Calibration Notebook Example

December 16, 2025

1 MaSim Country Calibration

This note book is used for running the country calibration processes for eventual use in the experimental simulation process where various response strategies are modeled. This notebook and the accompanying toolkit was developed by James Brodovsky and Sarit Adhikari as part of the calibration efforts for Burkino Faso and Mozambique in 2025. This notebook is ***version two*** and is being developed alongside the calibration of Angola and a design shift around batch processing on the Nessun Dorma cluster that the lab controls. The Temple University Owls Nest cluster will be decommissioned sometime in 2026 as such there will be a shift in how batch processing is handled.

This Python package is developed with Python 3.10+ and is compliant with PEP 621. It is recommended to use a virtual environment manager such as `uv` or `venv` to manage dependencies. The package dependencies are listed in the `pyproject.toml` file. While any compliant environment manager should work, `uv` is recommended and the officially supported environment manager.

1.1 Installation

Installation is in two parts. First, clone this repository to your local machine. Second, install the `masim_analysis` package into a virtual environment, e.g., using `pip` or `uv`.

```
pip install -e .
```

If you are using `uv` (and you should!) install via

```
uv pip install -e .
```

This will install the package in an editable mode, allowing you to make changes to the code and have them reflected in your local environment. If for some reason you are not seeing the changes reflected, try re-installing the package locally using the above commands again.

```
from src.masim_analysis import *
```

where `*` is the specific module you are interested in.

1.2 Package and repo structure

Please make note of the following directory structures: `conf` and `data`. These are the primary two directories for experimental country data and configuration files. The `conf` directory contains the configuration (`.yml`) files for the simulation, while the `data` directory contains the data files used in the simulation (typically raster `.asc` files).

Each of these folders is organized by country. For example, if you are working with Mozambique, which we abbreviate as `moz`, the directory structure would look like this:

```
data/
    moz/
        calibration/
        ...
conf/
    moz/
        calibration/
        ...
```

Additionally the templates folder contains the template files for the configuration files. These are used to generate the .yml files for the simulation. This templating system is gradually being phased out in favor of a more structured approach using Python data classes in the `configure` module. Ultimately, this package communicates with the MaSim simulation through a `.yml` configuration file that can still be created manually if you desire.

1.3 How to use this notebook

This notebook should be thought of as a structured interactive prompt that guides you through the process of calibrating and validating a country. The code sections are generally organized into sections that can be run independently. The notebook breaks up individual workflows by using markdown headings and note blocks. Due to the some what long-running nature of the tasks that calibration and validation involve, you'll sometimes have to wait and shutdown the kernel the notebook is running on and pick up where you left off another time.

To that end, this notebook is designed to make things organized but also segmented. Keep calibration constants (name, population, etc.) in a block below this and make sure to run that block every time you start the notebook. Module and library imports should be handled in the workflow segment you are currently working on. It is recommend (for speed of execution) to separate out imports from code execution to save time on re-importing.

Workflows are generally separated by a horizontal rule:

1.4 Calibration efforts

The primary calibration point is to relate the beta parameter (rate of infection/biting) with the actual reported prevelence, given the population size of a given map pixel and treatment access rate. This involves a few steps. As a preliminary step, obtain the relevant raster files that contain population data, district mapping values, treatment access, and prevalence (pfpr2-10, or a similar name) and place it under `data/<country>`. Fictitious calibration data will be stored under `data/<country>/calibration`. Create a new branch in the git repository for the calibration process. This is important to keep track of changes and to avoid conflicts with the main branch. The main branch should be reserved as a branching off point for strategy and treatment analysis or new calibration efforts. The workflow there should be to branch off from the main branch, implement the strategies and treatments, and then merge back in any useful changes. Strategy and treament analysis shouldn't be a main contribution to the main branch. The calibration branch name should be descriptive and include the country name, e.g., `calibration-moz`.

Calibration then occurs in two phases and should be done on a separate git branch. The first phase is generating the simulated data for beta calibration. This creates the fictitious configuration

and data files. This concludes with several command and job files to be run on a cluster. At the moment this is configured to work on Temple University's OwlsNest cluster, but the resulting `*_cmds.txt` files simply contain a list of shell commands that execute the simulation and should be generalizable to whatever parallel computing cluster system you are using.

The second phase is started when the batch processing is completed and downloaded locally to the `output/<country>/calibration` directory. These files are then summarized and the prevalence and beta values are fit using a log-sigmoid curve fit when broken down by pixel population and treatment access rate. These fits are then used to generate the beta map for eventual use in the experimental simulation.

1.5 Basic parameters

There are a few country specific parameters that we need to develop and organize. These parameters are frequently used and are listed below:

- `name`: The country code name. Usually two or three letters. Examples: `moz` for Mozambique, `bf` for Burkino Faso, `rwa` for Rwanda, `tz` for Tanzania, etc.
- `birth_rate`: The birth rate of the country. This is used to calculate the population growth rate. This is usually a constant value for the country. Data is typically given in births per 1000 people. Normalize that data to a decimal value.
- `target_population`: The population of the country in the calibration target year.
- `initial_age_structure`: The initial age bins of the country.
- `age_distribution`: The age distribution of the country as percentages corresponding to the age bins.
- `death_rate`: The death rate of the country corresponding to the age bins.
- `access_rates`: the treatment access rates for the country. This is determined by the unique values in the raster file typically called `<name>_treatementseeking.asc`

Some of these are configuration parameters that are fed into the `MaSim` simulation. Others are simply descriptive (e.g. `name`) are are used to organize files. It is useful to have these parameters as variables. Once we have gone through the initial tuning, these parameters are stable and it is useful to simply keep them in a Python source file and import them as neccessary.

1.6 Directory Setup

First: what is the long-form name of the country your are working with?

[1]:
```
long_name = "Angola"
```

Second: what is the short-form country code?

[2]:
```
name = "ago"
```

Now we'll do a basic file system configuration for our project.

[3]:
```
from masim_analysis import commands

commands.setup_directories(name)
```

```
All required raster files found in data/ago.
```

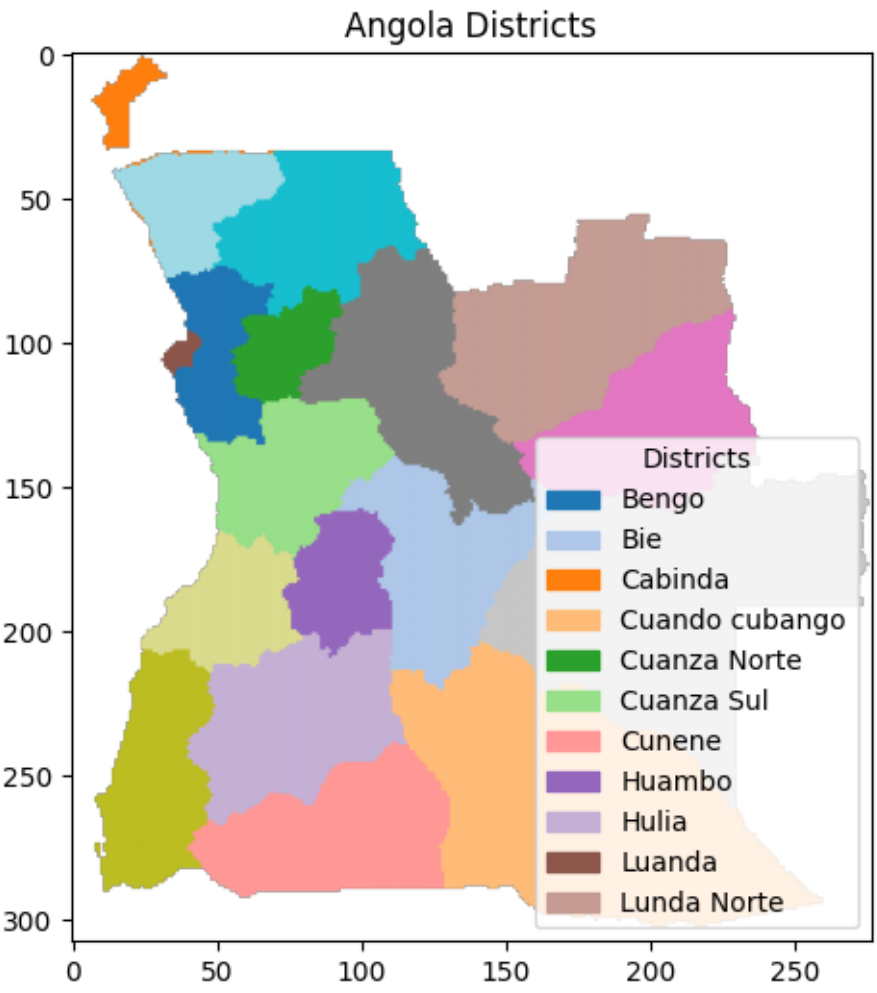
After this initial folder configuration is set up copy the data from the country folder on the drop box into `data/<name>`. This should be a collection of raster (`.asc`) and `.csv` files. At a minimum these should include: - `<name>_districts.asc` (mapping of individual raster pixels to larger scale provinces or health districts) - `<name>_population.asc` (calibration year population) - `<name>_initialpopulation.asc` (backed out initial population) - `<name>_pfpr210.asc` (pfpr210 raster) - `<name>_traveltime.asc` (travel time raster) - `<name>_treatmentseeking.asc` (treatment seeking raster)

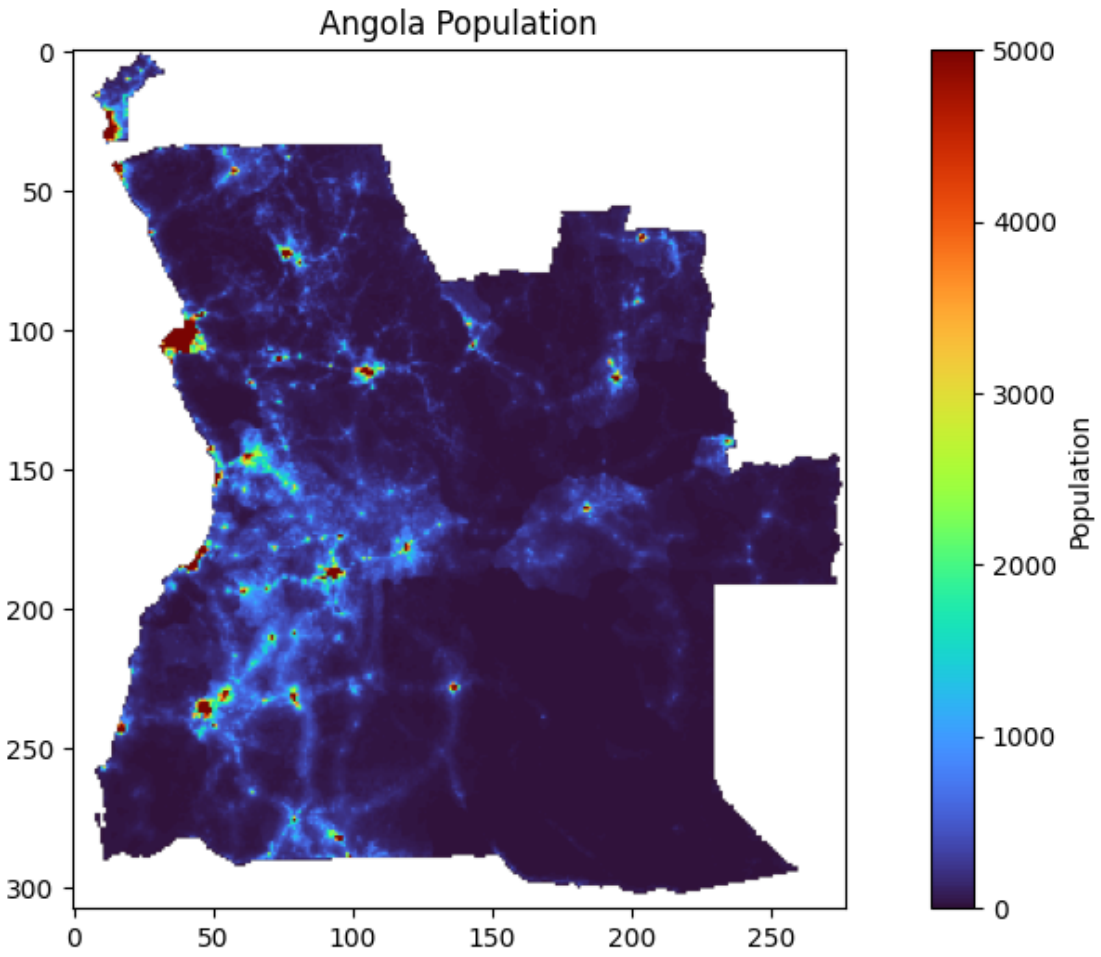
1.6.1 Raster data exploration

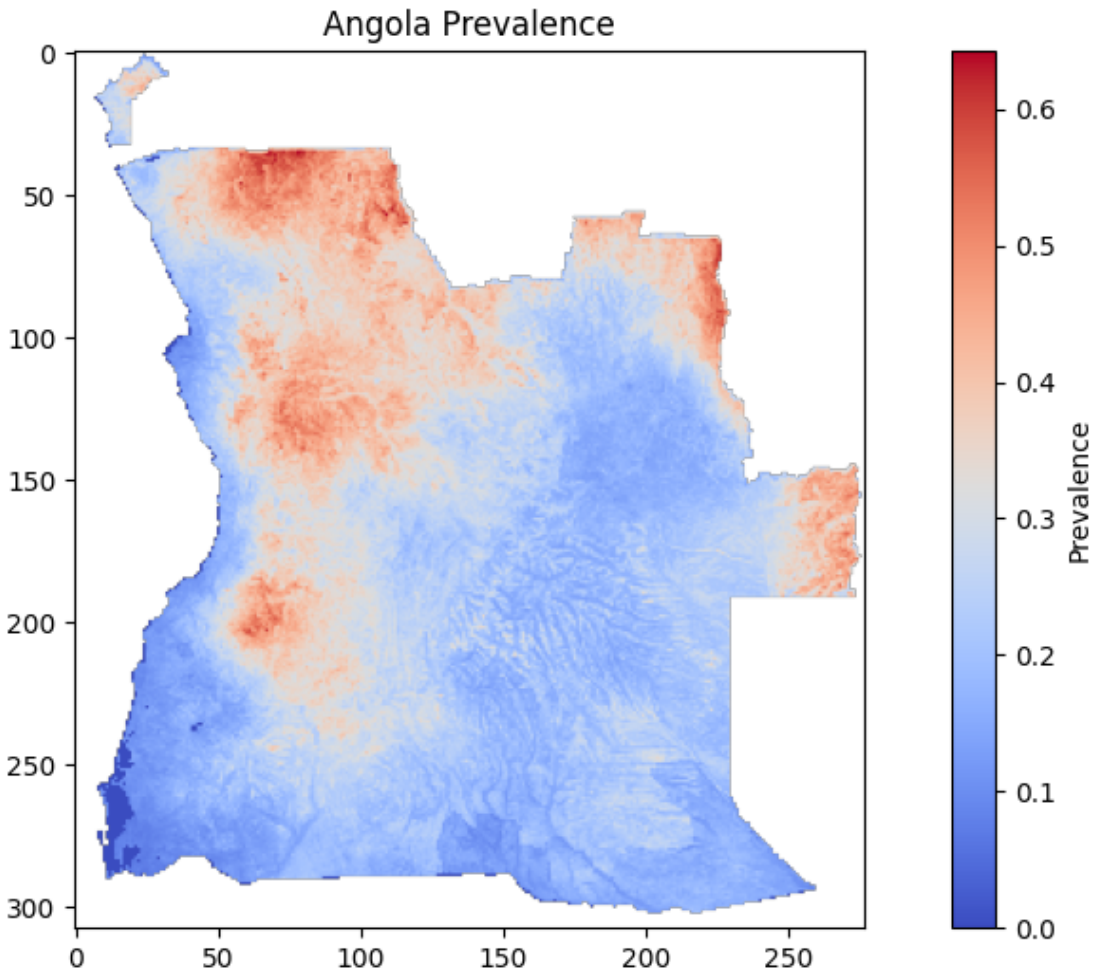
The first calibration step is to read in the relevant rasters to get a sense of some of the big picture data. The rasters are stored in the `data/<country>` directory. This is then used to get some basic statistics to run calibration. Of primary concern is getting the value of the initial population, calibration year population, ‘

[4]:
```
import os
import pandas as pd
from masim_analysis import utils

districts, _ = utils.read_raster(os.path.join("data", name, f"{name}_districts.
↪asc"))
population, _ = utils.read_raster(os.path.join("data", name,␣
↪f"{name}_initialpopulation.asc"))
prevalence, _ = utils.read_raster(os.path.join("data", name, f"{name}_pfpr210.
↪asc"))
district_names = pd.read_csv(os.path.join("data", name, f"{name}_mapping.csv"),␣
↪index_col="ID")
names = district_names.to_dict()[district_names.columns[0]]
```

[5]:
```
dist_fig = utils.plot_districts(districts, names, long_name, fig_size=(12, 6),␣
↪loc="lower right")
pop_fig = utils.plot_population(population, long_name, fig_size=(12, 6),␣
↪population_upper_limit=5000)
pfpr_plot = utils.plot_prevalence(prevalence, long_name, fig_size=(12, 6))
```


Angola Districts
0
50
100
150
200
250
300
0
50
100
150
200
250
Districts
Bengo
Bie
Cabinda
Cuando cubango
Cuanza Norte
Cuanza Sul
Cunene
Huambo
Hulia
Luanda
Lunda Norte


Angola Population
0
50
100
150
200
250
300
0
50
100
150
200
250
5000
4000
3000
2000
1000
0
Population



1.6.2 Drug distribution rates

Each country has a different drug distribution rate. This will factor into the calibration file. Read in the raw file and process it to remove any unnecessary columns or rows.

[6]:
```
drug_distribution = pd.read_csv(
    os.path.join("data", name, f"{name}_drugdistribution.csv"), index_col=0,␣
 ↪na_values="-99"
)
drug_distribution
```

[6]:

date	al	qn	sp	cq	aq	as	other	act	aspirin
2006	NaN	2.8	0.3	14.0	10.7	NaN	0.8	1.6	NaN
2007	NaN	2.8	0.3	14.0	10.7	NaN	0.8	1.6	NaN
2011	NaN	2.7	0.9	2.5	NaN	NaN	2.2	21.7	NaN

```
2015 NaN  9.3  8.3   2.0   7.0 NaN    1.6  76.7      NaN
2016 NaN  9.3  8.3   2.0   7.0 NaN    1.6  76.7      NaN
2023 NaN  NaN  NaN   NaN   NaN NaN    NaN   NaN      NaN
```

[7]:
```
# Drop any columns that are all NaN
drug_distribution.dropna(axis=1, how="all", inplace=True)
# Drop any rows that are all NaN
drug_distribution.dropna(axis=0, how="all", inplace=True)
# Fill any remaining NaN values with 0
drug_distribution.fillna(0, inplace=True)
# convert all columns to lowercase
drug_distribution.columns = [col.lower() for col in drug_distribution.columns]
# Convert to percentages [0, 1]
drug_distribution = drug_distribution.div(100)
drug_distribution
```

[7]:

date	qn	sp	cq	aq	other	act
2006	0.028	0.003	0.140	0.107	0.008	0.016
2007	0.028	0.003	0.140	0.107	0.008	0.016
2011	0.027	0.009	0.025	0.000	0.022	0.217
2015	0.093	0.083	0.020	0.070	0.016	0.767
2016	0.093	0.083	0.020	0.070	0.016	0.767

If in the case that the sum of the distribution rates does not equal 1.0, then you will need to normalize the values so that they do for the purposes of the simulation.

We then relate this to the `DRUG_DB` and `THERAPY_DB` dictionary in `masim_analysis.configure`. Note that the drug distribution may be either a single drug or a combination of drugs. For instance, `al` is a combination of artemether and lumefantrine. This would correspond to a *therapy* entry in `THERAPY_DB` that uses both the *artemether* and *lumefantrine* drugs from `DRUG_DB`.

[8]:
```
from masim_analysis.configure import DRUG_DB, THERAPY_DB

print("DRUGS:")
for idx in DRUG_DB.keys():
    print(f"Drug {idx}: {DRUG_DB[idx]['name']}")
print("============================")
print("THERAPIES:")
for idx in THERAPY_DB.keys():
    drug_ids = THERAPY_DB[idx]["drug_id"]
    print(f"Therapy {idx}: {[DRUG_DB[drug_id]['name'] for drug_id in␣
 ↪drug_ids]}")
```

```
DRUGS:
Drug 0: ART
Drug 1: AQ
Drug 2: SP
Drug 3: CQ
```

```
Drug 4: LUM
Drug 5: PQ
Drug 6: MF
Drug 7: QUIN
============================
THERAPIES:
Therapy 0: ['ART', 'AQ']
Therapy 1: ['ART', 'LUM']
Therapy 2: ['ART', 'MF']
Therapy 3: ['ART', 'PQ']
Therapy 4: ['AQ']
Therapy 5: ['ART']
Therapy 6: ['CQ']
Therapy 7: ['QUIN']
Therapy 8: ['SP']
```

We now create a mapping between the drug distribution table columns to the therapy ID numbers:

[9]:
```
column_mapping: list[int] = [7, 8, 6, 4, 0, 0]
```

Now remove any un-needed columns from the drug distribution table (i.e. 'sum', 'aspirin', 'other')

[10]:
```
# drug_distribution.drop(columns=["other"], inplace=True)
```

1.6.3 Public/Private Distribution

For countries that have a public and private sector split, the overall strategy will be constructed using a MFT. For the overall distribution percentage, we use a weighted value based on the proportion of the split between public and private sector distributions. For example for a country with a 70/30 public/private split a therapy that has a 25% distribution rate in the public sector and a 50% distribution in the private sector would have an overall strategy distribution of 0.7 * 0.25 + 0.3 * 0.50 = 0.325. The private market, however, can be difficult to ascertain in some countries, particularly in the distribution of specific therapies.

If there is no private market or no information on the private market, then we simply use the public sector distribution rates.

[11]:
```
import numpy as np

market_distribution = pd.DataFrame(
    {
        "private":  0.895 * np.ones_like(drug_distribution.index),
        "public": 0.105 * np.ones_like(drug_distribution.index)
    },
    index= drug_distribution.index
)
```

We now apply the public/private market distribution to the drug distribution rates to get the overall distribution rates:

[12]:
```
private_market = drug_distribution.mul(market_distribution["private"], axis=0)
public_market = drug_distribution.mul(market_distribution["public"], axis=0)
combined_distribution = private_market.add(public_market, fill_value=0)
combined_distribution
```

[12]:

date	qn	sp	cq	aq	other	act
2006	0.028	0.003	0.140	0.107	0.008	0.016
2007	0.028	0.003	0.140	0.107	0.008	0.016
2011	0.027	0.009	0.025	0.000	0.022	0.217
2015	0.093	0.083	0.020	0.070	0.016	0.767
2016	0.093	0.083	0.020	0.070	0.016	0.767

1.6.4 Start and End Dates

Next we need to establish the calibration year, starting date, and ending date of the simulation. The calibration year is the year of the most recent data available, typically 2-3 years old.

[13]:
```
calibration_year = 2023
```

Typical procedure is to back out ten years from the calibration year to establish the starting date for the simulation to permit sufficient "burn in" time and then to run the simulation for an additional year after the calibration year.

[14]:
```
from datetime import date

starting_date = date(calibration_year - 11, 1, 1)
ending_date = date(calibration_year + 1, 12, 31)
start_of_comparison_period = date(calibration_year, 1, 1)
```

1.6.5 Baseline Strategy and Event Databases

From this information we must manually construct the baseline strategy(ies) and event(s) databases. Similar to the drug and therapy "databases" we store the strategies in a dictionary with integer keys. Each key corresponds to a strategy which itself is a dictionary with the following keys: - `name`: The name of the strategy. This is used to identify the strategy in the simulation. - `type`: The type of strategy. Leave this as "MFT" until further notice. - `therapy_ids`: The therapy ids used in the strategy. This is a list of integers corresponding to the therapy ids in the `THERAPY_DB` dictionary. - `distribution`: The distribution of the strategy. This is a list of floats corresponding to the distribution of the strategy that we derived from the drug distribution rates from DHS data.

Next create a list that provides the mapping between the column names in the combined distribution table to the numeric index of the therapy in the `THERAPY_DB` dictionary.

[15]:
```
from ruamel.yaml import YAML

yaml = YAML()

os.makedirs(os.path.join("conf", name, "test"), exist_ok=True)
```

```
strategy_db = {}
events = []

i = 0
for _, row in combined_distribution.iterrows():
    strategy_db[i] = {
        "name": f"{row.name}_pub",
        "type": "MFT",
        "therapy_ids": column_mapping,
        "distribution": row.values.tolist(),
    }
    events.append({
        "name": f"{row.name}_strategy",
        "info": [{"day": date(int(row.name), 1, 1).strftime("%Y/%m/%d"),␣
 ↪"strategy_id": i}],
    })
    i += 1

yaml.dump(events, open(os.path.join("conf", name, "test", "events.yaml"), "w"))
yaml.dump(strategy_db, open(os.path.join("conf", name, "test", "strategy_db.
 ↪yaml"), "w"))
```

1.7 Birth Rate Check

This is all then put together to develop an initial configuration to test these baseline parameters for the growth rate. From the team, data files, and some additional research you need to identify the following parameters:

- `birth_rate`: the country's "crude" birth rate (we will be verifying this number in the step)
- `initial_age_structure`: the distribution of the population across different age groups at the start of the simulation
- `age_distribution`: the corresponding age brackets of the initial age distribution
- `death_rate`: the death rate of the country by age bracket.
- `target_population`: the calibration year's population value of the country

The important calibration point is the final year population. We back out the population data from this using a birth rate. To check that the initialization raster is correct, we need to run an initial simulation run to verify this growth rate considering malaria deaths as well. This is really just a sanity check to make sure the input data is correct.

Run an initial check to make sure the growth rate is working correctly. Given the input file for whatever year, run a single simulation and check that the population for the target year is correct. Make sure to scale the ending population by the simulation scale factor as well.

[16]:
```
age_distribution = [ 0.037, 0.132, 0.161, 0.142, 0.090, 0.086, 0.070, 0.052, 0.
 ↪044, 0.044, 0.031, 0.041, 0.024, 0.017, 0.013, 0.017]
birth_rate = 80.0 / 1000
```

```
death_rate = [0.048140, 0.041605, 0.052070, 0.048057, 0.048057, 0.00497, 0.
↪00497, 0.00497, 0.00497, 0.003540, 0.00354, 0.00758, 0.01113, 0.01113, 0.
↪01113]
initial_age_structure = [1, 5, 9, 14, 19, 24, 29, 34, 39, 44, 49, 54, 59, 64,␣
↪69, 100]
target_population = 36_749_906
```

[17]:
```
assert len(age_distribution) == len(initial_age_structure), "Please check to␣
↪make sure that the values and lengths of age_distribution and␣
↪initial_age_structure are consistent."
```

[18]:
```
import numpy as np

initial_population = np.nansum(utils.read_raster(os.path.join("data", name,␣
↪f"{name}_initialpopulation.asc"))[0])
```

Next we'll create a simple simulation run to verify the population growth rate.

[19]:
```
from masim_analysis import calibrate, configure

pop = 100_000
params = configure.configure(
    country_code=name,
    birth_rate=birth_rate,
    initial_age_structure=initial_age_structure,
    age_distribution=age_distribution,
    death_rates=death_rate,
    starting_date=starting_date,
    start_of_comparison_period=start_of_comparison_period,
    ending_date=ending_date,
    strategy_db=strategy_db,
    calibration_str=f"growth_validation_{pop}",
    beta_override=0.00,
    population_scalar=1.0,
    calibration=True,
)
params["events"].extend(events)

yaml.dump(params, open(os.path.join("conf", name, "test",␣
↪f"{name}_growth_validation_{pop}.yaml"), "w"))

calibrate.write_pixel_data_files(params["raster_db"], pop)
```

Now we'll run the simulation to verify the population growth rate.

[20]:
```
filename = os.path.join("conf", name, "test", f"{name}_growth_validation_{pop}.
↪yaml")
os.makedirs(os.path.join("output", name, "test"), exist_ok=True)
```

```
output_file = os.path.join("output", name, "test",␣
  ↪f"{name}_growth_validation_{pop}")
os.makedirs(os.path.dirname(output_file), exist_ok=True)
```

[21]:
```
os.system(f"./bin/MaSim -i ./{filename} -o ./{output_file} -r␣
  ↪SQLiteDistrictReporter")
```

```
[INFO] MaSim version 4.1.8
[INFO] Model initializing…
[INFO] Read input file: ./conf/ago/test/ago_growth_validation_100000.yaml
[INFO] Districts loaded with 1 districs
[INFO] District_lookup loaded with 1 pixels
[INFO] location_db appears to have been set by raster_db
[INFO] Using rainfall-based seasonal information.
[INFO] Relative probability that child travels compared to adult is not set in
input file, defaulting to 1.0
[INFO] Relative probability for a clinical case to travel is not set in input
file, defaulting to 1.0
[WARNING] Unusually high birth rate of 0.08 (80/1000 individuals)
[INFO] Random initializing with seed: 1789276532
[INFO] Starting day is 2012-01-01
[INFO] Location count: 1
[INFO] Population size: 100000
[INFO] Model starting…
[INFO] Perform before run events
[INFO] Simulation is running
[INFO] 16:16:51 - Day: 0
[INFO] 2012-01-01 : turn mutation off
[INFO] 16:16:52 - Day: 30
[INFO] 16:16:53 - Day: 60
[INFO] 16:16:53 - Day: 90
[INFO] 16:16:54 - Day: 120
[INFO] 16:16:55 - Day: 150
[INFO] 16:16:56 - Day: 180
[INFO] 16:16:56 - Day: 210
[INFO] 16:16:57 - Day: 240
[INFO] 16:16:58 - Day: 270
[INFO] 16:16:58 - Day: 300
[INFO] 16:16:59 - Day: 330
[INFO] 16:16:59 - Day: 360
[INFO] 16:17:00 - Day: 390
[INFO] 16:17:00 - Day: 420
[INFO] 16:17:01 - Day: 450
[INFO] 16:17:02 - Day: 480
[INFO] 16:17:02 - Day: 510
[INFO] 16:17:03 - Day: 540
[INFO] 16:17:03 - Day: 570
[INFO] 16:17:04 - Day: 600
```

```
[INFO] 16:17:04 - Day: 630
[INFO] 16:17:05 - Day: 660
[INFO] 16:17:06 - Day: 690
[INFO] 16:17:06 - Day: 720
[INFO] 16:17:07 - Day: 750
[INFO] 16:17:07 - Day: 780
[INFO] 16:17:08 - Day: 810
[INFO] 16:17:08 - Day: 840
[INFO] 16:17:09 - Day: 870
[INFO] 16:17:10 - Day: 900
[INFO] 16:17:10 - Day: 930
[INFO] 16:17:11 - Day: 960
[INFO] 16:17:11 - Day: 990
[INFO] 16:17:12 - Day: 1020
[INFO] 16:17:13 - Day: 1050
[INFO] 16:17:13 - Day: 1080
[INFO] 16:17:14 - Day: 1110
[INFO] 16:17:15 - Day: 1140
[INFO] 16:17:15 - Day: 1170
[INFO] 16:17:16 - Day: 1200
[INFO] 16:17:17 - Day: 1230
[INFO] 16:17:17 - Day: 1260
[INFO] 16:17:18 - Day: 1290
[INFO] 16:17:18 - Day: 1320
[INFO] 16:17:19 - Day: 1350
[INFO] 16:17:20 - Day: 1380
[INFO] 16:17:20 - Day: 1410
[INFO] 16:17:21 - Day: 1440
[INFO] 16:17:22 - Day: 1470
[INFO] 16:17:23 - Day: 1500
[INFO] 16:17:23 - Day: 1530
[INFO] 16:17:24 - Day: 1560
[INFO] 16:17:25 - Day: 1590
[INFO] 16:17:25 - Day: 1620
[INFO] 16:17:26 - Day: 1650
[INFO] 16:17:27 - Day: 1680
[INFO] 16:17:27 - Day: 1710
[INFO] 16:17:28 - Day: 1740
[INFO] 16:17:29 - Day: 1770
[INFO] 16:17:29 - Day: 1800
[INFO] No genotypes recorded in the simulation at timestep, 1827
[INFO] 16:17:31 - Day: 1830
[INFO] No genotypes recorded in the simulation at timestep, 1858
[INFO] 16:17:31 - Day: 1860
[INFO] No genotypes recorded in the simulation at timestep, 1886
[INFO] 16:17:32 - Day: 1890
[INFO] No genotypes recorded in the simulation at timestep, 1917
[INFO] 16:17:33 - Day: 1920
```

```
[INFO] No genotypes recorded in the simulation at timestep, 1947
[INFO] 16:17:34 - Day: 1950
[INFO] No genotypes recorded in the simulation at timestep, 1978
[INFO] 16:17:35 - Day: 1980
[INFO] No genotypes recorded in the simulation at timestep, 2008
[INFO] 16:17:36 - Day: 2010
[INFO] No genotypes recorded in the simulation at timestep, 2039
[INFO] 16:17:37 - Day: 2040
[INFO] 16:17:38 - Day: 2070
[INFO] No genotypes recorded in the simulation at timestep, 2070
[INFO] 16:17:39 - Day: 2100
[INFO] No genotypes recorded in the simulation at timestep, 2100
[INFO] 16:17:40 - Day: 2130
[INFO] No genotypes recorded in the simulation at timestep, 2131
[INFO] 16:17:41 - Day: 2160
[INFO] No genotypes recorded in the simulation at timestep, 2161
[INFO] 16:17:42 - Day: 2190
[INFO] No genotypes recorded in the simulation at timestep, 2192
[INFO] 16:17:43 - Day: 2220
[INFO] No genotypes recorded in the simulation at timestep, 2223
[INFO] 16:17:44 - Day: 2250
[INFO] No genotypes recorded in the simulation at timestep, 2251
[INFO] 16:17:45 - Day: 2280
[INFO] No genotypes recorded in the simulation at timestep, 2282
[INFO] 16:17:46 - Day: 2310
[INFO] No genotypes recorded in the simulation at timestep, 2312
[INFO] 16:17:46 - Day: 2340
[INFO] No genotypes recorded in the simulation at timestep, 2343
[INFO] 16:17:47 - Day: 2370
[INFO] No genotypes recorded in the simulation at timestep, 2373
[INFO] 16:17:48 - Day: 2400
[INFO] No genotypes recorded in the simulation at timestep, 2404
[INFO] 16:17:49 - Day: 2430
[INFO] No genotypes recorded in the simulation at timestep, 2435
[INFO] 16:17:50 - Day: 2460
[INFO] No genotypes recorded in the simulation at timestep, 2465
[INFO] 16:17:51 - Day: 2490
[INFO] No genotypes recorded in the simulation at timestep, 2496
[INFO] 16:17:52 - Day: 2520
[INFO] No genotypes recorded in the simulation at timestep, 2526
[INFO] 16:17:53 - Day: 2550
[INFO] No genotypes recorded in the simulation at timestep, 2557
[INFO] 16:17:54 - Day: 2580
[INFO] No genotypes recorded in the simulation at timestep, 2588
[INFO] 16:17:55 - Day: 2610
[INFO] No genotypes recorded in the simulation at timestep, 2616
[INFO] 16:17:56 - Day: 2640
[INFO] No genotypes recorded in the simulation at timestep, 2647
```

```
[INFO] 16:17:57 - Day: 2670
[INFO] No genotypes recorded in the simulation at timestep, 2677
[INFO] 16:17:58 - Day: 2700
[INFO] No genotypes recorded in the simulation at timestep, 2708
[INFO] 16:17:59 - Day: 2730
[INFO] No genotypes recorded in the simulation at timestep, 2738
[INFO] 16:18:00 - Day: 2760
[INFO] No genotypes recorded in the simulation at timestep, 2769
[INFO] 16:18:00 - Day: 2790
[INFO] No genotypes recorded in the simulation at timestep, 2800
[INFO] 16:18:02 - Day: 2820
[INFO] No genotypes recorded in the simulation at timestep, 2830
[INFO] 16:18:03 - Day: 2850
[INFO] No genotypes recorded in the simulation at timestep, 2861
[INFO] 16:18:04 - Day: 2880
[INFO] No genotypes recorded in the simulation at timestep, 2891
[INFO] 16:18:05 - Day: 2910
[INFO] No genotypes recorded in the simulation at timestep, 2922
[INFO] 16:18:06 - Day: 2940
[INFO] No genotypes recorded in the simulation at timestep, 2953
[INFO] 16:18:07 - Day: 2970
[INFO] No genotypes recorded in the simulation at timestep, 2982
[INFO] 16:18:07 - Day: 3000
[INFO] No genotypes recorded in the simulation at timestep, 3013
[INFO] 16:18:08 - Day: 3030
[INFO] No genotypes recorded in the simulation at timestep, 3043
[INFO] 16:18:09 - Day: 3060
[INFO] No genotypes recorded in the simulation at timestep, 3074
[INFO] 16:18:10 - Day: 3090
[INFO] No genotypes recorded in the simulation at timestep, 3104
[INFO] 16:18:11 - Day: 3120
[INFO] No genotypes recorded in the simulation at timestep, 3135
[INFO] 16:18:12 - Day: 3150
[INFO] No genotypes recorded in the simulation at timestep, 3166
[INFO] 16:18:13 - Day: 3180
[INFO] No genotypes recorded in the simulation at timestep, 3196
[INFO] 16:18:14 - Day: 3210
[INFO] No genotypes recorded in the simulation at timestep, 3227
[INFO] 16:18:15 - Day: 3240
[INFO] No genotypes recorded in the simulation at timestep, 3257
[INFO] 16:18:16 - Day: 3270
[INFO] No genotypes recorded in the simulation at timestep, 3288
[INFO] 16:18:17 - Day: 3300
[INFO] No genotypes recorded in the simulation at timestep, 3319
[INFO] 16:18:18 - Day: 3330
[INFO] No genotypes recorded in the simulation at timestep, 3347
[INFO] 16:18:19 - Day: 3360
[INFO] No genotypes recorded in the simulation at timestep, 3378
```

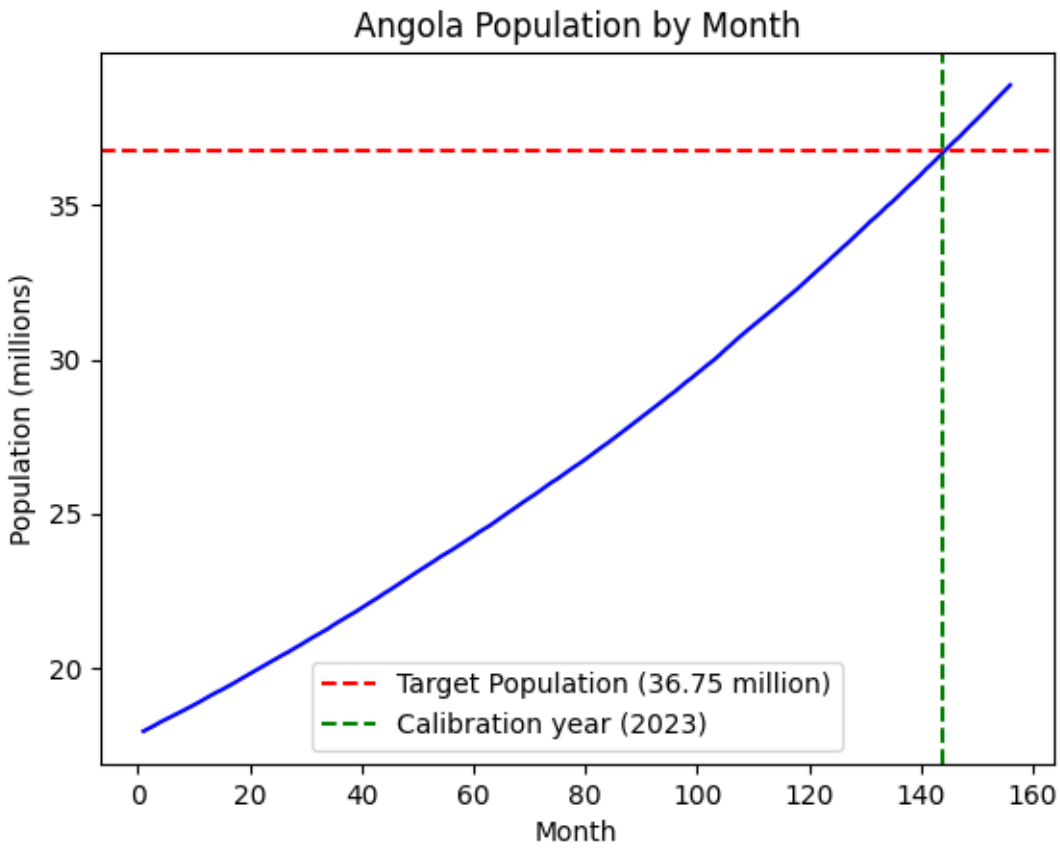
```
[INFO] 16:18:20 - Day: 3390
[INFO] No genotypes recorded in the simulation at timestep, 3408
[INFO] 16:18:22 - Day: 3420
[INFO] No genotypes recorded in the simulation at timestep, 3439
[INFO] 16:18:23 - Day: 3450
[INFO] No genotypes recorded in the simulation at timestep, 3469
[INFO] 16:18:24 - Day: 3480
[INFO] No genotypes recorded in the simulation at timestep, 3500
[INFO] 16:18:25 - Day: 3510
[INFO] No genotypes recorded in the simulation at timestep, 3531
[INFO] 16:18:26 - Day: 3540
[INFO] No genotypes recorded in the simulation at timestep, 3561
[INFO] 16:18:28 - Day: 3570
[INFO] No genotypes recorded in the simulation at timestep, 3592
[INFO] 16:18:29 - Day: 3600
[INFO] No genotypes recorded in the simulation at timestep, 3622
[INFO] 16:18:30 - Day: 3630
[INFO] No genotypes recorded in the simulation at timestep, 3653
[INFO] 16:18:31 - Day: 3660
[INFO] No genotypes recorded in the simulation at timestep, 3684
[INFO] 16:18:32 - Day: 3690
[INFO] No genotypes recorded in the simulation at timestep, 3712
[INFO] 16:18:33 - Day: 3720
[INFO] No genotypes recorded in the simulation at timestep, 3743
[INFO] 16:18:35 - Day: 3750
[INFO] No genotypes recorded in the simulation at timestep, 3773
[INFO] 16:18:36 - Day: 3780
[INFO] No genotypes recorded in the simulation at timestep, 3804
[INFO] 16:18:37 - Day: 3810
[INFO] No genotypes recorded in the simulation at timestep, 3834
[INFO] 16:18:38 - Day: 3840
[INFO] No genotypes recorded in the simulation at timestep, 3865
[INFO] 16:18:40 - Day: 3870
[INFO] No genotypes recorded in the simulation at timestep, 3896
[INFO] 16:18:41 - Day: 3900
[INFO] No genotypes recorded in the simulation at timestep, 3926
[INFO] 16:18:42 - Day: 3930
[INFO] No genotypes recorded in the simulation at timestep, 3957
[INFO] 16:18:43 - Day: 3960
[INFO] No genotypes recorded in the simulation at timestep, 3987
[INFO] 16:18:45 - Day: 3990
[INFO] No genotypes recorded in the simulation at timestep, 4018
[INFO] 16:18:46 - Day: 4020
[INFO] No genotypes recorded in the simulation at timestep, 4049
[INFO] 16:18:47 - Day: 4050
[INFO] No genotypes recorded in the simulation at timestep, 4077
[INFO] 16:18:48 - Day: 4080
[INFO] No genotypes recorded in the simulation at timestep, 4108
```

```
[INFO] 16:18:50 - Day: 4110
[INFO] No genotypes recorded in the simulation at timestep, 4138
[INFO] 16:18:51 - Day: 4140
[INFO] No genotypes recorded in the simulation at timestep, 4169
[INFO] 16:18:52 - Day: 4170
[INFO] No genotypes recorded in the simulation at timestep, 4199
[INFO] 16:18:53 - Day: 4200
[INFO] 16:18:54 - Day: 4230
[INFO] No genotypes recorded in the simulation at timestep, 4230
[INFO] 16:18:55 - Day: 4260
[INFO] No genotypes recorded in the simulation at timestep, 4261
[INFO] 16:18:57 - Day: 4290
[INFO] No genotypes recorded in the simulation at timestep, 4291
[INFO] 16:18:58 - Day: 4320
[INFO] No genotypes recorded in the simulation at timestep, 4322
[INFO] 16:18:59 - Day: 4350
[INFO] No genotypes recorded in the simulation at timestep, 4352
[INFO] 16:19:01 - Day: 4380
[INFO] No genotypes recorded in the simulation at timestep, 4383
[INFO] 16:19:02 - Day: 4410
[INFO] No genotypes recorded in the simulation at timestep, 4414
[INFO] 16:19:03 - Day: 4440
[INFO] No genotypes recorded in the simulation at timestep, 4443
[INFO] 16:19:05 - Day: 4470
[INFO] No genotypes recorded in the simulation at timestep, 4474
[INFO] 16:19:06 - Day: 4500
[INFO] No genotypes recorded in the simulation at timestep, 4504
[INFO] 16:19:07 - Day: 4530
[INFO] No genotypes recorded in the simulation at timestep, 4535
[INFO] 16:19:09 - Day: 4560
[INFO] No genotypes recorded in the simulation at timestep, 4565
[INFO] 16:19:10 - Day: 4590
[INFO] No genotypes recorded in the simulation at timestep, 4596
[INFO] 16:19:11 - Day: 4620
[INFO] No genotypes recorded in the simulation at timestep, 4627
[INFO] 16:19:13 - Day: 4650
[INFO] No genotypes recorded in the simulation at timestep, 4657
[INFO] 16:19:14 - Day: 4680
[INFO] No genotypes recorded in the simulation at timestep, 4688
[INFO] 16:19:16 - Day: 4710
[INFO] No genotypes recorded in the simulation at timestep, 4718
[INFO] 16:19:17 - Day: 4740
[INFO] Perform after run events
[INFO] Model finished!
[INFO] Final population: 218025
[INFO] Elapsed time (s): 146.192293659
[INFO] Memory used: 524596 Kb physical, 536268 Kb virtual
```

[21]: 0

[25]:
```
from matplotlib import pyplot as plt
from masim_analysis import analysis

data = analysis.get_table(f"{output_file}monthly_data_0.db", "monthlysitedata")
starting_pop = data[data["monthlydataid"] == 1]["population"].sum()
last_month = data["monthlydataid"].unique()[-13]
population_by_month = [
    data[data["monthlydataid"] == month]["population"].sum() for month in␣
 ↪data["monthlydataid"].unique()
]
population_by_month = np.array(population_by_month)
ending_population = population_by_month[-13]
growth_rate = (ending_population - starting_pop) / starting_pop
projected_population = initial_population * (1 + growth_rate)
print(f"Starting population: {starting_pop}")
print(f"Ending population: {ending_population}")
print(f"Growth rate: {growth_rate}")
print(f"Projected {calibration_year} population: {projected_population}")
print(f"Percent error: {100 * (projected_population - target_population) /␣
 ↪target_population:.4f}%")

# plots
population_scalar = population_by_month / starting_pop
plt.plot(data["monthlydataid"].unique(), (initial_population *␣
 ↪population_scalar) / 1_000_000, linestyle="-", color="b")
plt.axhline(y=target_population / 1_000_000, color="r", linestyle="--",␣
 ↪label=f"Target Population ({target_population / 1_000_000:.2f} million)")
plt.axvline(x=last_month, color="g", linestyle="--", label=f"Calibration year␣
 ↪({calibration_year})")
plt.xlabel("Month")
plt.ylabel("Population (millions)")
plt.title(f"{long_name} Population by Month")
plt.legend()
plt.savefig(os.path.join("images", name, f"{name}_population_by_month.png"))
plt.show()
```

```
Starting population: 100000
Ending population: 204678
Growth rate: 1.04678
Projected 2023 population: 36706608.660960004
Percent error: -0.1178%
```



You should shoot for an absolute error of less than 2%. Finally we'll add in the target case information.

[23]:
```
target_count = 9_098_006
lower_bound =  8_500_000
upper_bound =  10_500_000
```

With that achieved, we can now save off these initial configuration parameters.

[24]:
```
import json
from masim_analysis.configure import CountryParams

country = CountryParams(
    name,
    long_name,
    age_distribution,
    birth_rate,
    calibration_year,
    death_rate,
    initial_age_structure,
    target_population,
```

```
    starting_date,
    ending_date,
    start_of_comparison_period,
    target_count,
    lower_bound,
    upper_bound,
)
with open(os.path.join("conf", name, "test", f"{name}_country_params.json"),␣
 ↪"w") as f:
    f.write(json.dumps(country.to_dict(), indent=4))
```

Now you can use the below code block to load these parameters.

[25]:
```
import os
from masim_analysis.configure import CountryParams

country = CountryParams.load(name="ago")
```

1.8 Seasonality Calibration

Seasonality is a something of a manual process to fit precisely. The goal is to fit a curve that gives a scalar parameter that modifies the incidence rate for each day of the year. This is written to a `.csv` file historically called "adjustment" or "rainfall" and is a simple row-wise list of scalar values that effect the overall incidence or biting rate. This notebook choose to save this output as `seasonality.csv` in the `data/<country>/calibration` directory.

As we know from previous work, the seasonality of malaria (and most diseases) is roughly sinusoidal. The goal is to fit a sinusoidal curve to the data. With that as well the working assumption of the Boni Lab is to use the positive half of the sine curve. There is not a strong stance on this as seasonal incidence modeling is not a major research concern of the lab. Simply put, taking the positive half of the sine curve is what previous publications from the lab has done and to avoid unneccesary problems in the peer review process, we will continue to do this. This notebook will walk through the process for the full sine curve for completeness' sake.

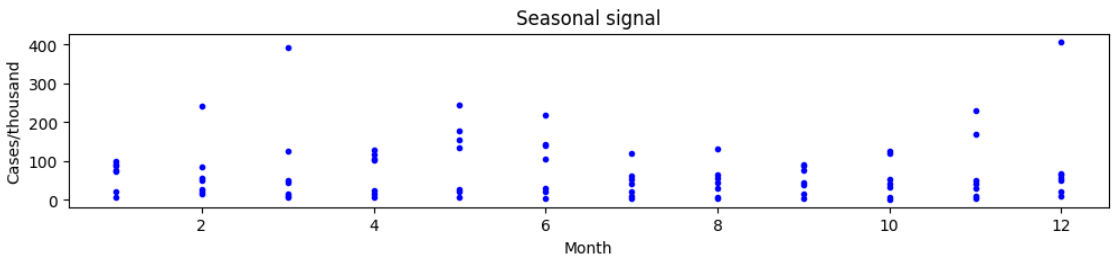
The fitting is done using the `scipy.optimize.curve_fit` function. The function takes in a model function and the data to fit. The model function is a sinusoidal function with a phase shift, period, amplitude, and offset.

$$s(x) = A \sin\left(\frac{2\pi x}{P} + \phi\right) + B$$

We are looking to fit a scalar multiplier of the base incidence data not a curve over the specific incidence. We can do this by normalizing the incidence data about the median, mean, or mode.

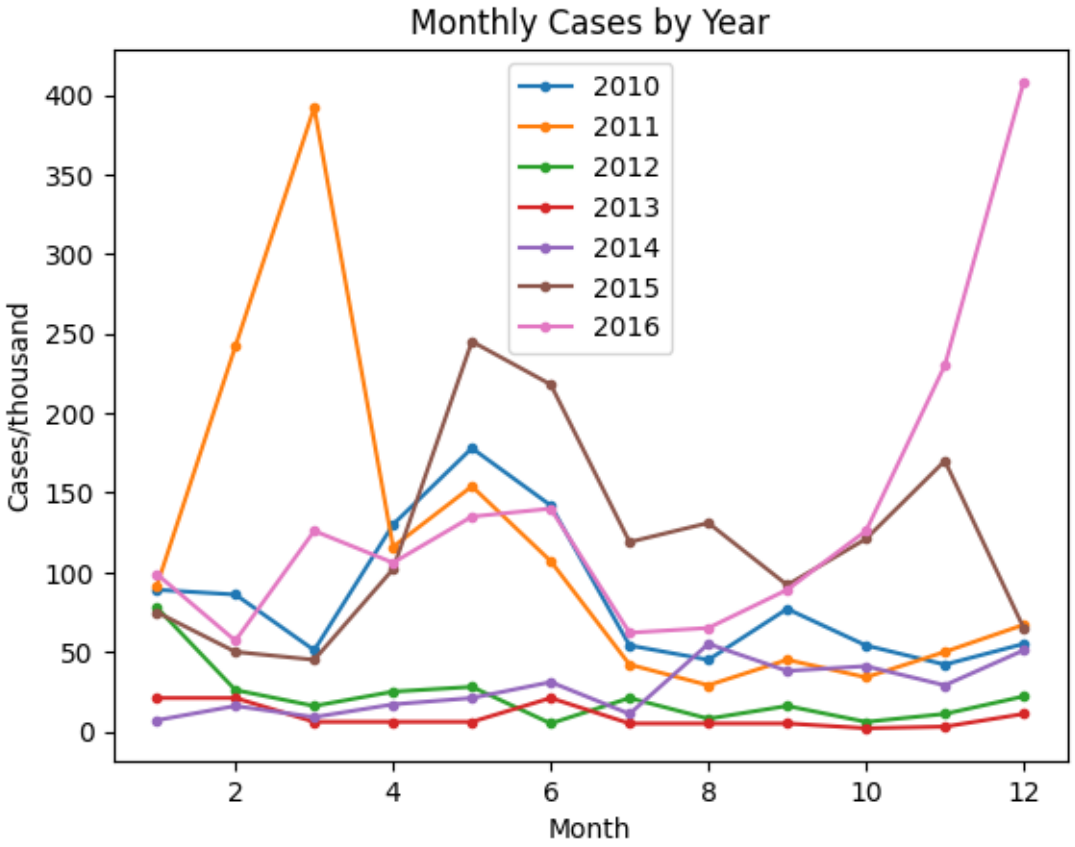
[27]:
```
import os
import pandas as pd
from matplotlib import pyplot as plt
from masim_analysis.configure import CountryParams
```

```
from pathlib import Path

country = CountryParams.load(name="ago")

data = pd.read_csv(Path("data") / country.country_code / f"{country.
 ↪country_code}_incidence_data.csv")
data.dropna(inplace=True)
# data["day"] = 1
# data["date"] = pd.to_datetime(data[["day", "month", "year"]])

x = data["month"].to_numpy()
y = data["cases"].to_numpy()

fig, ax = plt.subplots(figsize=(12, 2))
ax.plot(x, y, ".b", label="Data")
ax.set_xlabel("Month")
ax.set_ylabel("Cases/thousand")
ax.set_title("Seasonal signal")
plt.savefig(os.path.join("images", country.country_code, f"{country.
 ↪country_code}_seasonal_signal.png"))
plt.show()
```



[29]:
```
year_2010 = data.iloc[0:12]
year_2011 = data.iloc[12:24]
year_2012 = data.iloc[24:36]
year_2013 = data.iloc[36:48]
year_2014 = data.iloc[48:60]
year_2015 = data.iloc[60:72]
year_2016 = data.iloc[72:84]

plt.plot(year_2010["month"], year_2010["cases"], ".-", label="2010")
plt.plot(year_2011["month"], year_2011["cases"], ".-", label="2011")
plt.plot(year_2012["month"], year_2012["cases"], ".-", label="2012")
plt.plot(year_2013["month"], year_2013["cases"], ".-", label="2013")
plt.plot(year_2014["month"], year_2014["cases"], ".-", label="2014")
plt.plot(year_2015["month"], year_2015["cases"], ".-", label="2015")
```

```
plt.plot(year_2016["month"], year_2016["cases"], ".-", label="2016")
plt.legend()
plt.xlabel("Month")
plt.ylabel("Cases/thousand")
plt.title("Monthly Cases by Year")
plt.savefig(os.path.join("images", country.country_code, f"{country.
  ↪country_code}_monthly_cases_by_year.png"))
plt.show()
```



Cleaning done for Angola.

[30]:
```
data_clean = data.copy()
data_clean
```

[30]:

	month	year	cases
0	1	2010	89
1	2	2010	86
2	3	2010	51
3	4	2010	130
4	5	2010	178

```
..    …    …    …
79     8  2016    65
80     9  2016    89
81    10  2016   126
82    11  2016   230
83    12  2016   408

[84 rows x 3 columns]
```

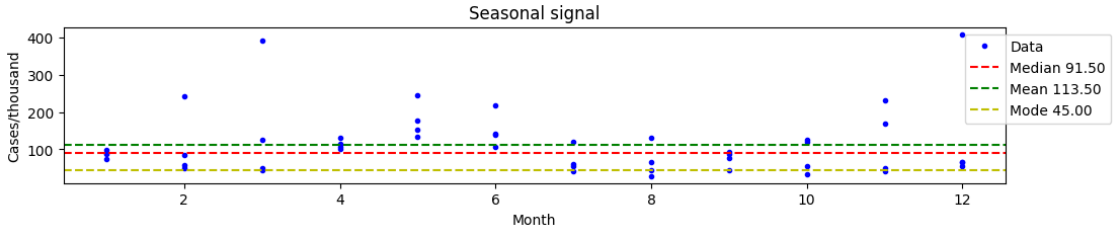
[31]:
```
data_clean.drop(index=data_clean.loc[data_clean["year"] == 2012].index,␣
 ↪inplace=True)
data_clean.drop(index=data_clean.loc[data_clean["year"] == 2013].index,␣
 ↪inplace=True)
data_clean.drop(index=data_clean.loc[data_clean["year"] == 2014].index,␣
 ↪inplace=True)
```

[32]:
```
x = data_clean["month"].to_numpy()
y = data_clean["cases"].to_numpy()
```

End cleaning done for Angola.

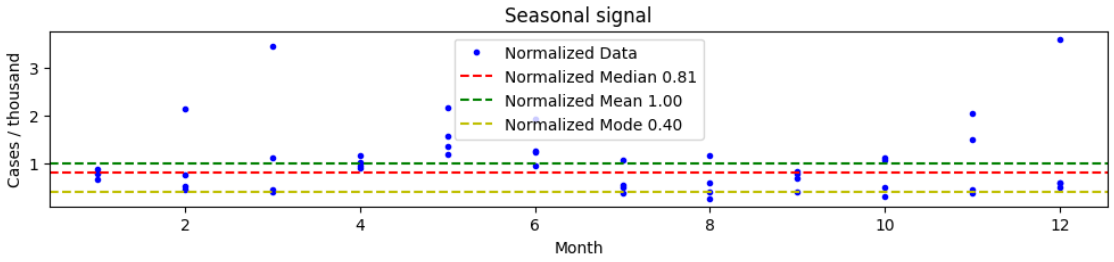
[33]:
```
from scipy.stats import mode as spmode
import numpy as np

median = float(np.median(y))
mean = float(np.mean(y))
mode = float(spmode(y).mode)
```

[34]:
```
fig, ax = plt.subplots(figsize=(12, 2))
ax.plot(x, y, ".b", label="Data")
ax.set_xlabel("Month")
ax.set_ylabel("Cases/thousand")
ax.set_title("Seasonal signal")
ax.axhline(median, color="r", linestyle="--", label=f"Median {median:.2f}")
ax.axhline(mean, color="g", linestyle="--", label=f"Mean {mean:.2f}")
ax.axhline(mode, color="y", linestyle="--", label=f"Mode {mode:.2f}")
ax.legend(loc="upper right", bbox_to_anchor=(1.12, 1))
plt.savefig(os.path.join("images", country.country_code, f"{country.
 ↪country_code}_seasonal_signal_means.png"))
plt.show()
```



Normailze the data with respect to the mean value.

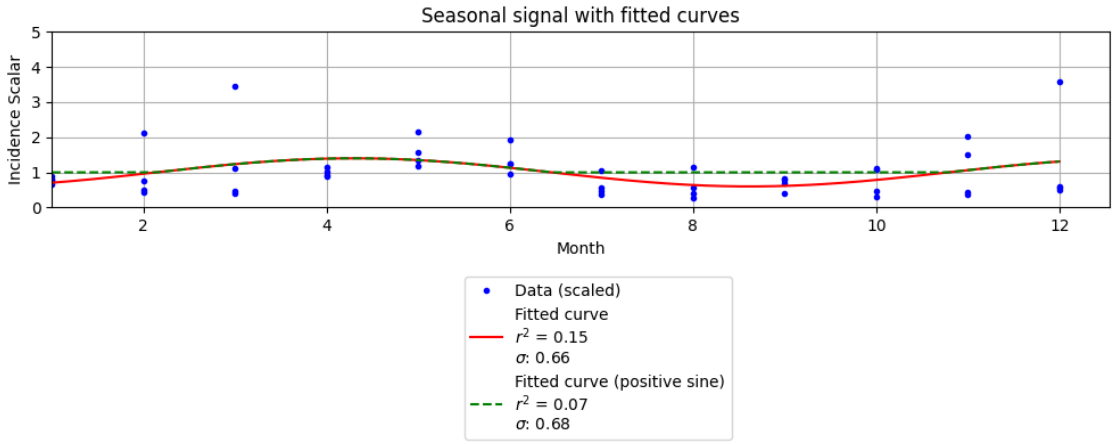
[35]:
```
y_norm = y / mean

fig, ax = plt.subplots(figsize=(12, 2))
ax.plot(x, y_norm, ".b", label="Normalized Data")
ax.set_xlabel("Month")
ax.set_ylabel("Cases / thousand")
ax.set_title("Seasonal signal")
ax.axhline(median / mean, color="r", linestyle="--", label=f"Normalized Median␣
 ↪{median / mean:.2f}")
ax.axhline(mean / mean,   color="g", linestyle="--", label=f"Normalized Mean␣
 ↪{mean / mean:.2f}")
ax.axhline(mode / mean,   color="y", linestyle="--", label=f"Normalized Mode␣
 ↪{mode / mean:.2f}")
ax.legend()
plt.savefig(os.path.join("images", country.country_code, f"{country.
 ↪country_code}_seasonal_signal_normalized.png"))
plt.show()
```



Run the curve fit function and analyze with respect to both the full sinusoid and the positive half of the sine curve. Models of these functions are found in the `calibrate` module.

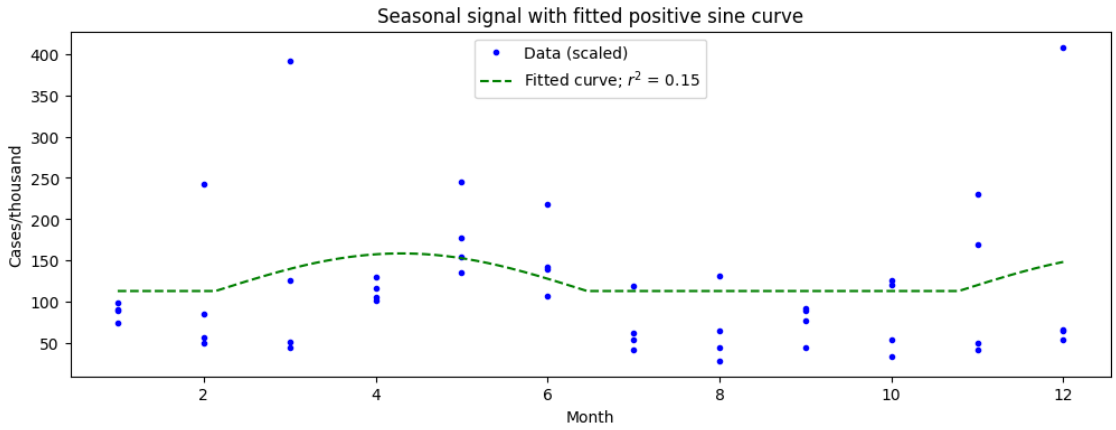
[36]:
```
from scipy.optimize import curve_fit
from masim_analysis.calibrate import sinusoidal, positive_sinusoidal

x_fit = x
p0 =[1, 12, 0, 1]
fit = curve_fit(sinusoidal, x, y_norm, p0)

coefs = fit[0]

t = np.linspace(1, 12, 1000)
```

```
full_sine = sinusoidal(x_fit, *coefs)
r2 = 1 - (np.sum((y_norm - full_sine) ** 2) / np.sum((y_norm - np.mean(y_norm))␣
 ↪** 2))
std_dev = np.std(y_norm - full_sine)

positive_sine = positive_sinusoidal(x_fit, *coefs)
r2_positive = 1 - (np.sum((y_norm - positive_sine) ** 2) / np.sum((y_norm - np.
 ↪mean(y_norm)) ** 2))
std_dev_positive = np.std(y_norm - positive_sine)

fig, ax = plt.subplots(figsize=(12, 2))
ax.plot(x, y_norm, ".b", label="Data (scaled)")
ax.plot(t, sinusoidal(t, *coefs), "r-", label=f"Fitted curve\n$r^2$ = {r2:0.
 ↪2f}\n$\\sigma$: {std_dev:0.2f}")
ax.plot(
    t,
    positive_sinusoidal(t, *coefs),
    "g--",
    label=f"Fitted curve (positive sine)\n$r^2$ = {r2_positive:0.2f}\n$\\sigma$:
 ↪ {std_dev_positive:0.2f}",
)
ax.set_xlabel("Month")
ax.set_ylabel("Incidence Scalar")
ax.set_yticks([0, 1, 2, 3, 4, 5])
#ax.set_xticks(data["date"])
#ax.set_xticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
ax.set_xlim(left=1)
ax.set_ylim(bottom=0)

ax.legend(loc="upper right", bbox_to_anchor=(0.65, -0.35))
ax.grid(True)
plt.title("Seasonal signal with fitted curves")
plt.savefig(os.path.join("images", country.country_code, f"{country.
 ↪country_code}_seasonal_signal_fitted.png"), bbox_inches="tight")
plt.show()
```



The fit is not great, but this is a relatively minor aspect of the simulation. The point is to introduce some degree of seasonal variation that is reasonable. Now let's transform the fit back to the incidence rate.

[40]:
```
fig, ax = plt.subplots(figsize=(12, 4))
ax.plot(x_fit, y, ".b", label="Data (scaled)")
ax.plot(t, positive_sinusoidal(t, *coefs) * mean, "g--", label="Fitted curve;␣
 ↪$r^2$ = %.2f" % r2)
ax.legend()
ax.set_xlabel("Month")
ax.set_ylabel("Cases/thousand")
ax.set_title("Seasonal signal with fitted positive sine curve")
fig.savefig(os.path.join("images", country.country_code, f"{country.
 ↪country_code}_seasonal_signal_fitted_positive.png"))
plt.show()
```



[36]:
```
# Write the fit to a file for 365 days
fit_x = np.arange(1, 366)
fit_y = positive_sinusoidal(fit_x, *fit[0])
data = pd.DataFrame({"day": fit_x, "cases/thousand": fit_y})
data = data.set_index("day")
data.to_csv(os.path.join("data", country.country_code, f"{country.
 ↪country_code}_seasonality.csv"), index=False, header=False)
```

1.9 Treatment Seeking Raster Adjustment

The treatment seeking or treatment access rate raster is a percentage. It may be given as a value between 0 and 1 or between 0 and 100. The MaSim simulation expects this raster to be in the range of 0 to 1. If the raster is in the range of 0 to 100, we need to divide the raster by 100 to get it into the correct range.

[37]:
```
from pathlib import Path
from masim_analysis import utils
from masim_analysis.configure import CountryParams
import numpy as np

country = CountryParams.load("ago")

raster, meta = utils.read_raster(Path("data") / country.country_code /␣
 ↪f"{country.country_code}_treatmentseeking.asc")
if np.nanmax(raster) > 1.0:
    print("Raster appears to be in the range 0-100. Dividing by 100 to convert␣
 ↪to 0-1 range.")
    raster /= 100.0

print(f"Treatment seeking rates: {np.unique(raster)}")

utils.write_raster(raster, Path("data") / country.country_code / f"{country.
 ↪country_code}_treatmentseeking.asc", meta["xllcorner"], meta["yllcorner"])
```

```
Treatment seeking rates: [0.366 0.416 0.446 0.46  0.465 0.468 0.48  0.494 0.502
0.533 0.54  0.569
 0.573 0.585 0.594 0.608 0.638 0.853   nan]
```

1.10 Transfer to Nessun Dorma

At this point all the necessary configuration and data files should be in place to run the calibration batch jobs on the Nessun Dorma cluster. The next step is to transfer the necessary files to the cluster and set up the job scripts to run the calibration jobs.

At the moment the current workflow is to check the updated data files into the git repository and then clone or pull the updates onto the Nessun Dorma cluster.

You can use the commands utility to manually setup and generate the calibration commands and jobs scripts, or alternatively you can use the below job script to automatically set up the directories, run the calibration, and validation in one go. This first requires that you set up and build the virtual environment on Nessun Dorma (use the set up script: `scripts/server_build.sh`). Once that is done you can use the below job script as a template to run the calibration and validation.

```
#!/bin/bash
#PBS -l walltime=240:00:00
#PBS -N <country code>
#PBS -q nd
#PBS -l select=1:ncpus=110:host=<target node>
#PBS -o ./<country name>.output
#PBS -e ./<country name>.error
#PBS -m bae
#PBS -M <your email>

cd $PBS_O_WORKDIR

source .venv/bin/activate

# Check if the calibrate command installed via the python venv is available
if ! command -v calibrate &> /dev/null
then
    echo "calibrate command could not be found. Ensure the virtual environment is set up corre
    exit 1
fi

commands setup <country code>

calibrate <country code> -r 20

echo "Calibration for <country name> completed."
echo "================================="
echo "Running validation now."

validate <country code> -r 50
```

Save that script as `<country code>.pbs` and submit it to the Nessun Dorma cluster using the `qsub` command:

```
qsub <country code>.pbs
```

When finished use `scp` to transfer the output files back to your local machine for analysis (typically in `output/<country code>/calibration`, `output/<country code>/validation`, `data/<country code>/<country code>_beta.asc`, and `images/<country code>/`).
