epidemik

Release 0.2.0

Bruno Gonçalves

Sep 15, 2026

USER GUIDE

1 Installation **3**
- 1.1 From PyPI . . . 3
- 1.2 From source . . . 3
- 1.3 Dependencies . . . 3

2 Quick start **5**
- 2.1 Building an SIR model . . . 5
- 2.2 Deterministic dynamics . . . 6
- 2.3 Stochastic dynamics . . . 6
- 2.4 Where to go next . . . 6

3 EpiModel: homogeneously mixed models **7**
- 3.1 Defining the dynamics . . . 7
- 3.2 Named parameters . . . 7
- 3.3 Computing R_0 . . . 8
- 3.4 Running the dynamics . . . 8
- 3.5 Stepping manually . . . 9
- 3.6 Vaccination . . . 9
- 3.7 Demographics: births and deaths . . . 9
- 3.8 Multi-group and vector-borne models . . . 9
- 3.9 Within-host models . . . 10
- 3.10 Age structure . . . 11
- 3.11 Seasonality . . . 11
- 3.12 Accessing and visualising results . . . 11
- 3.13 Saving, loading and sharing models . . . 12

4 NetworkEpiModel: epidemics on a contact network **13**
- 4.1 Building a network model . . . 13
- 4.2 Running a simulation . . . 13
- 4.3 R_0 on a network . . . 14
- 4.4 What is *not* supported . . . 14

5 MetaEpiModel: metapopulation models **15**
- 5.1 Building a metapopulation model . . . 15
- 5.2 Seeding and simulating . . . 16
- 5.3 Inspecting the result . . . 16
- 5.4 What is *not* supported . . . 16

6 API reference **17**
- 6.1 epidemik . . . 17
- 6.2 Package overview . . . 17

6.3 EpiModel . . . 17
6.4 NetworkEpiModel . . . 28
6.5 MetaEpiModel . . . 31
6.6 epidemik.utils . . . 36

7 Changelog **39**
7.1 0.2.0 (2026-09-15) . . . 39
7.2 0.1 (2024-06-10) . . . 41
7.3 0.0.17 – 0.0.22 (2024-04 – 2024-05) . . . 41

8 Indices and tables **43**

Python Module Index **45**

Index **47**

`epidemik` is a Python package for building, exploring and visualising compartmental epidemic models. It supports three flavours of model:

- *`EpiModel`* – a classical homogeneously mixed compartmental model that can be either integrated as a system of ODEs or simulated stochastically.
- *`NetworkEpiModel`* – a stochastic compartmental model that lives on top of a contact network supplied as a `networkx.Graph`.
- *`MetaEpiModel`* – a metapopulation model that couples several *`EpiModel`* instances together through a travel matrix.

All three classes share the same fluent API for declaring compartments and transitions, so moving from a homogeneous mixing model to a network or metapopulation variant is mostly a matter of swapping the class.

```
from epidemik import EpiModel

SIR = EpiModel()
SIR.add_interaction("S", "I", "I", 0.2)
SIR.add_spontaneous("I", "R", 0.1)
SIR.integrate(365, S=9_990, I=10, R=0)
SIR.plot()
```

CHAPTER

ONE

INSTALLATION

`epidemik` is published on PyPI and supports Python 3.8 and above. The latest release requires NumPy 1.20+, SciPy 1.10+, pandas 2.0+ and NetworkX 3+.

1.1 From PyPI

The recommended way to install the package is via **`pip`**:

```
pip install epidemik
```

1.2 From source

The repository is managed with uv. To set up a development environment with the package installed in editable mode and the test tools pinned in `uv.lock`:

```
git clone https://github.com/DataForScience/epidemik.git
cd epidemik
uv sync                 # add --group docs to also install Sphinx
uv run pytest           # run the test suite
uv build                # build the wheel and sdist
```

A plain `pip install -e .` also works if you prefer not to use uv.

1.3 Dependencies

`epidemik` relies on a handful of well-known scientific Python libraries:

- NumPy for array maths and random number generation.
- SciPy for ODE integration.
- NetworkX as the underlying graph representation of the compartmental structure and contact networks.
- pandas for tabular outputs.
- Matplotlib for plotting helpers.
- tqdm for progress bars in *`MetaEpiModel`* simulations.

These dependencies are installed automatically by **`pip`**. For nicer *`draw_model()`* output, you may additionally install pygraphviz – when present, it is used to lay out the compartment graph; otherwise a spectral fallback is used.

CHAPTER

TWO

QUICK START

This page walks through the canonical SIR example using *EpiModel*. By the end of it you will have built a model, inspected its structure, computed R_0, and explored its dynamics both deterministically and stochastically.

2.1 Building an SIR model

We start by creating an empty *EpiModel* and adding the two transitions that define the SIR dynamics. The interaction `S + I -> I + I` describes infections at rate β, and the spontaneous transition `I -> R` describes recoveries at rate μ.

```
from epidemik import EpiModel

beta = 0.2
mu = 0.1

SIR = EpiModel(seed=42)
SIR.add_interaction("S", "I", "I", beta=beta)
SIR.add_spontaneous("I", "R", mu=mu)
```

Rates are stored as named parameters (`beta` and `mu` here; a plain number would create anonymous `rate1`, `rate2`, … parameters) and can be expressions of one another, e.g. `mu="beta/2"`. A printable, YAML-formatted description of the model is available via `print`:

```
>>> print(SIR)
# Epidemic Model with 3 compartments and 2 transitions:

Compartments: [S, I, R]

Parameters:
  beta: 0.2
  mu: 0.1

Transitions:
  - S + I = I beta
  - I -> R mu

# R0=2.00
```

The same text is what *save_model()* writes and *load_model()* reads; ready-made models can be fetched with *download_model()*.

The corresponding compartment graph can be drawn with *draw_model()*:

```
SIR.draw_model()
```

The basic reproduction number R_0 is computed from the next-generation matrix:

```
SIR.R0()  # -> 2.0
```

2.2 Deterministic dynamics

To follow the mean-field evolution of the model, call *integrate()*. The first argument is the number of time steps, and the remaining keyword arguments give the initial size of each compartment.

```
N = 10_000
I0 = 10
SIR.integrate(365, S=N - I0, I=I0, R=0)
```

The full trajectory is available in *values_* as a pandas.DataFrame with one row per time step (the first row holds the initial conditions), and individual compartments can be obtained either as attributes or by indexing:

```
SIR.values_.head()
SIR.I                # equivalent to SIR.values_["I"]
SIR[["S", "R"]]      # multiple compartments at once
```

A convenience plotting helper is provided as well:

```
SIR.plot(title="Deterministic SIR")
```

2.3 Stochastic dynamics

To sample one realization of the corresponding stochastic process, use *simulate()* with the same signature. The output has the same shape and time index as the one produced by *integrate()*, so the two can be compared directly:

```
SIR.simulate(365, S=N - I0, I=I0, R=0)
SIR.plot(title="Stochastic SIR")
```

You can also advance a stochastic simulation one step at a time using *single_step()*: the first call takes the initial conditions and every later call appends one more row to *values_*. This is the building block used internally by *MetaEpiModel* to couple sub-populations together.

2.4 Where to go next

- *EpiModel: homogeneously mixed models* documents the full *EpiModel* API, including named parameters, demographics, vaccination, age structure, seasonality, multi-group (vector-borne) and within-host models, and saving/loading model files.
- *NetworkEpiModel: epidemics on a contact network* describes how to run the same kind of model on top of an arbitrary contact network.
- *MetaEpiModel: metapopulation models* shows how to spread an epidemic across many coupled sub-populations.
- *API reference* is the auto-generated API reference for the whole package.

CHAPTER

THREE

EPIMODEL: HOMOGENEOUSLY MIXED MODELS

EpiModel is the workhorse of the `epidemik` package. This page documents every feature it currently supports, with short examples for each.

3.1 Defining the dynamics

A model is built up from two basic types of transition:

- **Interaction** – `source + agent -> target + agent` at a given rate. This is the canonical mass-action contagion term: its contribution to the dynamics is `rate * source * agent / N`.
- **Spontaneous** – `source -> target` at a given rate, independent of any other compartment (recoveries, latency, waning immunity, …).

```
from epidemik import EpiModel

SEIR = EpiModel()
SEIR.add_interaction("S", "E", "I", 0.3)   # S + I -> E + I
SEIR.add_spontaneous("E", "I", 0.2)         # E -> I
SEIR.add_spontaneous("I", "R", 0.1)         # I -> R
```

You can declare compartments explicitly when constructing the model, but it is also fine to rely on the implicit creation triggered by the calls above (agents are created too, so an environmental reservoir that only ever appears as an agent is a valid compartment). Several transitions may share the same source and target (for example two interactions driven by different agents); their contributions are added together.

`print(model)` gives a YAML summary of the parameters, every transition, any birth/death rates and groups, and R_0; *draw_model()* draws the compartment graph.

3.2 Named parameters

Every rate is stored as a named parameter in *params*, an instance of *Parameters*. A rate can be given in three ways:

- as a plain number, in which case an anonymous parameter `rate1`, `rate2`, … is created;
- as a single keyword argument, `beta=0.3`, which names the parameter;
- as a string: the name of an existing parameter, or an *expression* in terms of other parameters such as `"beta/2"` (the functions `exp`, `log`, `sqrt`, `sin`, `cos`, `abs`, `min`, `max` and the constants `pi` and `e` are available).

```
SIR = EpiModel()
SIR.define_parameters(beta=0.3)
SIR.add_interaction("S", "I", "I", "beta")
```

(continues on next page)

(continued from previous page)

```
SIR.add_spontaneous("I", "R", mu="beta/3")   # mu is tied to beta

SIR.params["mu"]         # -> 0.1 (evaluated)
SIR.params.raw("mu")     # -> "beta/3"
SIR.define_parameters(beta=0.6)
SIR.R0()                 # -> 3.0, mu followed beta
```

Expressions are evaluated when the model is run (or when R_0 is computed); an expression that refers to an unknown name raises `KeyError` at that point, and one that cannot be evaluated raises `ValueError`.

3.3 Computing R_0

The basic reproduction number is computed using the next-generation matrix approach, so it works for any model topology – SIR, SEIR, SIS, SIRS, models with several infectious compartments, and so on:

```
SEIR.R0()
```

Vaccination transitions and demographics are ignored in the computation, and every susceptible class is assumed to hold the whole population at the disease-free equilibrium. Susceptible classes are the sources of interactions that are not infected themselves and are only fed by infected or post-infection compartments – so `S` in an SIRS model qualifies, while a class `V` fed by vaccinating `S` does not.

If the model has no infectious dynamics (or R_0 cannot be derived for some other reason), *`R0()`* returns `None`. Multi-group (vector-borne) models and within-host models (unnormalized interactions or viral generation) are not supported and also return `None`.

3.4 Running the dynamics

There are two ways to explore the dynamics of an *`EpiModel`*:

- *`integrate()`* solves the corresponding ODE system numerically using `scipy.integrate.odeint()`.
- *`simulate()`* runs a discrete-time stochastic simulation: at every step, the individuals in each compartment are redistributed with a multinomial draw whose probabilities are the transition rates.

Both methods take the number of time steps as a positional argument and the initial size of each compartment as keyword arguments:

```
SEIR.integrate(365, S=9_990, I=10, R=0)
SEIR.simulate(365, S=9_990, I=10, R=0)
```

All random numbers used by *`simulate()`* come from the model's *`rng`*, a `numpy.random.Generator` created from the `seed` argument of the constructor (or passed in directly as `rng`), so simulations are reproducible:

```
SEIR = EpiModel(seed=42)
```

Results are stored as a `pandas.DataFrame` in *`values_`*, indexed by time. Both methods produce exactly `timesteps` rows, the first of which (at `t_min`, which defaults to 1) holds the initial conditions, so deterministic and stochastic trajectories can be compared directly. Compartments that were not mentioned start at zero; initial conditions for names that are not compartments of the model are ignored with a warning.

Because *`simulate()`* uses the rates as per-step transition probabilities, they should be small compared to 1. If the probabilities out of some compartment add up to more than 1, they are renormalized and a warning is emitted.

3.5 Stepping manually

single_step() advances a stochastic simulation one step at a time. The first call takes the initial conditions; later calls continue from the last row of *values_* unless new compartment sizes are given, in which case they replace the current state:

```
SEIR.single_step(S=9_990, I=10, R=0)   # rows at t=1 (initial) and t=2
SEIR.single_step()                     # row at t=3
SEIR.single_step(S=9_000, I=1_000)     # override the state, then step
```

This is the building block *MetaEpiModel* uses to couple sub-populations together.

3.6 Vaccination

To model a time-gated vaccination campaign, use *add_vaccination()*. The transition behaves like a spontaneous transition that only becomes active for `time > start`:

```
# Vaccinate 1% of susceptibles per day, starting on day 60
SEIR.add_vaccination("S", "V", rate=0.01, start=60)
SEIR.integrate(365, S=9_990, I=10, R=0, V=0)
```

3.7 Demographics: births and deaths

Birth and death rates can be attached to one or all compartments. Enabling either puts the model in *demographics mode*:

```
SEIR.add_birth_rate(0.0001, comps=["S"])   # newborns are susceptible
SEIR.add_death_rate(0.0001)               # applied to every compartment
```

By default births into a compartment are proportional to the **total** population (`rate * N`, the classical setup where every newborn is susceptible) and deaths are proportional to the compartment itself (`rate * X`). Both methods accept further modes:

- `fixed=True` – a constant number of individuals per unit time (e.g. a constant supply of target cells);
- `global_rate=True/False` – proportional to the total population or to the compartment itself;
- `from_group=True` (births only) – proportional to the population of the group containing the compartment (see *Multi-group and vector-borne models* below).

Both ODE integration and stochastic simulation are aware of demographics: in *integrate()* births and deaths enter the equations as flow terms, while in *simulate()* births are sampled from a binomial (or Poisson) distribution and deaths compete with the other transitions out of each compartment in the same multinomial draw, so compartments never become negative.

For backwards compatibility the list of compartments may also be given first together with a named rate: `add_birth_rate("S", b=0.01)`.

3.8 Multi-group and vector-borne models

By default every interaction is normalized by the total population `N`. For models with several interacting populations – hosts and vectors, for instance – the force of infection acting on the hosts should instead be normalized by the number of *vectors*, and vice versa. *add_groups()* declares such a partition of the compartments: after the call, the interaction `X + A -> Y + A` contributes `rate * X * A / N_group(A)`, where `N_group(A)` is the total population of the group containing the agent `A`.

```
bh, bv, gh, gv, mh, mv = 0.3, 0.3, 0.1, 0.1, 0.01, 0.1

vector_model = EpiModel()
vector_model.add_interaction("Sh", "Ih", "Iv", bh)   # Sh + Iv -> Ih + Iv
vector_model.add_interaction("Sv", "Iv", "Ih", bv)   # Sv + Ih -> Iv + Ih
vector_model.add_spontaneous("Ih", "Rh", gh)

vector_model.add_groups({
    "host":   ["Sh", "Ih", "Rh"],
    "vector": ["Sv", "Iv"],
})

# hosts die at rate mh, vectors die at rate gv, and mv * N_vector new
# susceptible vectors are born per unit time
vector_model.add_death_rate(mh, comps=["Sh", "Ih", "Rh"])
vector_model.add_death_rate(gv, comps=["Sv", "Iv"])
vector_model.add_birth_rate(mv, comps=["Sv"], from_group=True)

vector_model.integrate(100, Sh=999, Ih=1, Rh=0, Sv=1_000, Iv=10)
```

This integrates the classical host-vector system

$$
\begin{aligned}
\dot{S}_h &= -b_h S_h I_v / N_v - m_h S_h \\
\dot{I}_h &= b_h S_h I_v / N_v - g_h I_h - m_h I_h \\
\dot{R}_h &= g_h I_h - m_h R_h \\
\dot{S}_v &= m_v N_v - b_v S_v I_h / N_h - g_v S_v \\
\dot{I}_v &= b_v S_v I_h / N_h - g_v I_v
\end{aligned}
$$

Every compartment of the model must belong to exactly one group; *integrate()* and *simulate()* raise `ValueError` if a compartment added after *add_groups()* was left unassigned. If a group is empty its force of infection is simply zero. Groups cannot be combined with age structure, and *R0()* returns `None` for multi-group models.

3.9 Within-host models

Two additional building blocks make it possible to write models of infection *within* a host, such as the target-cell-limited model of viral dynamics:

- `norm=False` on *add_interaction()* turns the term into the plain mass-action product `rate * source * agent`, without dividing by the population;
- *add_viral_generation()* encodes the asymmetric reaction `source => target`: infected cells die at one rate while, per infected cell, virions are produced at another.

```
TCL = EpiModel()
TCL.define_parameters(s=1e5, d=0.01, beta=2e-7, delta=0.5, p=100, c=2)
TCL.add_birth_rate("s", comps=["T"], fixed=True)   # constant supply of cells
TCL.add_death_rate("d", comps=["T"])
TCL.add_interaction("T", "I", "V", "beta", norm=False)  # T + V -> I + V
TCL.add_viral_generation("I", "V", "delta", "p")        # I dies, V is produced
TCL.add_death_rate("c", comps=["V"])                    # viral clearance
TCL.integrate(60, T=1e7, I=0, V=10)
```

This integrates $\dot{T} = s - dT - \beta TV$, $\dot{I} = \beta TV - \delta I$ and $\dot{V} = pI - cV$. In *simulate()*, cell death competes with the other transitions and virion production is drawn from a Poisson distribution. *R0()* returns `None` for such models, since the within-host reproduction number depends on the initial number of target cells.

3.10 Age structure

Age structure is layered on top of an existing model by calling *add_age_structure()* with a contact matrix and a vector of per-age-group populations. Every compartment is replicated into one compartment per age group (`S_a`, `S_b`, ...), every interaction rate is rescaled by the corresponding contact matrix entry, and interactions are normalized by the population of the *agent's* age group. Birth and death rates attached before the call are copied to every age stratum.

```
import numpy as np

contact = np.array([
    [10, 4, 1],
    [4, 12, 4],
    [1, 4, 8],
])
population = np.array([1_000, 2_000, 1_500])

age_SEIR = EpiModel()
age_SEIR.add_interaction("S", "E", "I", 0.05)
age_SEIR.add_spontaneous("E", "I", 0.2)
age_SEIR.add_spontaneous("I", "R", 0.1)
age_SEIR.add_age_structure(contact, population)

age_SEIR.integrate(365, S=4_490, I=10, R=0)
```

Initial conditions given with the original compartment names are split across age groups proportionally to `population` (exactly in *integrate()*, with a multinomial draw in *simulate()*). Expanded names such as `S_a=500` can also be passed directly. The rescaled interaction rates become new parameters named `<rate>_<ij>` (e.g. `beta_ab`) defined as expressions of the original ones, so redefining `beta` afterwards still works. After the run, the per-age-group time series are available in *values_ages_* and the totals (across age groups) in *values_*. Individual age strata can be retrieved with `model["S_a"]` or `model[["S_a", "S_b"]]`.

3.11 Seasonality

Both *integrate()* and *simulate()* accept a `seasonality` argument: a array of multiplicative factors (typically one per day of the year) that is applied to every interaction rate. At time `t` the effective rate is multiplied by `seasonality[t % len(seasonality)]`.

```
import numpy as np

t = np.arange(365)
season = 1 + 0.3 * np.cos(2 * np.pi * t / 365)
SEIR.integrate(365, seasonality=season, S=9_990, I=10, R=0)
```

3.12 Accessing and visualising results

After running either *integrate()* or *simulate()*:

- *values_* is the full trajectory as a `pandas.DataFrame` indexed by time.

- `model.X` returns the time series of compartment `X` (it is shorthand for `model.values_["X"]`).
- `model["X"]` and `model[["X", "Y"]]` return the time series of one or several compartments; `None` is returned when the name is unknown.
- *plot()* produces a quick line plot with reasonable defaults (using the colours defined in *epidemik.utils.epi_colors*), optionally on a caller-supplied `ax`.
- *draw_model()* draws the compartmental structure itself.

Using `model["X"]` or *plot()* before running the model raises *NotInitialized*. *reset()* discards the results of a previous run.

3.13 Saving, loading and sharing models

A model definition can be written to a YAML file and read back:

```
SIR.name = "SIR"
SIR.save_model("SIR.yaml")
SIR2 = EpiModel.load_model("SIR.yaml", seed=1)
```

The file is exactly the output of `print(SIR)`:

```
# Epidemic Model with 3 compartments and 2 transitions:
Name: SIR

Compartments: [S, I, R]

Parameters:
  beta: 0.3
  mu: "beta/3"

Transitions:
  - S + I = I beta
  - I -> R mu

# R0=3.00
```

Transitions are written as `S + I = I beta` (interaction, with a trailing `unnormalized` when `norm=False`), `I -> R mu` (spontaneous), `S -> V nu starting at 60 days` (vaccination) and `I => V delta p` (viral generation). Rates may be parameter names or numeric literals. Optional `Demographics` (a list of `{compartment: S, birth: b}` / `{compartment: I, death: d}` mappings with `fixed`, `global` and `from_group` flags) and `Groups` sections round-trip the corresponding features; `Compartments` fixes the order of the compartments and may be omitted.

A small library of ready-made models (SI, SIR, SEIR, SEIRS, SEIIR, SEIIRD) is hosted in the `epidemik` GitHub repository:

```
EpiModel.list_models()                    # -> ["SEIIR.yaml", "SEIIRD.yaml", ...]
SEIR = EpiModel.download_model("SEIR.yaml")
```

Downloaded files are cached in a per-user directory (see *epidemik.utils.get_cache_directory()*; override it with the `EPIDEMIK_CACHE_DIR` environment variable). Both functions accept a `repo` argument that can be another GitHub repository or a local directory of model files.

CHAPTER

FOUR

NETWORKEPIMODEL: EPIDEMICS ON A CONTACT NETWORK

NetworkEpiModel lifts an *EpiModel* onto an arbitrary contact network supplied as a `networkx.Graph`. Every individual is represented by a single network node and can only infect its direct neighbours. Only stochastic simulation is supported – the corresponding ODE system has no closed-form analogue on a generic graph.

4.1 Building a network model

The constructor takes the contact network as its first argument; the disease transitions are then added in exactly the same way as for an *EpiModel*. Node labels can be of any hashable type (integers, strings, tuples, ...):

```
import networkx as nx
from epidemik import NetworkEpiModel

G = nx.erdos_renyi_graph(n=1_000, p=0.01, seed=42)

SIR = NetworkEpiModel(G, seed=42)
SIR.add_interaction("S", "I", "I", beta=0.05)
SIR.add_spontaneous("I", "R", mu=0.1)
```

Rates are named parameters exactly as in *EpiModel* (numbers, keywords or expressions). On a network, a rate is the probability that a single contact transmits the disease during one time step (for interactions) or that an individual undergoes the transition during one time step (for spontaneous transitions). If you want to keep R_0 comparable to a mean-field model defined with the same rates, you can ask *add_interaction()* to rescale the rate by the average degree of the network; the rescaled value is stored as a new parameter `<name>_rescaled`:

```
SIR.add_interaction("S", "I", "I", beta=0.05, rescale=True)
```

Vaccination campaigns are supported through *add_vaccination()* and behave like time-gated spontaneous transitions.

4.2 Running a simulation

Instead of supplying initial compartment sizes, simulations are seeded by passing a `{node_label: compartment_name}` dictionary. All non-seeded nodes start in the susceptible compartment, which is detected automatically (the compartment without incoming transitions) or can be given explicitly with `susceptible=`:

```
seeds = {0: "I", 1: "I", 2: "I"}
SIR.simulate(timesteps=60, seeds=seeds)
```

After the run:

- `SIR.population_` is a `pandas.DataFrame` of shape `(timesteps, n_nodes)` containing the compartment of every node at every time step. Columns are labelled by node, rows by time.
- *`values_`* contains the aggregate compartment counts across the network over time, with one column per compartment of the model (compartments that are never reached contain zeros).
- *`plot()`* produces the usual aggregate plot.

As for *`EpiModel`*, the first row of both data frames (at `t_min`, which defaults to 1) holds the initial conditions, and all random numbers come from the model's `rng` (see the `seed` argument of the constructor). Seeding a node that is not part of the network, or with a state that is not a compartment, raises `ValueError`. If the transition probabilities out of a compartment (or from one contact) add up to more than 1 they are renormalized and a warning is emitted. When several interactions share the same source and agent, one of the targets is chosen at random with probabilities proportional to the rates.

4.3 R_0 on a network

The *`R0()`* method returns the network-corrected basic reproduction number. The mean-field R_0 of the underlying disease is multiplied by the average degree of the contact network (stored in `kavg_`) to account for the limited number of contacts each individual has, and rounded to two decimals.

```
SIR.R0()
```

The method returns `None` whenever the mean-field R_0 cannot be computed (see *`EpiModel.R0`*).

4.4 What is *not* supported

Some features available to *`EpiModel`* are intentionally omitted on networks:

- *`integrate()`* raises `NotImplementedError` – explore the dynamics with *`simulate()`* instead.
- Births, deaths, viral generation, groups and age structure do not make sense when every node is one individual of a fixed population; the corresponding `add_*` methods raise `NotImplementedError`.
- Seasonality is not taken into account by *`simulate()`*.

CHAPTER

FIVE

METAEPIMODEL: METAPOPULATION MODELS

MetaEpiModel simulates an epidemic spreading across several coupled sub-populations – think US states, cities or districts. Each sub-population is driven by its own *EpiModel*, and at every time step individuals are exchanged between sub-populations according to a travel matrix.

5.1 Building a metapopulation model

The constructor takes two `pandas.DataFrame` arguments:

- `travel_graph` – a square data frame whose rows and columns are labelled by sub-population identifiers. Entry `(i, j)` is the probability that an individual currently in sub-population `i` travels to sub-population `j` during a single time step. Each row must sum to 1.
- `populations` – a data frame indexed by sub-population identifier containing at least one column with the total population of each sub-population (named `"Population"` by default; use the `population` argument to pick another column).

```
import pandas as pd
from epidemik import MetaEpiModel

travel = pd.DataFrame(
    {"A": [0.99, 0.10], "B": [0.01, 0.90]}, index=["A", "B"]
)
populations = pd.DataFrame(
    {"Population": [100_000, 10_000]}, index=["A", "B"]
)

SIR = MetaEpiModel(travel, populations, seed=42)
SIR.add_interaction("S", "I", "I", beta=0.3)
SIR.add_spontaneous("I", "R", mu=0.1)
```

The constructor validates its inputs and raises `ValueError` if the travel matrix is not square, if its rows do not sum to 1, or if a sub-population is missing from `populations`. The optional `seed` (or `rng`) argument makes the whole simulation – travel and every sub-population – reproducible. Rates are named parameters exactly as in *EpiModel*; `SIR.params` and *define_parameters()* give access to them.

Every transition added at the metapopulation level – *add_interaction()*, *add_spontaneous()*, *add_vaccination()*, *add_viral_generation()*, *add_birth_rate()*, *add_death_rate()* and *add_groups()* – is propagated to every sub-population, so the disease is assumed to be the same everywhere.

5.2 Seeding and simulating

The initial epidemic seeds are placed in a designated *seed sub-population* via the `seed_state` argument of *`simulate()`*. The remaining keyword arguments give the initial size of each non-susceptible compartment in that sub-population; the susceptible compartment is adjusted automatically so that the total population of the seed sub-population is preserved.

```
SIR.simulate(timesteps=60, seed_state="A", I=10)
```

Internally, every time step alternates between a *spread* phase (every sub-population's *`EpiModel`* advances by one stochastic step via *`single_step()`*) and a *travel* phase (individuals are redistributed across sub-populations according to `travel_graph`). A `seasonality` array, if given, is forwarded to every sub-population.

If `seed_state` is not provided, a *`NotInitialized`* exception is raised; seeds larger than the population of `seed_state` raise `ValueError`. Every call starts a fresh simulation in each sub-population. The first positional argument used to be called `timestamp`; that spelling still works but emits a `DeprecationWarning`.

5.3 Inspecting the result

After a simulation:

- `SIR.compartments_` holds the current compartment counts per sub-population (one row per sub-population, one column per compartment).
- `SIR.get_state("A")` returns the *`EpiModel`* that drove sub-population `"A"`, so you can inspect or plot its trajectory using the usual API. Its *`values_`* has `timesteps + 1` rows: the initial conditions followed by the state after each step.

  ```
  SIR.get_state("A").plot()
  ```

- *`plot()`* lays out one mini-panel per sub-population showing the trajectory of a compartment (`"I"` by default), marks the time of the local peak with a vertical line and colours each panel by how many steps after the start that peak occurs.
- *`plot_peaks()`* produces a heat-map of the same normalized trajectories, which highlights how the epidemic peak diffuses across sub-populations.

Calling either plotting method before *`simulate()`* raises *`NotInitialized`*.

5.4 What is *not* supported

- *`integrate()`* raises `NotImplementedError`: the metapopulation dynamics only make sense as a stochastic process.
- Age structure is not available at the metapopulation level.
- With *`add_groups()`*, `populations` only seeds the `susceptible` compartment of every sub-population. The initial size of the other groups (e.g. the vectors) can only be set in `seed_state` through the keyword arguments of *`simulate()`*.

CHAPTER

SIX

API REFERENCE

This section documents the public API of every module in the *epidemik* package. The content is generated automatically from the docstrings in the source code.

6.1 epidemik

epidemik is a Python package for the simulation of compartmental epidemic models.

The package exposes three main model classes:

- *EpiModel* – compartmental model in a homogeneously mixed population, supporting numerical integration and stochastic simulation.
- *NetworkEpiModel* – compartmental model on top of a contact network represented by a `networkx.Graph`.
- *MetaEpiModel* – metapopulation model where several sub-populations are coupled together by a travel matrix.

The *epidemik.utils* submodule provides a small number of shared helpers, including the *NotInitialized* exception.

See https://www.github.com/DataForScience/epidemik for the source code and https://epidemik.readthedocs.io/ for the full documentation.

6.2 Package overview

EpiModel	Compartmental epidemic model on a homogeneously mixed population.
NetworkEpiModel	Compartmental epidemic model that lives on a contact network.
MetaEpiModel	Metapopulation compartmental epidemic model.
utils	Shared utilities used across the *epidemik* package.

6.3 EpiModel

class epidemik.**EpiModel**(*compartments=None*, *seed=None*, *rng=None*, *name=None*)

Bases: `object`

Compartmental epidemic model on a homogeneously mixed population.

The model is internally represented as a directed multigraph (`networkx.MultiDiGraph`) whose nodes are compartments and whose edges are transitions. Each transition can be either *spontaneous* (`X -> Y` at some rate) or an *interaction* mediated by an agent compartment (`X + A -> Y + A` at some rate). Once defined, the model

dynamics can be explored deterministically with *integrate()* (which numerically solves the corresponding ODEs) or stochastically with *simulate()*.

Every rate is stored as a named parameter in *params*. Rates can be given as plain numbers (an anonymous parameter `rate1`, `rate2`, … is created), as a named keyword argument (`beta=0.3`), or as an expression in terms of other parameters (`mu="beta/2"`).

name

Optional name of the model, stored by *save_model()*.

Type
str or None

transitions

Graph of compartments (nodes) and transitions (edges).

Type
networkx.MultiDiGraph

params

Named parameters of the model.

Type
epidemik.utils.Parameters

rng

Random number generator used by *simulate()*.

Type
numpy.random.Generator

values_

Trajectory of every compartment, indexed by time. Only available after *integrate()* or *simulate()* has been called.

Type
pandas.DataFrame

values_ages_

Age-stratified trajectory (one column per `compartment_age` pair). Only available after *add_age_structure()* and a call to *integrate()* or *simulate()*.

Type
pandas.DataFrame

__init__(*compartments=None*, *seed=None*, *rng=None*, *name=None*)

Initialize an empty *EpiModel* instance.

Parameters

- **compartments** (*list of str, optional*) – Optional list of compartment names to seed the model with. New compartments are also added automatically the first time they appear in a call to *add_interaction()*, *add_spontaneous()* or *add_vaccination()*.
- **seed** (*int, optional*) – Seed for the random number generator used by *simulate()*. Ignored when `rng` is given.
- **rng** (*numpy.random.Generator, optional*) – Random number generator to use. If `None` a new generator is created with `numpy.random.default_rng()`.
- **name** (*str, optional*) – Human-readable name of the model.

define_parameters(***kwargs*) → None

Define (or redefine) named parameters.

Parameters

****kwargs** – `name=value` pairs where `value` is a number or an expression string in terms of other parameters (`"beta/2"`).

Examples

```
>>> model = EpiModel()
>>> model.define_parameters(beta=0.3, mu="beta/3")
>>> model.add_interaction("S", "I", "I", "beta")
>>> model.add_spontaneous("I", "R", "mu")
```

add_interaction(
source: str,
target: str,
agent: str,
rate=None,
norm: bool = True,
***rates*,
) → None

Add an interaction transition between two compartments.

Encodes a reaction of the form `source + agent -> target + agent`. In the mean-field equations the contribution of this transition is `rate * source * agent / N` where `N` is the total population (or the population of the agent's group/age stratum – see *add_groups()* and *add_age_structure()*). With `norm=False` the term is the plain mass-action product `rate * source * agent`, which is the natural choice for within-host models (cells and virions).

Parameters

- **source** (*str*) – Name of the source compartment.
- **target** (*str*) – Name of the target compartment.
- **agent** (*str*) – Name of the agent compartment driving the interaction.
- **rate** (*float or str, optional*) – Per-capita rate, given positionally: a number, the name of an existing parameter, or an expression string.
- **norm** (*bool, default=True*) – Whether to normalize the interaction by the population size.
- ****rates** – Alternatively, the rate as a single named parameter, e.g. `beta=0.3`.

add_spontaneous(*source: str*, *target: str*, *rate=None*, ***rates*) → None

Add a spontaneous transition between two compartments.

Encodes a reaction of the form `source -> target` that occurs at the given rate, independently of any other compartment.

Parameters

- **source** (*str*) – Name of the source compartment.
- **target** (*str*) – Name of the target compartment.
- **rate** (*float or str, optional*) – Per-capita rate, given positionally (number, parameter name or expression).

- ****rates** – Alternatively, the rate as a single named parameter, e.g. `mu=0.1`.

add_viral_generation(
source: str,
target: str,
source_rate=None,
target_rate=None,
***rates*,
) → None

Add an asymmetric production transition (e.g. virions from infected cells).

Encodes `source => target`: individuals in `source` are removed at rate `source_rate` (cell death) while, independently, new individuals appear in `target` at rate `target_rate` per `source` individual (viral production). Unlike a spontaneous transition the two flows are not equal.

Parameters

- **source** (*str*) – Producing compartment (e.g. infected cells).
- **target** (*str*) – Produced compartment (e.g. free virus).
- **source_rate** (*float or str, optional*) – Removal rate of `source` individuals.
- **target_rate** (*float or str, optional*) – Production rate of `target` individuals per `source` individual.
- ****rates** – Alternatively, exactly two named parameters, the first being the source rate and the second the target rate, e.g. `delta=0.5, p=100`.

add_birth_rate(
rate=None,
comps: List | None = None,
fixed: bool = False,
global_rate: bool = True,
from_group: bool = False,
***rates*,
) → None

Attach a birth rate to one or more compartments.

Enabling births (or deaths) switches the model into demographics mode: inflows are added to the chosen compartments during both *integrate()* and *simulate()*. The inflow into each listed compartment `X` is, per unit time:

- `rate * N` with `global_rate=True` (the default), where `N` is the total population – the classical setup where every newborn enters the susceptible compartment;
- `rate * X` with `global_rate=False`;
- `rate * N_group(X)` with `from_group=True`, where `N_group(X)` is the population of the group containing `X` (see *add_groups()*) – the canonical setup for vector-borne models where all newborn vectors are susceptible;
- exactly `rate` individuals with `fixed=True`.

Parameters

- **rate** (*float or str, optional*) – Birth rate (number, parameter name or expression). For backwards compatibility, a list of compartments may also be given here together with a named rate (`add_birth_rate(["S"], b=0.1)`).
- **comps** (*list, optional*) – Compartments to which to assign this birth rate. If `None` (the default) the rate is applied to every compartment.

- **fixed** (*bool, default=False*) – If True, `rate` is a constant number of births per unit time.
- **global_rate** (*bool, default=True*) – If True, births are proportional to the total population.
- **from_group** (*bool, default=False*) – If True, births are proportional to the group total. Requires *add_groups()*.
- ****rates** – Alternatively, the rate as a single named parameter, e.g. `b=0.01`.

Raises

ValueError – If `from_group=True` but *add_groups()* has not been called yet.

Notes

When combined with *add_age_structure()*, birth and death rates attached *before* the age structure is added are copied to every age stratum of the compartment. Rates added *after* must use the expanded compartment names (`"S_a"`, `"S_b"`, …).

add_death_rate(
rate=None,
comps: List | None = None,
fixed: bool = False,
global_rate: bool = False,
***rates*,
) → None

Attach a death rate to one or more compartments.

The outflow from each listed compartment `X` is, per unit time, `rate * X` by default, `rate * N` (with `N` the total population) with `global_rate=True`, or exactly `rate` individuals with `fixed=True`.

Parameters

- **rate** (*float or str, optional*) – Death rate (number, parameter name or expression). A list of compartments may also be given here together with a named rate.
- **comps** (*list, optional*) – Compartments to which to assign this death rate. If `None` (the default) the rate is applied to every compartment.
- **fixed** (*bool, default=False*) – If True, `rate` is a constant number of deaths per unit time.
- **global_rate** (*bool, default=False*) – If True, deaths are proportional to the total population.
- ****rates** – Alternatively, the rate as a single named parameter, e.g. `d=0.01`.

add_vaccination(
source: str,
target: str,
rate=None,
start: int | None = None,
***rates*,
) → None

Add a time-gated vaccination transition between two compartments.

Identical to *add_spontaneous()* except that the transition is only active for `time > start`.

Parameters

- **source** (*str*) – Name of the source compartment (e.g. `"S"`).

- **target** (*str*) – Name of the target compartment (e.g. "V").
- **rate** (*float or str, optional*) – Per-capita vaccination rate (number, parameter name or expression).
- **start** (*int*) – Time step at which the vaccination campaign begins.
- ****rates** – Alternatively, the rate as a single named parameter, e.g. nu=0.01.

add_age_structure(*matrix: List*, *population: List*) → None

Expand the model into an age-structured version.

Every compartment X is replicated into one compartment per age group X_a, X_b, … and the interactions are rescaled by the provided contact matrix. The population argument provides the absolute size of each age group, which is used by *integrate()* and *simulate()* to split the initial population across age strata. Any birth or death rates already attached to a compartment are copied to every one of its age strata. The rescaled interaction rates are stored as new parameters named <rate>_<i><j> defined in terms of the original ones, so the original parameters can still be changed afterwards.

Parameters

- **matrix** (*list of list of float*) – Square contact matrix. Entry matrix[i][j] is the average number of contacts a member of age group i has with members of age group j.
- **population** (*list of int*) – Number of individuals in each age group, in the same order as the rows of matrix.

Raises

ValueError – If the model is already in multi-group (vector-borne) mode via *add_groups()*, if matrix is not square, or if population does not have one entry per age group.

add_groups(*groups: Dict[str, List[str]]*) → None

Partition compartments into named sub-populations (groups).

When groups are declared the model becomes a *multi-group* (e.g. vector-borne) model: cross-group interactions normalize the force of infection by the **agent's** group total rather than by the global total population. Concretely, an interaction

$$X + A \to Y + A$$

contributes

$$\text{rate} \cdot X \cdot \frac{A}{N_{\text{group}(A)}}$$

to the dynamics, where $N_{\text{group}(A)}$ is the total population of the group that contains the agent compartment A. This recovers the standard host-vector SIR dynamics where, for example, susceptible hosts become infected at rate $b_h S_h I_v / N_v$ and susceptible vectors at rate $b_v S_v I_h / N_h$.

Parameters

groups (*dict of str -> list of str*) – Mapping from group name (e.g. "host", "vector") to the list of compartment names that belong to that group. Every compartment present in the model must appear in exactly one group. Compartments added *after* this call must also belong to a group; *integrate()* and *simulate()* check this and raise ValueError otherwise.

Raises

ValueError – If the model is already in age-structured mode (see *add_age_structure()*), if a compartment is listed in more than one group, or if any compartment in the model is left unassigned.

Examples

```
>>> model = EpiModel()
>>> model.add_interaction("Sh", "Ih", "Iv", 0.3)
>>> model.add_interaction("Sv", "Iv", "Ih", 0.3)
>>> model.add_spontaneous("Ih", "Rh", 0.1)
>>> model.add_groups({
...     "host":   ["Sh", "Ih", "Rh"],
...     "vector": ["Sv", "Iv"],
... })
```

plot(
title: str | None = None,
normed: bool = True,
show: bool = True,
ax: Axes | None = None,
***kwargs*,
)

Plot the time evolution of every compartment.

Must be called after *integrate()* or *simulate()*.

Parameters

- **title** (*str, optional*) – Title to set on the resulting axes.
- **normed** (*bool, default=True*) – If True, divide every series by the total initial population so that values represent fractions instead of absolute counts.
- **show** (*bool, default=True*) – If True, call `matplotlib.pyplot.show()` after plotting.
- **ax** (*matplotlib.axes.Axes, optional*) – Axes to draw onto. If `None`, the current axes are used.
- ****kwargs** – Extra keyword arguments forwarded to `pandas.Series.plot()`.

Returns
The axes that were drawn on.

Return type
matplotlib.axes.Axes

Raises
NotInitialized – If neither *integrate()* nor *simulate()* has been called yet.

__getattr__(*name: str*) → Series

Return the time series of a compartment as an attribute.

After *integrate()* or *simulate()*, the time series of any compartment `X` can be accessed via `model.X`.

Parameters
name (*str*) – Name of the compartment.

Returns
Time series of the requested compartment.

Return type
pandas.Series

Raises
AttributeError – If the model has not been run yet, or if `name` is not a compartment of the model.

`__getitem__`(*key*)

Return one or several compartment trajectories by name.

Parameters
key (*str or list of str*) – Name(s) of the compartment(s) of interest. When the model has age structure, age-stratified compartments such as `"S_a"` are also accepted.

Returns
Time series of the requested compartment(s), or `None` if no matching compartment can be found.

Return type
pandas.Series or pandas.DataFrame or None

Raises
NotInitialized – If neither *integrate()* nor *simulate()* has been called yet.

simulate(
timesteps: int,
t_min: int = 1,
seasonality: ndarray | None = None,
***kwargs*,
) → None

Run a discrete-time stochastic simulation of the model.

At every time step the individuals of each compartment are redistributed with a multinomial draw whose probabilities are the outgoing transition rates (deaths included); births are drawn from a binomial (or Poisson, for fixed rates) distribution and viral production from a Poisson distribution. Results are stored in the *values_* attribute as a `pandas.DataFrame` indexed by time: the first row (`t_min`) holds the initial conditions and the remaining `timesteps - 1` rows hold the state after each step, mirroring *integrate()*. All random numbers come from *rng*.

Parameters

- **timesteps** (*int*) – Number of time steps (rows of *values_*) to produce, including the initial condition.
- **t_min** (*int, default=1*) – Starting time (used to align the index of *values_* and to evaluate any time-gated transitions, e.g. vaccination).
- **seasonality** (*numpy.ndarray, optional*) – Array of seasonal forcing factors, typically of length 365. Interaction rates at time `t` are multiplied by `seasonality[t % len(seasonality)]`.
- ****kwargs** – Initial size of each compartment, passed as keyword arguments (e.g. `S=9990, I=10, R=0`). Unknown compartment names are ignored with a warning. For age-structured models, values are split across age strata with a multinomial draw.

Warns
UserWarning – If, for some compartment, the sum of the outgoing transition probabilities exceeds 1 at some time step. The probabilities are renormalized so that every individual leaves the compartment; consider using smaller rates (i.e. a finer time step).

integrate(
timesteps: int,
t_min: int = 1,

seasonality: ndarray | None = None,
***kwargs*,
) → None

Numerically integrate the model's ODE system.

Uses `scipy.integrate.odeint()` to advance the deterministic mean-field equations of the model. Results are stored in *values_* (the first row, at `t_min`, holds the initial conditions); when an age structure has been declared via *add_age_structure()*, the per-age-group trajectories are also stored in *values_ages_*.

Parameters

- **timesteps** (*int*) – Number of time steps to integrate over.
- **t_min** (*int, default=1*) – Starting time of the integration.
- **seasonality** (*numpy.ndarray, optional*) – Array of seasonal forcing factors, typically of length 365. See *simulate()* for details.
- ****kwargs** – Initial size of each compartment, passed as keyword arguments (e.g. `S=9990, I=10, R=0`). Unknown compartment names are ignored with a warning. For age-structured models, values are split across age strata proportionally to the age group sizes.

single_step(*seasonality=None*, *t_min: int = 1*, ***kwargs*)

Advance the stochastic simulation by a single time step.

On the first call the initial conditions given as keyword arguments are stored at `t_min` and one stochastic step is taken. On subsequent calls the simulation is continued from the last row of *values_*, unless new compartment sizes are provided as keyword arguments, in which case they replace the current state before the step is taken – this is the building block used by *MetaEpiModel* to couple multiple sub-populations together.

Parameters

- **seasonality** (*numpy.ndarray, optional*) – Array of seasonal forcing factors, typically of length 365.
- **t_min** (*int, default=1*) – Starting time, used on the first call only.
- ****kwargs** – Compartment populations. Required on the first call; optional afterwards.

reset() → None

Discard the results of any previous *integrate()* or *simulate()* call.

__repr__() → str

Return a human-readable, YAML-compatible description of the model.

The text lists the parameters, every transition (interaction, spontaneous, vaccination or viral generation), the demographic rates and groups, and – as a comment – the basic reproduction number R_0. It is exactly what *save_model()* writes and what *load_model()* reads back.

Returns
Multi-line description of the model.

Return type
str

draw_model(
ax: Axes | None = None,
show: bool = True,
) → None

Render the compartmental structure of the model as a graph.

Each compartment is drawn as a node and each transition as a directed edge labelled by the agent compartment (for interactions) or left unlabelled (for spontaneous transitions). When births or deaths have been enabled, extra invisible source/sink nodes are added at the top/bottom of every compartment.

Parameters

- **ax** (*matplotlib.axes.Axes, optional*) – Axes to draw onto. If `None` a new figure is created.
- **show** (*bool, default=True*) – Whether to call `matplotlib.pyplot.show()` after drawing.

R0() → float | None

Compute the basic reproduction number R_0 of the model.

The calculation is fully generic and uses the next-generation matrix approach described in *Diekmann, Heesterbeek and Roberts, J. R. Soc. Interface* **7**, 873 (2010). R_0 is returned as the spectral radius of FV^{-1}, where F collects the rates of new infections in each infected compartment and V collects the rates of all other transitions among infected compartments. Every susceptible class (see `_get_susceptible()`) is assumed to hold the whole population at the disease-free equilibrium. Vaccination transitions and demographics are ignored.

Returns

The basic reproduction number, or `None` if it cannot be computed: when the model has no infectious dynamics defined, when V is singular, when the model has been put in multi-group mode via *add_groups()*, or when it contains unnormalized interactions or viral generation transitions (within-host models), whose R_0 depends on the initial conditions.

Return type

float or None

save_model(*filename: str*) → None

Save the model definition to a YAML file.

The file contains the output of `repr(model)`: the name, parameters, transitions, demographics and groups. It can be read back with *load_model()*.

Parameters

filename (*str*) – Path of the file to write.

static load_model(*filename: str*, *seed=None*, *rng=None*) → *EpiModel*

Load a model definition from a YAML file written by *save_model()*.

The file is a mapping with the (case-insensitive) keys `Name`, `Compartments` (optional list fixing the compartment order), `Parameters` (`name: value` pairs, values being numbers or expressions), `Transitions` (a list of strings such as `S + I = I beta`, `I -> R mu`, `S -> V nu starting at 60 days` or `I => V delta p`), and optionally `Demographics` (a list of `{compartment: S, birth: b}` / `{compartment: I, death: d}` mappings with optional `fixed`, `global` and `from_group` flags) and `Groups` (a mapping from group name to list of compartments).

Parameters

- **filename** (*str*) – Path of the file to read.
- **seed** (*int, optional*) – Seed for the random number generator of the new model.
- **rng** (*numpy.random.Generator, optional*) – Random number generator for the new model.

Returns
The loaded model.

Return type
EpiModel

Raises
ValueError – If a transition cannot be parsed or refers to an undefined parameter.

static list_models(*repo: str | None = None*) → List[str]

List the model files available in a repository.

Parameters
repo (*str, optional*) – GitHub repository URL (defaults to the official `epidemik` repository) or local directory. A GitHub repository must provide a `model_list.txt` index next to the model files.

Returns
Sorted file names, including any `.yaml` files previously saved in the local cache directory (see *epidemik.utils.get_cache_directory()*).

Return type
list of str

static download_model(
filename: str,
repo: str | None = None,
load_model: bool = True,
force: bool = False,
seed=None,
rng=None,
)

Fetch a model file from a repository (caching it locally).

Parameters

- **filename** (*str*) – Name of the model file, e.g. `"SIR.yaml"` (see *list_models()*).
- **repo** (*str, optional*) – GitHub repository URL (defaults to the official `epidemik` repository) or local directory.
- **load_model** (*bool, default=True*) – If True, return the loaded *EpiModel*; otherwise return the path of the cached file.
- **force** (*bool, default=False*) – Re-download the file even if it is already cached.
- **seed** (*int, optional*) – Seed for the random number generator of the loaded model.
- **rng** (*numpy.random.Generator, optional*) – Random number generator for the loaded model.

Returns
The loaded model, or the local path of the cached file.

Return type
EpiModel or str

6.4 NetworkEpiModel

class epidemik.**NetworkEpiModel**(*network*, *compartments=None*, *seed=None*, *rng=None*, *name=None*)

Bases: *EpiModel*

Compartmental epidemic model that lives on a contact network.

Unlike *EpiModel*, which assumes a homogeneously mixed population, *NetworkEpiModel* simulates individual nodes that can only infect their neighbours along the edges of the provided networkx.Graph. Only stochastic simulation is supported – numerical integration of the corresponding ODEs is not meaningful in this setting.

network

The contact network.

Type
networkx.Graph

kavg_

Average degree of the contact network.

Type
float

population_

State of every node at every time step (one column per node), available after *simulate()*.

Type
pandas.DataFrame

values_

Number of nodes in each compartment at every time step, available after *simulate()*.

Type
pandas.DataFrame

__init__(*network*, *compartments=None*, *seed=None*, *rng=None*, *name=None*)

Initialize a network-based epidemic model.

Parameters

- **network** (*networkx.Graph*) – Contact network over which the epidemic will spread. Nodes represent individuals and edges represent contacts that can transmit the disease. Node labels can be of any hashable type.
- **compartments** (*list of str, optional*) – Optional list of compartment names. Just like in *EpiModel*, compartments are also added implicitly the first time they appear in a transition.
- **seed** (*int, optional*) – Seed for the random number generator used by *simulate()*.
- **rng** (*numpy.random.Generator, optional*) – Random number generator to use instead of a new seeded one.
- **name** (*str, optional*) – Human-readable name of the model.

integrate(**args*, ***kwargs*)

Not supported for network models.

Raises
NotImplementedError – Always. Network models can only be explored via *simulate()*.

add_interaction(
source: str,
target: str,
agent: str,
rate=None,
rescale: bool = False,
***rates*,
) → None

Add a contagious interaction between two compartments.

Parameters

- **source** (*str*) – Name of the source compartment.
- **target** (*str*) – Name of the target compartment.
- **agent** (*str*) – Name of the agent compartment driving the interaction.
- **rate** (*float or str, optional*) – Per-contact transmission probability per time step (number, parameter name or expression).
- **rescale** (*bool, default=False*) – If True, divide the rate by the average degree of the underlying network. This is useful when porting a mean-field rate to a network setting while keeping R_0 comparable. The rescaled value is stored as a new parameter `<name>_rescaled` defined in terms of the original one.
- ****rates** – Alternatively, the rate as a single named parameter, e.g. `beta=0.05`.

add_spontaneous(*source: str*, *target: str*, *rate=None*, ***rates*) → None

Add a spontaneous transition between two compartments.

Parameters

- **source** (*str*) – Name of the source compartment.
- **target** (*str*) – Name of the target compartment.
- **rate** (*float or str, optional*) – Per-individual transition probability per time step (number, parameter name or expression).
- ****rates** – Alternatively, the rate as a single named parameter, e.g. `mu=0.1`.

add_vaccination(
source: str,
target: str,
rate=None,
start: int | None = None,
***rates*,
) → None

Add a time-gated vaccination transition between two compartments.

Behaves like *add_spontaneous()* but is only active for `time > start`.

Parameters

- **source** (*str*) – Name of the source compartment.
- **target** (*str*) – Name of the target compartment.
- **rate** (*float or str, optional*) – Per-individual vaccination probability per time step.
- **start** (*int*) – Time step at which the campaign begins.
- ****rates** – Alternatively, the rate as a single named parameter.

add_birth_rate(**args*, ***kwargs*)

Not supported for network models (raises NotImplementedError).

add_death_rate(**args*, ***kwargs*)

Not supported for network models (raises NotImplementedError).

add_viral_generation(**args*, ***kwargs*)

Not supported for network models (raises NotImplementedError).

add_age_structure(**args*, ***kwargs*)

Not supported for network models (raises NotImplementedError).

add_groups(**args*, ***kwargs*)

Not supported for network models (raises NotImplementedError).

simulate(
timesteps: int,
seeds: Dict,
t_min: int = 1,
susceptible: str | None = None,
***kwargs*,
) → None

Run a discrete-time stochastic simulation on the network.

At every time step each active node attempts to infect its neighbours and may also undergo spontaneous (or, once started, vaccination) transitions. The per-node state trajectory is stored in *population_* and the aggregate compartment counts over time are stored in *values_*. In both data frames the first row (`t_min`) holds the initial conditions. All random numbers come from `rng`. Seasonality is not supported on networks.

Parameters

- **timesteps** (*int*) – Number of time steps (rows) to produce, including the initial condition.
- **seeds** (*dict*) – Mapping `{node_label: compartment_name}` that sets the initial state of a subset of network nodes. All other nodes are placed in the susceptible compartment.
- **t_min** (*int, default=1*) – Time label of the first row of *population_* and *values_*.
- **susceptible** (*str, optional*) – Name of the susceptible compartment. If omitted, the compartment without incoming transitions is used (as for *EpiModel*).
- ****kwargs** – Reserved for future use; currently ignored.

Raises

ValueError – If a seed node is not part of the network, if a seed state is not a compartment of the model, or if the susceptible compartment cannot be determined.

Warns

UserWarning – If the spontaneous transition probabilities out of a compartment sum to more than 1 (they are renormalized).

R0() → float | None

Compute the network-corrected basic reproduction number.

The underlying mean-field R_0 (computed by *EpiModel.R0()*) is multiplied by the average degree of the contact network, since each infected node only has $\langle k \rangle$ contacts on average.

Returns

Network R_0 rounded to two decimals, or `None` if the mean-field R_0 cannot be computed.

Return type

float or None

6.5 MetaEpiModel

class epidemik.**MetaEpiModel**(
travel_graph: DataFrame,
populations: DataFrame,
population: str = 'Population',
seed=None,
rng=None,
)

Bases: `object`

Metapopulation compartmental epidemic model.

A *MetaEpiModel* is a collection of independent *EpiModel* instances – one per sub-population – coupled together by a travel matrix. At every simulation step each sub-population is advanced by one stochastic time step, after which individuals move between sub-populations according to the travel probabilities. The same disease model is shared across every sub-population.

travel_graph

Row-stochastic travel matrix, indexed and columned by sub-population label.

Type
pandas.DataFrame

populations

Data frame with the total population of each sub-population.

Type
pandas.DataFrame

models

Mapping from sub-population label to its *EpiModel*.

Type
dict

compartments_

Current number of individuals in each compartment (columns) of each sub-population (rows). Available after *simulate()*.

Type
pandas.DataFrame

__init__(
travel_graph: DataFrame,
populations: DataFrame,
population: str = 'Population',
seed=None,
rng=None,
)

Initialize a metapopulation model.

Parameters

- **travel_graph** (*pandas.DataFrame*) – Square data frame indexed and columned by sub-population labels. Entry `(i, j)` is the probability that an individual currently in sub-population `i` travels to sub-population `j` during a single time step. Each row must sum to 1.

- **populations** (*pandas.DataFrame*) – Data frame indexed by sub-population label containing at least one column with the total population of each sub-population.
- **population** (*str, default='Population'*) – Name of the column in `populations` that holds the population counts.
- **seed** (*int, optional*) – Seed for the random number generator shared by the travel step and every sub-population.
- **rng** (*numpy.random.Generator, optional*) – Random number generator to use instead of a new seeded one.

Raises

ValueError – If `travel_graph` is not square, if its rows and columns are not labelled by the same sub-populations, if its rows do not sum to 1, or if `populations` is missing a sub-population or the `population` column.

__repr__() → str

Return a human-readable description of the metapopulation model.

Returns

Multi-line string describing the number of sub-populations and the underlying disease model.

Return type

str

define_parameters(***kwargs*) → None

Define named parameters in every sub-population.

See *EpiModel.define_parameters()*.

property params

Named parameters of the disease model (shared by every sub-population).

add_interaction(
source: str,
target: str,
agent: str,
rate=None,
norm: bool = True,
***rates*,
) → None

Add an interaction transition to every sub-population.

The transition is added to every *EpiModel* managed by this metapopulation model – the disease is assumed to be the same everywhere. See *EpiModel.add_interaction()* for the arguments.

add_spontaneous(*source: str*, *target: str*, *rate=None*, ***rates*) → None

Add a spontaneous transition to every sub-population.

See *EpiModel.add_spontaneous()* for the arguments.

add_vaccination(
source: str,
target: str,
rate=None,
start: int | None = None,
***rates*,
) → None

Add a time-gated vaccination transition to every sub-population.

See *EpiModel.add_vaccination()* for the arguments.

add_viral_generation(
source: str,
target: str,
source_rate=None,
target_rate=None,
***rates*,
) → None

Add a viral generation transition to every sub-population.

See *EpiModel.add_viral_generation()* for the arguments.

add_birth_rate(
rate=None,
comps: List | None = None,
fixed: bool = False,
global_rate: bool = True,
from_group: bool = False,
***rates*,
) → None

Attach a birth rate to one or more compartments in every sub-population.

See *EpiModel.add_birth_rate()* for the arguments.

add_death_rate(
rate=None,
comps: List | None = None,
fixed: bool = False,
global_rate: bool = False,
***rates*,
) → None

Attach a death rate to one or more compartments in every sub-population.

See *EpiModel.add_death_rate()* for the arguments.

add_groups(*groups: Dict[str, List[str]]*) → None

Declare compartment groups (e.g. host/vector) in every sub-population.

Forwards to *EpiModel.add_groups()* on every underlying *EpiModel*, so the same multi-group structure is used everywhere. See *EpiModel.add_groups()* for the semantics.

Note that *simulate()* only seeds the *susceptible* compartment of every sub-population from `populations`; the initial size of the other groups can only be set in `seed_state` through the keyword arguments of *simulate()*.

Parameters
groups (*dict of str -> list of str*) – Mapping from group name to the list of compartments in that group.

R0() → float | None

Compute the basic reproduction number of the underlying disease.

Since every sub-population shares the same disease model, the R_0 of the metapopulation model is simply the R_0 of any individual sub-population.

Returns
Basic reproduction number, or `None` if it cannot be computed.

Return type
float or None

`get_state`(*state: str*) → *EpiModel*

Return the *EpiModel* for a given sub-population.

Parameters
state (*str*) – Label of the sub-population.

Returns
The underlying *EpiModel* driving `state`.

Return type
EpiModel

`simulate`(
timesteps: int | None = None,
t_min: int = 1,
seasonality: ndarray | None = None,
seed_state: str | None = None,
susceptible: str = 'S',
***kwargs*,
) → None

Run a coupled stochastic simulation across every sub-population.

Each time step alternates between an *epidemic* phase (every sub-population advances by one step of its own *EpiModel*) and a *travel* phase (individuals are redistributed across sub-populations according to *travel_graph*). The initial epidemic seed is placed in `seed_state`.

Parameters

- **timesteps** (*int*) – Number of time steps to simulate. (The old spelling `timestamp` is still accepted but deprecated.)
- **t_min** (*int, default=1*) – Starting time of the simulation.
- **seasonality** (*numpy.ndarray, optional*) – Array of seasonal forcing factors (typically of length 365), forwarded to every sub-population.
- **seed_state** (*str*) – Label of the sub-population where the epidemic seeds are placed. Must be provided.
- **susceptible** (*str, default='S'*) – Name of the compartment representing susceptible individuals.
- ****kwargs** – Initial size of each non-susceptible compartment in `seed_state` (e.g. `I=10`). The susceptible compartment is adjusted automatically so that the total population of `seed_state` remains unchanged.

Raises

- *NotInitialized* – If `seed_state` is not provided.
- `ValueError` – If `seed_state` is not a known sub-population, if `susceptible` is not a compartment of the model, or if the seeds exceed the population of `seed_state`.

`integrate`(**args*, ***kwargs*)

Not supported for metapopulation models.

Raises
`NotImplementedError` – Always. Metapopulation models can only be explored via *simulate()*.

draw_model(
ax: Axes | None = None,
show: bool = True,
) → None

Draw the underlying disease model.

Delegates to *EpiModel.draw_model()* on the first sub-population's model, since every sub-population shares the same disease.

Parameters

- **ax** (*matplotlib.axes.Axes, optional*) – Axes to draw onto. If None a new figure is created.
- **show** (*bool, default=True*) – Whether to call matplotlib.pyplot.show() after drawing.

plot(
title: str | None = None,
normed: bool = True,
layout=None,
compartment: str = 'I',
figsize=(16, 22),
***kwargs*,
) → Axes

Plot the infection time series for every sub-population.

Each sub-population is rendered as a small panel whose position is given by `layout` (or by an automatic square layout if `layout` is omitted). Inside each panel the trajectory of `compartment` is drawn normalized by its maximum, a vertical line marks the time of the local peak, and the panel colour encodes the number of time steps between the start of the simulation and that peak.

Parameters

- **title** (*str, optional*) – Title to set on the resulting axes.
- **normed** (*bool, default=True*) – Unused: every curve is always normalized by its own maximum. Kept for API symmetry with *EpiModel.plot()*.
- **layout** (*pandas.DataFrame, optional*) – Optional layout with columns `x` and `y` indexed by sub-population label, providing the on-screen position of every panel. If omitted, an automatic grid layout is generated.
- **compartment** (*str, default='I'*) – Compartment whose trajectory is drawn in every panel.
- **figsize** (*tuple, default=(16, 22)*) – Size of the figure created for the plot.
- ****kwargs** – Reserved for future use.

Returns
The axes that were drawn on.

Return type
matplotlib.axes.Axes

Raises

- ***NotInitialized*** – If *simulate()* has not been called yet.
- **ValueError** – If `compartment` is not a compartment of the model.

plot_peaks(*compartment: str = 'I'*) → Axes

Render a heat-map of the infection peak across sub-populations.

For every sub-population, the time series of `compartment` is normalized by its maximum value and stacked vertically into a heat-map. The result highlights how the epidemic peak diffuses across sub-populations over time.

Parameters
compartment (*str, default='I'*) – Compartment whose trajectory is used.

Returns
The axes that were drawn on.

Return type
matplotlib.axes.Axes

Raises

- ***NotInitialized*** – If *simulate()* has not been called yet.
- **ValueError** – If `compartment` is not a compartment of the model.

6.6 epidemik.utils

Shared utilities used across the *epidemik* package.

This module exposes:

- *NotInitialized*, raised when a model is used before it has been run or fully set up;
- *Parameters*, the dictionary of named model parameters (which may be numbers or expressions in terms of other parameters);
- *epi_colors*, the default colour palette used by the plotting helpers;
- helpers used by *EpiModel.download_model* and *EpiModel.list_models* to locate the model repository and the local cache.

exception epidemik.utils.**NotInitialized**

Bases: Exception

Raised when a model is used before it has been fully initialised.

Typical situations include calling *plot()* before either *integrate()* or *simulate()*, or calling *simulate()* without specifying a `seed_state`.

class epidemik.utils.**Parameters**(***kwargs*)

Bases: dict

Named parameters of a model.

Behaves like a dictionary whose values are either numbers or *string expressions* written in terms of other parameters, e.g. `"beta/2"` or `"2*mu"`. Reading a key (`params["mu"]`) always returns the evaluated numeric value, while *raw()* returns what was stored.

Examples

```
>>> params = Parameters(beta=0.2)
>>> params["mu"] = "beta/2"
>>> params["mu"]
0.1
```

(continues on next page)

(continued from previous page)

```
>>> params.raw("mu")
'beta/2'
```

get(*key*, *default=None*)

Return the value for key if key is in the dictionary, else default.

define_parameters(***kwargs*) → None

Define one or more parameters.

Parameters

****kwargs** – `name=value` pairs. Numeric strings (`"0.2"`) are converted to floats; any other string is kept as an expression to be evaluated lazily against the other parameters.

Raises

TypeError – If a value is neither a number nor a string.

raw(*key*)

Return the stored value of `key` (number or expression string).

compute_parameter(*param: str*, *_stack=()*) → float

Evaluate a parameter.

Parameters

param (*str*) – Name of the parameter.

Returns

The numeric value of the parameter, evaluating any expression (recursively) in terms of the other parameters.

Return type

float

Raises

- **KeyError** – If `param` (or a parameter it refers to) is not defined.
- **ValueError** – If the expression cannot be evaluated, or if parameters refer to each other in a cycle.

resolve() → Dict[str, float]

Return a plain `{name: float}` dictionary with every parameter evaluated.

copy() → a shallow copy of D

epidemik.utils.**EPI_COLORS = {'D': '#8b8b8b', 'E': '#f9e351', 'I': '#cf51f9', 'R': '#70bf41', 'S': '#51a7f9'}**

Alias of *epi_colors* kept for backwards compatibility.

epidemik.utils.**OFFICIAL_REPO = 'https://github.com/DataForScience/epidemik/'**

GitHub repository hosting the official library of model files.

epidemik.utils.**get_cache_directory**() → str

Return the directory where downloaded model files are cached.

The location can be overridden with the `EPIDEMIK_CACHE_DIR` environment variable. Otherwise `~/Library/Caches/epidemik` is used on macOS and `$XDG_CACHE_HOME/epidemik` (defaulting to `~/.cache/epidemik`) elsewhere.

epidemik.utils.**get_remote_path**(*repo: str | None = None*) → str

Return the location of the model files in a repository.

Parameters

repo (*str, optional*) – Either the URL of a GitHub repository (defaults to *OFFICIAL_REPO*) or the path of a local directory containing model files.

Returns

Base URL (ending with `/`) from which `<name>.yaml` files can be fetched, or the local directory path.

Return type

str

Raises

ValueError – If `repo` is neither a GitHub URL nor an existing directory.

epidemik.utils.**epi_colors = <colour palette keyed by compartment initial>**

Default colours used by the plotting helpers, keyed by the first letter of the compartment name (`"S"` -> blue, `"E"` -> yellow, `"I"` -> magenta, `"R"` -> green, `"D"` -> grey). Any other compartment falls back to orange.

CHAPTER

SEVEN

CHANGELOG

Notable changes between releases of `epidemik`. Older releases are listed on the GitHub releases page and on PyPI.

7.1 0.2.0 (2026-09-15)

7.1.1 New features

- Named parameters: every rate is stored in *EpiModel.params* (a *Parameters*) and can be given as a number, as a keyword (`beta=0.3`) or as an expression of other parameters (`mu="beta/2"`); *EpiModel.define_parameters* declares them up front.
- Reproducible simulations: every model takes `seed`/`rng` arguments and draws all random numbers from its *rng*; a *MetaEpiModel* shares one generator across the travel step and every sub-population.
- Within-host models: `norm=False` interactions and *EpiModel.add_viral_generation*.
- Birth and death rates gained `fixed` and `global_rate` modes; births are now proportional to the total population by default.
- *EpiModel.save_model*, *EpiModel.load_model*, *EpiModel.list_models* and *EpiModel.download_model* read and write YAML model files (including demographics, groups, vaccination and viral generation) and fetch the model library hosted on GitHub, with a local cache. `models/` ships six ready-made models.
- *NetworkEpiModel* supports *add_vaccination()* and several interactions with the same source and agent.
- *EpiModel.reset* discards previous results; `EpiModel(name=...)` names a model.
- *EpiModel.add_groups* partitions the compartments into named groups (e.g. hosts and vectors). Interactions are then normalized by the population of the agent's group, which yields the standard host-vector dynamics. *EpiModel.add_birth_rate* gained a `from_group` flag for births proportional to a group total.
- *MetaEpiModel* forwards *add_groups()*, *add_birth_rate()* and *add_death_rate()* to every sub-population.
- *MetaEpiModel.plot* and *MetaEpiModel.plot_peaks* accept a `compartment` argument instead of always using `"I"`; `plot` also honours `title` and accepts `figsize`.
- *NetworkEpiModel.simulate* accepts `t_min` and `susceptible` arguments and supports networks with arbitrary (non-integer) node labels.
- *EpiModel.single_step* continues from the current state when called without keyword arguments.
- Age-structured models accept initial conditions for individual strata (`S_a=...`), and birth/death rates attached before *add_age_structure()* are propagated to every stratum.

7.1.2 Behaviour changes

- `print(model)` / `repr(model)` now produce the YAML model format (compartments, parameters, transitions, demographics, groups, and R_0 as a comment) instead of the previous free-form listing.
- Edge and node attributes of `transitions` store parameter *names* rather than numeric rates; look the values up in `model.params`.
- *EpiModel.add_birth_rate* defaults to births proportional to the total population (`global_rate=True`); pass `global_rate=False` for the previous per-compartment behaviour. Its first positional argument is the rate; a list of compartments is still accepted there for compatibility.
- `requires-python` is now `>=3.9` and `pyyaml` is a dependency.
- The project is built with hatchling and developed with uv (`uv sync`, `uv run pytest`, `uv build`); dependencies are pinned in `uv.lock` and the GitHub workflows use uv. The poetry configuration and `requirements.txt` are gone.
- Interaction agents are registered as compartments, so a compartment that only appears as an agent (e.g. an environmental reservoir) no longer crashes *integrate()*.
- *MetaEpiModel.simulate* starts a fresh simulation in every sub-population on each call (previously the histories accumulated) and raises `ValueError` when the seeds exceed the population of `seed_state`.
- The susceptible classes used by R_0 (and by *NetworkEpiModel*) are the interaction sources fed only by infected or post-infection compartments, so the result no longer depends on whether waning immunity is present.
- *NetworkEpiModel* raises `NotImplementedError` for births, deaths, viral generation, groups and age structure instead of silently ignoring them.
- *EpiModel.simulate* now stores the initial conditions in the first row of `values_` (at `t_min`) and produces exactly `timesteps` rows, exactly like *integrate()*. Previously the first row was the state *after* one step, so stochastic and deterministic trajectories were offset by one time step.
- *NetworkEpiModel.simulate* indexes `population_` and `values_` by time starting at `t_min` (default 1) rather than 0, labels the columns of `population_` by node, and always includes every compartment of the model in `values_` (in model order).
- The initial conditions of age-structured models are now split across age groups deterministically in *integrate()* (previously a random multinomial draw made integration non-reproducible).
- The first argument of *MetaEpiModel.simulate* is now called `timesteps`; the old `timestamp` spelling still works but emits a `DeprecationWarning`. The simulation always runs `timesteps` steps regardless of `t_min`.
- *NetworkEpiModel.R0* no longer requires a compartment named `"R"` to exist.
- Unknown compartment names passed as initial conditions to *integrate()* or *simulate()* are ignored with a warning (they were silently ignored by `integrate` and crashed `simulate`).
- `model["X"]` and *plot()* raise *NotInitialized* when the model has not been run yet; the constructor of *MetaEpiModel* validates the travel matrix and populations and raises `ValueError` on bad input.

7.1.3 Bug fixes

- *MetaEpiModel.plot* crashed unless a sub-population was called `"NY"`, and used the time label of the peak as a positional index. *MetaEpiModel.draw_model* always crashed.
- *MetaEpiModel.simulate* ignored its `seasonality` argument.
- *NetworkEpiModel* truncated compartment names to a single character during simulation, so multi-letter compartments never transitioned; only the last spontaneous transition out of a compartment was kept; and non-integer node labels were ignored.

- *EpiModel.simulate* crashed with `pvals < 0` when the transition probabilities out of a compartment exceeded 1, crashed on age-structured models, and overwrote (instead of adding) the contributions of several transitions with the same source and target.
- Multi-group models produced `NaN` when a group was empty.
- In *EpiModel.simulate* deaths were sampled independently of the other transitions out of a compartment, so compartments could become negative. Deaths now compete with the other transitions in the same multinomial draw.
- Compartments added to an age-structured model after *add_age_structure()* were missing from `values_`.
- The automatically detected susceptible compartment (used by *NetworkEpiModel* and by R_0) could be a seed-only infectious compartment; compartments that are the source of an interaction are now preferred.
- `seasonality` arrays of any length are accepted (the index wraps around `len(seasonality)` instead of a hard-coded 365), and `timesteps < 1` raises `ValueError` instead of failing obscurely.
- *EpiModel.R0* returned `None` for models with spontaneous transitions out of non-infectious compartments (e.g. waning immunity `R -> S`) and hid every error behind a bare `except`.
- *EpiModel.plot* ignored the `ax` argument, and *EpiModel.draw_model* drew the labels on the current axes instead of `ax`.
- `model["X"]` raised `KeyError` for unknown names on models without age structure instead of returning `None`.
- *EpiModel.add_age_structure* validates its inputs with `ValueError` instead of `assert`.

7.1.4 Documentation

- Every public class and method has a complete NumPy-style docstring, the Sphinx configuration uses `sphinx.ext.napoleon` and `sphinx.ext.intersphinx`, and the user guide covers *EpiModel*, *NetworkEpiModel* and *MetaEpiModel* in detail.

7.2 0.1 (2024-06-10)

- Type hints across the public API.
- Sphinx documentation with autodoc support.

7.3 0.0.17 – 0.0.22 (2024-04 – 2024-05)

- Added *MetaEpiModel*.
- Added the test suite and continuous integration.
- Improved the plotting functions of *EpiModel*.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

epidemik, 17
epidemik.utils, 36

INDEX

Symbols

__getattr__() (*epidemik.EpiModel method*), 23
__getitem__() (*epidemik.EpiModel method*), 24
__init__() (*epidemik.EpiModel method*), 18
__init__() (*epidemik.MetaEpiModel method*), 31
__init__() (*epidemik.NetworkEpiModel method*), 28
__repr__() (*epidemik.EpiModel method*), 25
__repr__() (*epidemik.MetaEpiModel method*), 32

A

add_age_structure() (*epidemik.EpiModel method*), 22
add_age_structure() (*epidemik.NetworkEpiModel method*), 30
add_birth_rate() (*epidemik.EpiModel method*), 20
add_birth_rate() (*epidemik.MetaEpiModel method*), 33
add_birth_rate() (*epidemik.NetworkEpiModel method*), 30
add_death_rate() (*epidemik.EpiModel method*), 21
add_death_rate() (*epidemik.MetaEpiModel method*), 33
add_death_rate() (*epidemik.NetworkEpiModel method*), 30
add_groups() (*epidemik.EpiModel method*), 22
add_groups() (*epidemik.MetaEpiModel method*), 33
add_groups() (*epidemik.NetworkEpiModel method*), 30
add_interaction() (*epidemik.EpiModel method*), 19
add_interaction() (*epidemik.MetaEpiModel method*), 32
add_interaction() (*epidemik.NetworkEpiModel method*), 28
add_spontaneous() (*epidemik.EpiModel method*), 19
add_spontaneous() (*epidemik.MetaEpiModel method*), 32
add_spontaneous() (*epidemik.NetworkEpiModel method*), 29
add_vaccination() (*epidemik.EpiModel method*), 21
add_vaccination() (*epidemik.MetaEpiModel method*), 32
add_vaccination() (*epidemik.NetworkEpiModel method*), 29
add_viral_generation() (*epidemik.EpiModel method*), 20
add_viral_generation() (*epidemik.MetaEpiModel method*), 33
add_viral_generation() (*epidemik.NetworkEpiModel method*), 30

C

compartments_ (*epidemik.MetaEpiModel attribute*), 31
compute_parameter() (*epidemik.utils.Parameters method*), 37
copy() (*epidemik.utils.Parameters method*), 37

D

define_parameters() (*epidemik.EpiModel method*), 18
define_parameters() (*epidemik.MetaEpiModel method*), 32
define_parameters() (*epidemik.utils.Parameters method*), 37
download_model() (*epidemik.EpiModel static method*), 27
draw_model() (*epidemik.EpiModel method*), 25
draw_model() (*epidemik.MetaEpiModel method*), 34

E

EPI_COLORS (*in module epidemik.utils*), 37
epi_colors (*in module epidemik.utils*), 38
epidemik
    module, 17
epidemik.utils
    module, 36
EpiModel (*class in epidemik*), 17

G

get() (*epidemik.utils.Parameters method*), 37
get_cache_directory() (*in module epidemik.utils*), 37
get_remote_path() (*in module epidemik.utils*), 37
get_state() (*epidemik.MetaEpiModel method*), 34

I

integrate() (*epidemik.EpiModel method*), 24

integrate() (*epidemik.MetaEpiModel method*), 34
integrate() (*epidemik.NetworkEpiModel method*), 28

K

kavg_ (*epidemik.NetworkEpiModel attribute*), 28

L

list_models() (*epidemik.EpiModel static method*), 27
load_model() (*epidemik.EpiModel static method*), 26

M

MetaEpiModel (*class in epidemik*), 31
models (*epidemik.MetaEpiModel attribute*), 31
module
    epidemik, 17
    epidemik.utils, 36

N

name (*epidemik.EpiModel attribute*), 18
network (*epidemik.NetworkEpiModel attribute*), 28
NetworkEpiModel (*class in epidemik*), 28
NotInitialized, 36

O

OFFICIAL_REPO (*in module epidemik.utils*), 37

P

Parameters (*class in epidemik.utils*), 36
params (*epidemik.EpiModel attribute*), 18
params (*epidemik.MetaEpiModel property*), 32
plot() (*epidemik.EpiModel method*), 23
plot() (*epidemik.MetaEpiModel method*), 35
plot_peaks() (*epidemik.MetaEpiModel method*), 35
population_ (*epidemik.NetworkEpiModel attribute*), 28
populations (*epidemik.MetaEpiModel attribute*), 31

R

R0() (*epidemik.EpiModel method*), 26
R0() (*epidemik.MetaEpiModel method*), 33
R0() (*epidemik.NetworkEpiModel method*), 30
raw() (*epidemik.utils.Parameters method*), 37
reset() (*epidemik.EpiModel method*), 25
resolve() (*epidemik.utils.Parameters method*), 37
rng (*epidemik.EpiModel attribute*), 18

S

save_model() (*epidemik.EpiModel method*), 26
simulate() (*epidemik.EpiModel method*), 24
simulate() (*epidemik.MetaEpiModel method*), 34
simulate() (*epidemik.NetworkEpiModel method*), 30
single_step() (*epidemik.EpiModel method*), 25

T

transitions (*epidemik.EpiModel attribute*), 18
travel_graph (*epidemik.MetaEpiModel attribute*), 31

V

values_ (*epidemik.EpiModel attribute*), 18
values_ (*epidemik.NetworkEpiModel attribute*), 28
values_ages_ (*epidemik.EpiModel attribute*), 18