is represented by a number *75* in the value for a particular datum.

If you request data at station locations the same checks are done; the main difference is observed data are provided where possible, and gridded data are provided if observed data are not available on a given day(s).

> Differences between the API and NetCDF values only occur when a datum fails one of the aforementioned range checks, for example, when the interpolated maximum temperature is lower than the interpolated minimum temperature. Such situations typically arise due to errors in the observed data (leading to errors

in the gridded surface), or in regions where there are very few recording stations. We expect there to be more errors in the gridded surfaces for the early years, as there were relatively few stations recording data (other than rainfall) before 1957. Plots showing the number of stations recording each variable as a function of time are provided in our 2001 paper (see the Publications section on SILO).

5.2 More information

https://www.jojeda.com/project/project-6/

CHAPTER 6

Installation

1. Clone this repo
2. Install required packages.

6.1 Using Anaconda (preferred)

1. `conda create -y --name bestiapop python==3.7`
2. `conda install --name bestiapop -y -c conda-forge --file requirements.txt`
3. `conda activate bestiapop`

This option might take a *very* long time due to the multiple dependencies that Anaconda might have to solve on your default **base environment**. Preferably, install using the next method which creates a custom environment.

1. Open your anaconda prompt for the *base* environment (default)
2. `conda install -y -c conda-forge --file require`

6.2 Using pip

1. Change directory to the repo folder
2. `pip install -r requirements.txt`

CHAPTER 7

Main References

The following papers implemented this code and can be used as references

1. Ojeda JJ, Eyshi Rezaei E, Remeny TA, Webb MA, Webber HA, Kamali B, Harris RMB, Brown JN, Kidd DB, Mohammed CL, Siebert S, Ewert F, Meinke H (2019) Effects of soil- and climate data aggregation on simulated potato yield and irrigation water demand. Science of the Total Environment. 710, 135589. doi:10.1016/j.scitotenv.2019.135589
2. Ojeda JJ, Perez D, Eyshi Rezaei E (2020) The BestiaPop - A Python package to automatically generate gridded climate data for crop models. APSIM Symposium, Brisbane, Australia.

7.1 Package references

1. https://registry.opendata.aws/silo/
2. https://towardsdatascience.com/handling-netcdf-files-using-xarray-for-absolute-beginners-111a8ab4463f
3. http://xarray.pydata.org/en/stable/dask.html

Python Module Index

b

bestiapop, 1

c

common.bestiapop_utils, 7
connectors.nasapower_connector, 9
connectors.silo_connector, 8

Index

B

bestiapop (*module*), 1

C

common.bestiapop_utils (*module*), 7
connectors.nasapower_connector (*module*), 9
connectors.silo_connector (*module*), 8

D

DATAOUTPUT (*class in producers.output*), 11
download_nc4_file_from_cloud() (*common.bestiapop_utils.MyUtilityBeast method*), 7

G

generate_climate_dataframe_from_disk() (*common.bestiapop_utils.MyUtilityBeast method*), 7
generate_climate_dataframe_from_nasapower_cloud_api() (*connectors.nasapower_connector.NASAPowerClimateDataConnector method*), 9, 13
generate_climate_dataframe_from_silo_cloud_api() (*connectors.silo_connector.SILOClimateDataConnector method*), 8, 12
generate_met() (*producers.output.DATAOUTPUT method*), 11
generate_output() (*producers.output.DATAOUTPUT method*), 11
generate_wth() (*producers.output.DATAOUTPUT method*), 11
get_yearly_data() (*connectors.nasapower_connector.NASAPowerClimateDataConnector method*), 10, 13
get_yearly_data() (*connectors.silo_connector.SILOClimateDataConnector method*), 9, 12

L

load_cdf_file() (*common.bestiapop_utils.MyUtilityBeast method*), 8

M

MyUtilityBeast (*class in common.bestiapop_utils*), 7

N

NASAPowerClimateDataConnector (*class in connectors.nasapower_connector*), 9, 13

S

SILOClimateDataConnector (*class in connectors.silo_connector*), 8, 11