laser-measles

Release 0.6.1

Author name not set

Jun 09, 2025

CONTENTS

Spatial models of measles implemented with the LASER toolkit.

CHAPTER

ONE

OVERVIEW

docs
tests
package

Spatial models of measles implemented with the LASER toolkit.

- Free software: MIT license

1.1 Installation

```
pip install laser-measles
```

You can also install the in-development version with:

```
pip install https://github.com/InstituteforDiseaseModeling/laser-measles/archive/main.zip
```

1.2 Documentation

https://laser-measles.readthedocs.io/en/latest/

1.3 Development

To run all the tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER

TWO

INSTALLATION

At the command line:

```
pip install laser-measles
```

Or from the source code:

```
git clone https://github.com/InstituteforDiseaseModeling/laser-measles.git
cd laser-measles
pip install -e .
```

CHAPTER

THREE

USAGE

To use the project:

```
import laser_measles
laser_measles.compute(...)
```

CHAPTER

FOUR

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

4.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.2 Documentation improvements

laser_measles could always use more documentation, whether as part of the official laser_measles docs, in docstrings, or even on the web in blog posts, articles, and such.

4.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/InstituteforDiseaseModeling/laser-measles/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

4.4 Development

To set up *laser-measles* for local development:

1. Fork laser-measles (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:YOURGITHUBNAME/laser-measles.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

4. When you're done making changes run all the checks and docs builder with one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

4.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`).
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

4.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel*:

```
tox -p auto
```

CHAPTER

FIVE

AUTHORS

- Christopher Lorton - https://www.idmod.org

CHAPTER

SIX

CHANGELOG

6.1 0.0.1 (2024-10-18)

- First release on PyPI.

CHAPTER

SEVEN

LASER_MEASLES

7.1 laser_measles package

class Births(*model*, *verbose=False*)

Bases: object

A component to handle the birth events in a model.

model

The model instance containing population and parameters.

verbose

Flag to enable verbose output. Default is False.

Type
bool

initializers

List of initializers to be called on birth events.

Type
list

metrics

DataFrame to holding timing metrics for initializers.

Type
DataFrame

property initializers

Returns the initializers to call on new agent births.

This method retrieves the initializers that are used to set up the initial state or configuration for agents at birth.

Returns
A list of initializers - instances of objects with an *on_birth* method.

Return type
list

property metrics

Returns the timing metrics for the births initializers.

This method retrieves the timing metrics for the births initializers.

Returns
A Pandas DataFrame of timing metrics for the births initializers.

Return type
DataFrame

`plot`(*fig=None*)

Plots the births in the top 5 most populous patches and a pie chart of birth initializer times.

Parameters
fig (*Figure, optional*) – A matplotlib Figure object. If None, a new figure will be created. Defaults to None.

Yields
None – This function yields twice to allow for intermediate plotting steps.

`class Incubation`(*model*, *verbose=False*)

Bases: `object`

A component to update the incubation timers of a population in a model.

`static nb_set_etimers`(*istart*, *iend*, *incubation*, *value*)

Numba compiled function to set exposure timers for a range of agents in parallel.

Return type
None

`static nb_update_exposure_timers`(*count*, *etimers*, *itimers*, *inf_mean*, *inf_std*)

Numba compiled function to check and update exposure timers for the population in parallel.

Return type
None

`on_birth`(*model*, *_tick*, *istart*, *iend*)

This function is called when a birth event occurs in the model. It sets the incubation timer for the newborns to zero, indicating that they are not incubating/exposed.

Parameters

- **model** (*object*) – The model instance containing the population data.
- **tick** (*int*) – The current tick or time step in the simulation (unused in this function).
- **istart** (*int*) – The start index of the newborns in the population array.
- **iend** (*int*) – The end index of the newborns in the population array.

Return type
None

Returns
None

`plot`(*fig=None*)

Plots the incubation timer values for currently incubating (exposed) agents in the population.

Parameters
fig (*Figure, optional*) – A Matplotlib Figure object. If None, a new figure will be created with default size and DPI.

Yields
None – This function uses a generator to yield control back to the caller.

class Infection(*model*, *verbose=False*)

Bases: object

A component to update the infection timers of a population in a model.

static nb_infection_update(*count*, *itimers*)

Numba compiled function to check and update infection timers for the population in parallel.

static nb_set_itimers(*istart*, *iend*, *itimers*, *value*)

Numba compiled function to set infection timers for a range of individuals in parallel.

Return type
None

on_birth(*model*, *_tick*, *istart*, *iend*)

This function sets the infection timer for newborns to zero, indicating that they are not infectious.

Parameters

- **model** – The simulation model containing the population data.
- **tick** – The current tick or time step in the simulation (unused in this function).
- **istart** – The starting index of the newborns in the population array.
- **iend** – The ending index of the newborns in the population array.

Return type
None

Returns
None

plot(*fig=None*)

Plots the distribution of infections by age.

This function creates a bar chart showing the number of individuals in each age group, and overlays a bar chart showing the number of infected individuals in each age group.

Parameters
fig (*Figure, optional*) – A Matplotlib Figure object to plot on. If None, a new figure is created.

Yields
None – This function uses a generator to yield control back to the caller.

class MaternalAntibodies(*model*, *verbose=False*)

Bases: object

A component to manage maternal antibodies in a population model.

static nb_update_ma_timers(*count*, *ma_timers*, *susceptibility*)

Numba compiled function to check and update maternal antibody timers for the population in parallel.

on_birth(*model*, *_tick*, *istart*, *iend*)

This function is called when births occur in the model. It updates the susceptibility and maternal antibody timers for newborns.

Parameters

- **model** (*object*) – The model instance containing the population data.
- **tick** (*int*) – The current tick or time step in the simulation (unused in this function).

- **istart** (*int*) – The starting index of the newborns in the population array.
- **iend** (*int*) – The ending index of the newborns in the population array.

Return type
None

Returns
None

plot(*fig=None*)

Plots a pie chart showing the distribution of infants with and without maternal antibodies.

Parameters
fig (*Figure, optional*) – A Matplotlib Figure object. If None, a new figure will be created with default size and DPI.

Returns
None

class Model(*scenario*, *parameters*, *name='measles'*)

Bases: object

A class to represent a simulation model for measles spread.

Parameters

- **scenario** (*pd.DataFrame*) – A DataFrame containing the scenario data, including population, latitude, and longitude.
- **parameters** (*PropertySet*) – A set of parameters for the model, including seed, nticks, k, a, b, c, max_frac, cbr, verbose, and pyramid_file.
- **name** (*str, optional*) – The name of the model. Defaults to "measles".

Notes

This class initializes the model with the given scenario and parameters. The scenario DataFrame must include the following columns:

- *name* (string): The name of the patch or location.
- *population* (integer): The population count for the patch.
- *latitude* (float degrees): The latitude of the patch (e.g., from geographic or population centroid).
- *longitude* (float degrees): The longitude of the patch (e.g., from geographic or population centroid).

property components: list

Retrieve the list of model components.

Returns
A list containing the components.

Return type
list

plot(*fig=None*)

Plots various visualizations related to the scenario and population data.

Parameters
fig (*Figure, optional*) – A matplotlib Figure object to use for plotting. If None, a new figure will be created.

Yields
None – This function uses a generator to yield control back to the caller after each plot is created.

The function generates three plots:

1. A scatter plot of the scenario patches and populations.
2. A histogram of the distribution of the day of birth for the initial population.
3. A pie chart showing the distribution of update phase times.

`run()`

Execute the model for a specified number of ticks, recording the time taken for each phase.

This method initializes the start time, iterates over the number of ticks specified in the model parameters, and for each tick, it executes each phase of the model while recording the time taken for each phase.

The metrics for each tick are stored in a list. After completing all ticks, it records the finish time and, if verbose mode is enabled, prints a summary of the timing metrics.

`tstart`

The start time of the model execution.
Type
datetime

`tfinish`

The finish time of the model execution.
Type
datetime

`metrics`

A list of timing metrics for each tick and phase.
Type
list

Return type
None

Returns
None

`visualize(pdf=True)`

Visualize each compoonent instances either by displaying plots or saving them to a PDF file.

Parameters
pdf (*bool*) – If True, save the plots to a PDF file. If False, display the plots interactively. Default is True.

Return type
None

Returns
None

`class NonDiseaseDeaths(model, verbose=False)`

Bases: `object`

A component to model non-disease related deaths in a population.

on_birth(*model*, *tick*, *istart*, *iend*)

Handles the birth of new agents in the model.

This function updates the population's alive status and predicted date of death (dod) for newly born agents. It also pushes agents with a date of death within the simulation's maximum ticks to the non-disease death queue (nddq).

Parameters

- **model** (*object*) – The simulation model containing population and parameters.
- **tick** (*int*) – The current tick or time step in the simulation.
- **istart** (*int*) – The starting index of the newly born agents in the population array.
- **iend** (*int*) – The ending index of the newly born agents in the population array.

Returns
None

plot(*fig=None*)

Plots the cumulative non-disease deaths for the year 0 population.

Parameters
fig (*Figure, optional*) – A matplotlib Figure object. If None, a new figure is created. Defaults to None.

Returns
None

Yields
None

Notes:

- The function plots two lines:
  1. The cumulative number of non-disease deaths over the years since birth, marked with red 'x'.
  2. The expected cumulative deaths based on the model's estimator, marked with blue '+'.
- If no individuals are found born in the first year, a message is printed.

class RoutineImmunization(*model*, *verbose=False*)

Bases: object

A component to handle the routine immunization process within a model.

static nb_update_ri_timers(*count*, *ri_timers*, *susceptibility*)

Numba compiled function to check and update routine immunization timers for the population in parallel.

on_birth(*model*, *_tick*, *istart*, *iend*)

Handles the birth event in the model by setting the MCV (Measles Conjugate Vaccine) status (effective MCV1, effective MCV2, or unvaccinated/ineffective vaccination) and initializing the MCV timers for the newborns.

Parameters

- **model** (*object*) – The model instance containing the population data.
- **tick** (*int*) – The current tick or time step in the simulation (unused in this function).
- **istart** (*int*) – The starting index of the newborns in the population array.
- **iend** (*int*) – The ending index of the newborns in the population array.

Returns
None

`plot`(*fig=None*)

Plots a pie chart representing the routine immunization status of the population born during the simulation (initial agent population does not have MCV status computed or tracked).

Parameters
fig (*Figure, optional*) – A matplotlib Figure object to plot on. If None, a new figure is created.

Raises
AssertionError – If the sum of unvaccinated, MCV1, and MCV2 individuals does not match the total number of individuals.

Yields
None

`class Susceptibility`(*model*, *verbose=False*)

Bases: `object`

A component to represent the susceptibility of a population in a model.

`static nb_initialize_susceptibility`(*count*, *dob*, *susceptibility*)

Numba compiled function to initialize susceptibility based on date of birth.

Return type
None

`static nb_set_susceptibility`(*istart*, *iend*, *susceptibility*, *value*)

Numba compiled function to set the susceptibility of a range of individuals.

Return type
None

`on_birth`(*model*, *_tick*, *istart*, *iend*)

Handle the birth event in the model by setting the susceptibility of newborns.

This method is called when a birth event occurs in the model. It sets the susceptibility of the newborns to 0, indicating that they are not susceptible to the disease.

Parameters

- **model** (*object*) – The model object containing the population data.
- **tick** (*int*) – The current tick or time step in the simulation.
- **istart** (*int*) – The starting index of the newborns in the population array.
- **iend** (*int*) – The ending index of the newborns in the population array.

Returns
None

`plot`(*fig=None*)

Plots the susceptibility distribution by age.

Parameters
fig (*Figure, optional*) – A Matplotlib Figure object. If None, a new figure is created with a size of 12x9 inches and a DPI of 128.

Yields
None – This function uses a generator to yield control back to the caller.

class Transmission(*model*, *verbose=False*)

Bases: object

A component to model the transmission of disease in a population.

static nb_transmission_update(*susceptibilities*, *nodeids*, *forces*, *etimers*, *count*, *exp_shape*, *exp_scale*, *incidence*)

Numba compiled function to stochastically transmit infection to agents in parallel.

plot(*fig=None*)

Plots the cases and incidence for the two largest patches in the model.

This function creates a figure with four subplots:

- Cases for the largest patch
- Incidence for the largest patch
- Cases for the second largest patch
- Incidence for the second largest patch

If no figure is provided, a new figure is created with a size of 12x9 inches and a DPI of 128.

Parameters

fig (*Figure, optional*) – A Matplotlib Figure object to plot on. If None, a new figure is created.

Yields

None

compute()

Docstring for compute function.

7.1.1 Subpackages

laser_measles.biweekly package

class BaseScenario(*df*)

Bases: object

unwrap()

Return type

DataFrame

class BiweeklyModel(*scenario*, *parameters*, *name='biweekly'*)

Bases: object

A class to represent the biweekly model.

Parameters

- **scenario** (*pl.DataFrame*) – A DataFrame containing the scenario data, including population, latitude, and longitude.
- **parameters** (BiweeklyParams) – A set of parameters for the model.
- **name** (*str, optional*) – The name of the model. Defaults to “template”.

Notes

This class initializes the model with the given scenario and parameters. The scenario DataFrame must include the following columns:

- *ids* (string): The name of the patch or location.
- *pop* (integer): The population count for the patch.
- *lat* (float degrees): The latitude of the patches (e.g., from geographic or population centroid).
- *lon* (float degrees): The longitude of the patches (e.g., from geographic or population centroid).
- *mcv1* (float): The MCV1 coverage for the patches.

property components: list

Retrieve the list of model components.

Returns
A list containing the components.

Return type
list

run()

Execute the model for a specified number of ticks, recording the time taken for each phase.

This method initializes the start time, iterates over the number of ticks specified in the model parameters, and for each tick, it executes each phase of the model while recording the time taken for each phase.

The metrics for each tick are stored in a list. After completing all ticks, it records the finish time and, if verbose mode is enabled, prints a summary of the timing metrics.

tstart

The start time of the model execution.
Type
datetime

tfinish

The finish time of the model execution.
Type
datetime

metrics

A list of timing metrics for each tick and phase.
Type
list

Return type
None

Returns
None

step(*tick*)

Return type
None

class BiweeklyParams(**data)

Bases: BaseModel

Parameters for the biweekly model.

property mixing: ndarray

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

property time_step_days: int

beta: float

crude_birth_rate: float

crude_death_rate: float

distance_exponent: float

mixing_scale: float

nticks: int

seasonality: float

season_start: int

seed: int

start_time: str

states: list[str]

verbose: bool

Subpackages

laser_measles.biweekly.components package

class FadeOutTracker(*model*, *verbose=False*)

Bases: *BaseComponent*

Component for tracking the number of nodes with fade-outs.

class Infection(*model*, *verbose=False*)

Bases: *BaseComponent*

Component for simulating the spread of infection in the model

class VitalDynamics(*model*, *verbose=False*)

Bases: *BaseComponent*

Component for simulating the vital dynamics in the model

Submodules

laser_measles.biweekly.components.fadeout_tracker module

class FadeOutTracker(*model*, *verbose=False*)

Bases: *BaseComponent*

Component for tracking the number of nodes with fade-outs.

laser_measles.biweekly.components.infection module

cast_type(*a*, *dtype*)

class Infection(*model*, *verbose=False*)

Bases: *BaseComponent*

Component for simulating the spread of infection in the model

laser_measles.biweekly.components.routine_immunization module

laser_measles.biweekly.components.vital_dynamics module

cast_type(*a*, *dtype*)

class VitalDynamics(*model*, *verbose=False*)

Bases: *BaseComponent*

Component for simulating the vital dynamics in the model

Submodules

laser_measles.biweekly.base module

Basic classes

class BaseComponent(*model*, *verbose=False*)

Bases: `object`

plot(*fig=None*)

Placeholder for plotting method.

class BaseScenarioSchema(***data*)

Bases: `BaseModel`

Schema for the scenario data.

pop: list[int]

lat: list[float]

lon: list[float]

ids: list[str | int]

mcv1: list[float]

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class BaseScenario(*df*)

Bases: object

unwrap()

Return type
DataFrame

laser_measles.biweekly.mixing module

pairwise_haversine(*lon*, *lat*)

Calculate pairwise distances for all (lon, lat) points using the Haversine formula.

Parameters

- **lon** (ndarray) – Array of longitude values in degrees
- **lat** (ndarray) – Array of latitude values in degrees

Return type
ndarray

Returns
Compressed matrix of pairwise distances in kilometers, where only the upper triangle (excluding diagonal) is stored. The full matrix can be reconstructed using: full_matrix = np.zeros((n, n)) full_matrix[np.triu_indices(n, k=1)] = compressed_matrix full_matrix = full_matrix + full_matrix.T

init_gravity_diffusion(*df*, *scale*, *dist_exp*)

Initialize a gravity diffusion matrix for population mixing.

Parameters

- **df** (DataFrame | tuple[ndarray, ndarray]) – Either a DataFrame with 'population', 'lat', and 'lon' columns, or a tuple of (lon, lat) arrays
- **scale** (float) – Scaling factor for the diffusion matrix
- **dist_exp** (float) – Distance exponent for the gravity model

Return type
ndarray

Returns
Normalized diffusion matrix where each row sums to 1

laser_measles.biweekly.model module

This module defines the *Model* class for simulation

Classes:
Model: A class to represent the simulation model.

Imports:

Model Class:

Methods:

__init__(self, scenario: pd.DataFrame, parameters: PropertySet, name: str = "template") -> None:
Initializes the model with the given scenario and parameters.

components(self) -> list:
Gets the list of components in the model.

components(self, components: list) -> None:
Sets the list of components in the model and initializes instances and phases.

__call__(self, model, tick: int) -> None:
Updates the model for a given tick.

run(self) -> None:
Runs the model for the specified number of ticks.

visualize(self, pdf: bool = True) -> None:
Generates visualizations of the model's results, either displaying them or saving to a PDF.

plot(self, fig: Figure = None):
Generates plots for the scenario patches and populations, distribution of day of birth, and update phase times.

`class BiweeklyModel`(*scenario*, *parameters*, *name='biweekly'*)

Bases: `object`

A class to represent the biweekly model.

Parameters

- **scenario** (*pl.DataFrame*) – A DataFrame containing the scenario data, including population, latitude, and longitude.
- **parameters** (BiweeklyParams) – A set of parameters for the model.
- **name** (*str, optional*) – The name of the model. Defaults to "template".

Notes

This class initializes the model with the given scenario and parameters. The scenario DataFrame must include the following columns:

- *ids* (string): The name of the patch or location.
- *pop* (integer): The population count for the patch.
- *lat* (float degrees): The latitude of the patches (e.g., from geographic or population centroid).
- *lon* (float degrees): The longitude of the patches (e.g., from geographic or population centroid).
- *mcv1* (float): The MCV1 coverage for the patches.

`property components: list`

Retrieve the list of model components.

Returns
A list containing the components.

Return type
list

`step`(*tick*)

Return type
None

run()

Execute the model for a specified number of ticks, recording the time taken for each phase.

This method initializes the start time, iterates over the number of ticks specified in the model parameters, and for each tick, it executes each phase of the model while recording the time taken for each phase.

The metrics for each tick are stored in a list. After completing all ticks, it records the finish time and, if verbose mode is enabled, prints a summary of the timing metrics.

tstart

The start time of the model execution.

Type
datetime

tfinish

The finish time of the model execution.

Type
datetime

metrics

A list of timing metrics for each tick and phase.

Type
list

Return type
None

Returns
None

laser_measles.biweekly.params module

class BiweeklyParams(***data*)

Bases: BaseModel

Parameters for the biweekly model.

beta: float

crude_birth_rate: float

crude_death_rate: float

distance_exponent: float

mixing_scale: float

nticks: int

seasonality: float

season_start: int

seed: int

start_time: str

states: list[str]

verbose: bool

property time_step_days: int

property mixing: ndarray

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

laser_measles.demographics package

Submodules

laser_measles.demographics.admin_shapefile module

laser_measles.demographics.base module

laser_measles.demographics.cache module

laser_measles.demographics.gadm module

laser_measles.demographics.raster_patch module

laser_measles.demographics.shapefiles module

laser_measles.generic package

Submodules

laser_measles.generic.metapop module

laser_measles.generic.model module

laser_measles.generic.params module

laser_measles.nigeria package

Submodules

laser_measles.nigeria.lgas module

laser_measles.nigeria.metapop module

laser_measles.nigeria.model module

laser_measles.nigeria.params module

7.1.2 Submodules

7.1.3 laser_measles.antibodies module

This module defines the MaternalAntibodies class and the nb_update_ma_timers function for simulating the presence of maternal antibodies in a population model.

Classes:

MaternalAntibodies: Manages the maternal antibodies for a population model, including initialization, updates, and plotting.

Usage:

The MaternalAntibodies class should be instantiated with a model object and can be called to update the model at each tick. It also provides a method to handle newborns and a method to plot the current state of maternal antibodies in the population.

class MaternalAntibodies(*model*, *verbose=False*)

Bases: `object`

A component to manage maternal antibodies in a population model.

static nb_update_ma_timers(*count*, *ma_timers*, *susceptibility*)

Numba compiled function to check and update maternal antibody timers for the population in parallel.

on_birth(*model*, *_tick*, *istart*, *iend*)

This function is called when births occur in the model. It updates the susceptibility and maternal antibody timers for newborns.

Parameters

- **model** (*object*) – The model instance containing the population data.
- **tick** (*int*) – The current tick or time step in the simulation (unused in this function).
- **istart** (*int*) – The starting index of the newborns in the population array.
- **iend** (*int*) – The ending index of the newborns in the population array.

Return type
None

Returns
None

plot(*fig=None*)

Plots a pie chart showing the distribution of infants with and without maternal antibodies.

Parameters
fig (*Figure, optional*) – A Matplotlib Figure object. If None, a new figure will be created with default size and DPI.

Returns
None

7.1.4 laser_measles.births module

This module defines the Births component, which is responsible for simulating births in a population model.

Classes:

Births:

Manages the birth process within a population model, including initializing births, updating population data, and plotting birth statistics.

Usage:

The Births component requires a model with a *population* attribute that has a *dob* attribute. It calculates the number of births based on the model's parameters and updates the population accordingly. It also provides methods to plot birth statistics.

Example

model = YourModelClass() births = Births(model) births(model, tick) births.plot()

model

The population model.

Type
object

_initializers

List of initializers to be called on birth.

Type
list

_metrics

List to store timing metrics for initializers.

Type
list

class Births(*model*, *verbose=False*)

Bases: object

A component to handle the birth events in a model.

model

The model instance containing population and parameters.

verbose

Flag to enable verbose output. Default is False.

Type
bool

initializers

List of initializers to be called on birth events.

Type
list

metrics

DataFrame to holding timing metrics for initializers.

Type
DataFrame

property initializers

Returns the initializers to call on new agent births.

This method retrieves the initializers that are used to set up the initial state or configuration for agents at birth.

Returns
A list of initializers - instances of objects with an *on_birth* method.

Return type
list

property metrics

Returns the timing metrics for the births initializers.

This method retrieves the timing metrics for the births initializers.

Returns
A Pandas DataFrame of timing metrics for the births initializers.

Return type
DataFrame

plot(*fig=None*)

Plots the births in the top 5 most populous patches and a pie chart of birth initializer times.

Parameters
fig (*Figure, optional*) – A matplotlib Figure object. If None, a new figure will be created. Defaults to None.

Yields
None – This function yields twice to allow for intermediate plotting steps.

7.1.5 laser_measles.cli module

Module that contains the command line app.

Why does this file exist, and why not put this in __main__?

You might be tempted to import things from __main__ later, but that will cause problems: the code will get executed twice:

- When you run *python -mlaser_measles* python will execute `__main__.py` as a script. That means there will not be any `laser_measles.__main__` in `sys.modules`.
- When you import __main__ it will get executed again (as a module) because there"s no `laser_measles.__main__` in `sys.modules`.

Also see (1) from https://click.palletsprojects.com/en/stable/setuptools/

7.1.6 laser_measles.core module

compute()

Docstring for compute function.

7.1.7 laser_measles.incubation module

This module defines the Incubation class, which simulates the incubation period of a disease within a population model.

Classes:
Incubation: Manages the incubation period of a disease, updating exposure timers and handling birth events.

__init__(self, model, verbose

bool = False) -> None: Initializes the Incubation instance with the given model and optional verbosity.

__call__(*self*, *model*, *tick*) → None

Updates the exposure timers for the population at each tick of the simulation.

nb_update_exposure_timers(*count*, *etimers*, *itimers*, *inf_mean*, *inf_std*) → None

Numba-optimized static method to update exposure timers and set infection timers when exposure ends.

on_birth(*self*, *model*, *_tick*, *istart*, *iend*) → None

Handles the birth event by setting the exposure timers of newborns to zero.

nb_set_etimers(*istart*, *iend*, *incubation*, *value*) → None

Numba-optimized static method to set exposure timers for a range of individuals.

plot(self, fig

Figure = None): Plots the distribution of the incubation period using matplotlib.

model

The population model associated with the incubation period.

class Incubation(*model*, *verbose=False*)

Bases: `object`

A component to update the incubation timers of a population in a model.

static nb_update_exposure_timers(*count*, *etimers*, *itimers*, *inf_mean*, *inf_std*)

Numba compiled function to check and update exposure timers for the population in parallel.

Return type
None

on_birth(*model*, *_tick*, *istart*, *iend*)

This function is called when a birth event occurs in the model. It sets the incubation timer for the newborns to zero, indicating that they are not incubating/exposed.

Parameters

- **model** (*object*) – The model instance containing the population data.
- **tick** (*int*) – The current tick or time step in the simulation (unused in this function).
- **istart** (*int*) – The start index of the newborns in the population array.
- **iend** (*int*) – The end index of the newborns in the population array.

Return type
None

Returns
None

static nb_set_etimers(*istart*, *iend*, *incubation*, *value*)

Numba compiled function to set exposure timers for a range of agents in parallel.

Return type
None

plot(*fig=None*)

Plots the incubation timer values for currently incubating (exposed) agents in the population.

Parameters
fig (*Figure, optional*) – A Matplotlib Figure object. If None, a new figure will be created with default size and DPI.

Yields
None – This function uses a generator to yield control back to the caller.

7.1.8 laser_measles.infection module

This module defines the Infection class, which models the infection dynamics within a population.

Classes:

Infection: A class to handle infection updates, initialization, and plotting of infection data.

Functions:

Infection.__init__(self, model, verbose: bool = False) -> None:
Initializes the Infection class with a given model and verbosity option.

Infection.__call__(self, model, tick) -> None:
Updates the infection status of the population at each tick.

Infection.nb_infection_update(count, itimers):
A static method that updates the infection timers for the population using Numba for performance.

Infection.on_birth(self, model, _tick, istart, iend) -> None:
Resets the infection timer for newborns in the population.

Infection.nb_set_itimers(istart, iend, itimers, value) -> None:
A static method that sets the infection timers for a range of individuals in the population using Numba for performance.

Infection.plot(self, fig: Figure = None):
Plots the infection data by age using Matplotlib.

`class Infection`(*model*, *verbose=False*)

Bases: `object`

A component to update the infection timers of a population in a model.

`static nb_infection_update`(*count*, *itimers*)

Numba compiled function to check and update infection timers for the population in parallel.

`on_birth`(*model*, *_tick*, *istart*, *iend*)

This function sets the infection timer for newborns to zero, indicating that they are not infectious.

Parameters

- **model** – The simulation model containing the population data.
- **tick** – The current tick or time step in the simulation (unused in this function).
- **istart** – The starting index of the newborns in the population array.
- **iend** – The ending index of the newborns in the population array.

Return type
`None`

Returns
None

`static nb_set_itimers`(*istart*, *iend*, *itimers*, *value*)

Numba compiled function to set infection timers for a range of individuals in parallel.

Return type
`None`

plot(*fig=None*)

Plots the distribution of infections by age.

This function creates a bar chart showing the number of individuals in each age group, and overlays a bar chart showing the number of infected individuals in each age group.

Parameters

fig (*Figure, optional*) – A Matplotlib Figure object to plot on. If None, a new figure is created.

Yields

None – This function uses a generator to yield control back to the caller.

7.1.9 laser_measles.model module

This module defines the *Model* class for simulating the spread of measles using a gravity model for migration and demographic data for population initialization.

Classes:

Model: A class to represent the measles simulation model.

Imports:

- datetime: For handling date and time operations.
- click: For command-line interface utilities.
- numpy as np: For numerical operations.
- pandas as pd: For data manipulation and analysis.
- laser_core.demographics: For demographic data handling.
- laser_core.laserframe: For handling laser frame data structures.
- laser_core.migration: For migration modeling.
- laser_core.propertyset: For handling property sets.
- laser_core.random: For random number generation.
- matplotlib.pyplot as plt: For plotting.
- matplotlib.backends.backend_pdf: For PDF generation.
- matplotlib.figure: For figure handling.
- alive_progress: For progress bar visualization.
- laser_measles.measles_births: For handling measles birth data.
- laser_measles.utils: For utility functions.

Model Class:

Methods:

__init__(self, scenario: pd.DataFrame, parameters: PropertySet, name: str = "measles") -> None:
Initializes the model with the given scenario and parameters.

components(self) -> list:
Gets the list of components in the model.

components(self, components: list) -> None:
Sets the list of components in the model and initializes instances and phases.

__call__(self, model, tick: int) -> None:
Updates the model for a given tick.

run(self) -> None:
Runs the model for the specified number of ticks.

visualize(self, pdf: bool = True) -> None:
Generates visualizations of the model's results, either displaying them or saving to a PDF.

plot(self, fig: Figure = None):
Generates plots for the scenario patches and populations, distribution of day of birth, and update phase times.

`class Model`(*scenario*, *parameters*, *name='measles'*)

Bases: `object`

A class to represent a simulation model for measles spread.

Parameters

- **`scenario`** (*`pd.DataFrame`*) – A DataFrame containing the scenario data, including population, latitude, and longitude.
- **`parameters`** (*`PropertySet`*) – A set of parameters for the model, including seed, nticks, k, a, b, c, max_frac, cbr, verbose, and pyramid_file.
- **`name`** (*`str, optional`*) – The name of the model. Defaults to "measles".

Notes

This class initializes the model with the given scenario and parameters. The scenario DataFrame must include the following columns:

- *name* (string): The name of the patch or location.
- *population* (integer): The population count for the patch.
- *latitude* (float degrees): The latitude of the patch (e.g., from geographic or population centroid).
- *longitude* (float degrees): The longitude of the patch (e.g., from geographic or population centroid).

`property components: list`

Retrieve the list of model components.

Returns
A list containing the components.

Return type
list

`run`()

Execute the model for a specified number of ticks, recording the time taken for each phase.

This method initializes the start time, iterates over the number of ticks specified in the model parameters, and for each tick, it executes each phase of the model while recording the time taken for each phase.

The metrics for each tick are stored in a list. After completing all ticks, it records the finish time and, if verbose mode is enabled, prints a summary of the timing metrics.

`tstart`

The start time of the model execution.
Type
datetime

tfinish

The finish time of the model execution.

Type
datetime

metrics

A list of timing metrics for each tick and phase.

Type
list

Return type
None

Returns
None

visualize(*pdf=True*)

Visualize each compoonent instances either by displaying plots or saving them to a PDF file.

Parameters
pdf (*bool*) – If True, save the plots to a PDF file. If False, display the plots interactively. Default is True.

Return type
None

Returns
None

plot(*fig=None*)

Plots various visualizations related to the scenario and population data.

Parameters
fig (*Figure, optional*) – A matplotlib Figure object to use for plotting. If None, a new figure will be created.

Yields
None – This function uses a generator to yield control back to the caller after each plot is created.

The function generates three plots:

1. A scatter plot of the scenario patches and populations.
2. A histogram of the distribution of the day of birth for the initial population.
3. A pie chart showing the distribution of update phase times.

7.1.10 laser_measles.mortality module

This module defines the NonDiseaseDeaths class, which models non-disease related deaths in a population over time.

Classes:
NonDiseaseDeaths: A class to handle non-disease mortality in a population model.

Dependencies:

- click
- numpy as np
- laser_core.demographics.KaplanMeierEstimator

- laser_core.sortedqueue.SortedQueue
- matplotlib.pyplot as plt
- matplotlib.figure.Figure
- alive_progress

Usage:

The NonDiseaseDeaths class is initialized with a model and an optional verbosity flag. It adds scalar properties to the population, estimates dates of death using a Kaplan-Meier estimator, and manages a sorted queue of non-disease death events. The class provides methods to handle births, update the model at each tick, and plot cumulative non-disease deaths.

Example

model = SomeModel() non_disease_deaths = NonDiseaseDeaths(model) non_disease_deaths.on_birth(model, tick, istart, iend) non_disease_deaths(model, tick) non_disease_deaths.plot()

`class NonDiseaseDeaths(`*model*, *verbose=False*`)`

Bases: `object`

A component to model non-disease related deaths in a population.

`on_birth(`*model*, *tick*, *istart*, *iend*`)`

Handles the birth of new agents in the model.

This function updates the population's alive status and predicted date of death (dod) for newly born agents. It also pushes agents with a date of death within the simulation's maximum ticks to the non-disease death queue (nddq).

Parameters

- **model** (*object*) – The simulation model containing population and parameters.
- **tick** (*int*) – The current tick or time step in the simulation.
- **istart** (*int*) – The starting index of the newly born agents in the population array.
- **iend** (*int*) – The ending index of the newly born agents in the population array.

Returns

None

`plot(`*fig=None*`)`

Plots the cumulative non-disease deaths for the year 0 population.

Parameters

fig (*Figure, optional*) – A matplotlib Figure object. If None, a new figure is created. Defaults to None.

Returns

None

Yields

None

Notes:

- The function plots two lines:
  1. The cumulative number of non-disease deaths over the years since birth, marked with red 'x'.
  2. The expected cumulative deaths based on the model's estimator, marked with blue '+'.

- If no individuals are found born in the first year, a message is printed.

7.1.11 laser_measles.routine module

This module implements the Routine Immunization (RI) process for a population model. It includes the initialization of RI coverage for patches, the assignment of MCV (Measles Containing Vaccine) status to agents, and the updating of RI timers for agents.

Classes:

RoutineImmunization: Manages the routine immunization process, including initialization, updating RI timers,
handling births, and plotting immunization status.

Functions:

nb_update_ri_timers(count, ri_timers, susceptibility): Numba-optimized function to update RI timers and adjust
susceptibility when timers expire.

set_mcv_status(model, istart, iend): Assigns MCV status to agents based on RI coverage and probabilities of MCV1
and MCV2 take.

set_mcv_timers(model, istart, iend): Sets the RI timers for agents based on their MCV status and predefined
time ranges for MCV1 and MCV2.

Constants:

GET_MCV1: Constant representing the status of an agent who has received an effective MCV1 vaccination. GET_MCV2: Constant representing the status of an agent who has received an effective MCV2 vaccination. GET_NONE: Constant representing the status of an unvaccinated agent or an agent with ineffective vaccination.

`class RoutineImmunization(`*model*, *verbose=False*`)`

Bases: `object`

A component to handle the routine immunization process within a model.

`static nb_update_ri_timers(`*count*, *ri_timers*, *susceptibility*`)`

Numba compiled function to check and update routine immunization timers for the population in parallel.

`on_birth(`*model*, *_tick*, *istart*, *iend*`)`

Handles the birth event in the model by setting the MCV (Measles Conjugate Vaccine) status (effective MCV1, effective MCV2, or unvaccinated/ineffective vaccination) and initializing the MCV timers for the newborns.

Parameters

- **model** (*object*) – The model instance containing the population data.
- **tick** (*int*) – The current tick or time step in the simulation (unused in this function).
- **istart** (*int*) – The starting index of the newborns in the population array.
- **iend** (*int*) – The ending index of the newborns in the population array.

Returns
None

plot(*fig=None*)

Plots a pie chart representing the routine immunization status of the population born during the simulation (initial agent population does not have MCV status computed or tracked).

Parameters

fig (*Figure, optional*) – A matplotlib Figure object to plot on. If None, a new figure is created.

Raises

AssertionError – If the sum of unvaccinated, MCV1, and MCV2 individuals does not match the total number of individuals.

Yields

None

set_mcv_status(*model*, *istart*, *iend*)

Set the MCV (Measles Conjugate Vaccine) status for a subset of the population. This function assigns (effective) MCV1, (effective) MCV2, or NONE (no or ineffective vaccination) status to individuals in the population based on the model's parameters and random draws. The MCV1 and MCV2 statuses are determined by the coverage and probability of vaccine take specified in the model.

Parameters

- **model** (*object*) – The model containing population and parameters for MCV coverage and take probabilities.
- **istart** (*int*) – The starting index of the population subset to update.
- **iend** (*int*) – The ending index (exclusive) of the population subset to update.

Returns

None

set_mcv_timers(*model*, *istart*, *iend*)

Set the MCV (Measles Containing Vaccine) timers for a subset of the population in the model.

This function assigns random timer values for MCV1 or MCV2 vaccinations to individuals in the population based on the specified start and end indices. The timer values are generated using the model's pseudo-random number generator (PRNG) and are constrained within the start and end parameters for MCV1 and MCV2.

Parameters

- **model** (*object*) – The model object containing the population and parameters.
- **istart** (*int*) – The starting index of the subset of the population.
- **iend** (*int*) – The ending index of the subset of the population.

Returns

None

7.1.12 laser_measles.susceptibility module

This module defines the Susceptibility class and associated functions for managing and visualizing the susceptibility of a population to measles based on age.

Classes:

Susceptibility: Manages the susceptibility property of a population and provides methods for updating and plotting susceptibility data.

Functions:

nb_initialize_susceptibility(count, dob, susceptibility):
Initializes the susceptibility of individuals in the population based on their date of birth.

nb_set_susceptibility(istart, iend, susceptibility, value):
Sets the susceptibility of a range of individuals in the population to a specified value.

`class Susceptibility(`*model*, *verbose=False*`)`

Bases: `object`

A component to represent the susceptibility of a population in a model.

`static nb_initialize_susceptibility(`*count*, *dob*, *susceptibility*`)`

Numba compiled function to initialize susceptibility based on date of birth.

Return type
`None`

`static nb_set_susceptibility(`*istart*, *iend*, *susceptibility*, *value*`)`

Numba compiled function to set the susceptibility of a range of individuals.

Return type
`None`

`on_birth(`*model*, *_tick*, *istart*, *iend*`)`

Handle the birth event in the model by setting the susceptibility of newborns.

This method is called when a birth event occurs in the model. It sets the susceptibility of the newborns to 0, indicating that they are not susceptible to the disease.

Parameters

- **model** (*object*) – The model object containing the population data.
- **tick** (*int*) – The current tick or time step in the simulation.
- **istart** (*int*) – The starting index of the newborns in the population array.
- **iend** (*int*) – The ending index of the newborns in the population array.

Returns
None

`plot(`*fig=None*`)`

Plots the susceptibility distribution by age.

Parameters
fig (*Figure, optional*) – A Matplotlib Figure object. If None, a new figure is created with a size of 12x9 inches and a DPI of 128.

Yields
None – This function uses a generator to yield control back to the caller.

7.1.13 laser_measles.transmission module

This module defines the Transmission class, which models the transmission of measles in a population.

Classes:

Transmission: A class to model the transmission dynamics of measles within a population.

Functions:

Transmission.__init__(self, model, verbose: bool = False) -> None:

Initializes the Transmission object with the given model and verbosity.

Transmission.__call__(self, model, tick) -> None:

Executes the transmission dynamics for a given model and tick.

Transmission.nb_transmission_update(susceptibilities, nodeids, forces, etimers, count, exp_shape, exp_scale, incidence):

A Numba-compiled static method to update the transmission dynamics in parallel.

Transmission.plot(self, fig: Figure = None):

Plots the cases and incidence for the two largest patches in the model.

`class Transmission`(*model*, *verbose=False*)

Bases: `object`

A component to model the transmission of disease in a population.

`static nb_transmission_update`(*susceptibilities*, *nodeids*, *forces*, *etimers*, *count*, *exp_shape*, *exp_scale*, *incidence*)

Numba compiled function to stochastically transmit infection to agents in parallel.

`plot`(*fig=None*)

Plots the cases and incidence for the two largest patches in the model.

This function creates a figure with four subplots:

- Cases for the largest patch
- Incidence for the largest patch
- Cases for the second largest patch
- Incidence for the second largest patch

If no figure is provided, a new figure is created with a size of 12x9 inches and a DPI of 128.

Parameters
`fig` (*Figure, optional*) – A Matplotlib Figure object to plot on. If None, a new figure is created.

Yields
None

7.1.14 laser_measles.utils module

This module provides utility functions for the laser-measles project.

Functions:

calc_distances(latitudes: np.ndarray, longitudes: np.ndarray, verbose: bool = False) -> np.ndarray:
Calculate the pairwise distances between points given their latitudes and longitudes.

calc_capacity(population: np.uint32, nticks: np.uint32, cbr: np.float32, verbose: bool = False) -> np.uint32:
Calculate the population capacity after a given number of ticks based on a constant birth rate.

seed_infections_randomly(model, ninfections: int = 100) -> None:
Seed initial infections in random locations at the start of the simulation.

seed_infections_in_patch(model, ipatch: int, ninfections: int = 100) -> None:
Seed initial infections in a specific location at the start of the simulation.

calc_distances(*latitudes*, *longitudes*, *verbose=False*)

Calculate the pairwise distances between points given their latitudes and longitudes.

Parameters

- **latitudes** (*np.ndarray*) – A 1-dimensional array of latitudes.
- **longitudes** (*np.ndarray*) – A 1-dimensional array of longitudes with the same shape as latitudes.
- **verbose** (*bool, optional*) – If True, prints the upper left corner of the distance matrix. Default is False.

Returns

A 2-dimensional array where the element at [i, j] represents the distance between the i-th and j-th points.

Return type

np.ndarray

Raises

AssertionError – If latitudes is not 1-dimensional or if latitudes and longitudes do not have the same shape.

calc_capacity(*population*, *nticks*, *cbr*, *verbose=False*)

Calculate the population capacity after a given number of ticks based on a constant birth rate (CBR).

Parameters

- **population** (*np.uint32*) – The initial population.
- **nticks** (*np.uint32*) – The number of ticks (time steps) to simulate.
- **cbr** (*np.float32*) – The constant birth rate per 1000 people per year.
- **verbose** (*bool, optional*) – If True, prints detailed population growth information. Defaults to False.

Returns

The estimated population capacity after the given number of ticks.

Return type

np.uint32

seed_infections_randomly(*model*, *ninfections=100*)

Seed initial infections in random locations at the start of the simulation. This function randomly selects individuals from the population and seeds them with an infection, based on the specified number of initial infections.

Parameters

- **model** – The simulation model containing the population and parameters.
- **ninfections** (*int, optional*) – The number of initial infections to seed. Defaults to 100.

Return type

None

Returns

None

seed_infections_in_patch(*model*, *ipatch*, *ninfections=100*)

Seed initial infections in a specific patch of the population at the start of the simulation. This function randomly selects individuals from the specified patch and sets their infection timer to the mean infection duration, effectively marking them as infected. The process continues until the desired number of initial infections is reached.

Parameters

- **model** – The simulation model containing the population and parameters.
- **ipatch** (*int*) – The identifier of the patch where infections should be seeded.
- **ninfections** (*int, optional*) – The number of initial infections to seed. Defaults to 100.

Return type
None

Returns
None

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

B

C

D

F

I

L

M

N

O

P

R

S

T

U

V