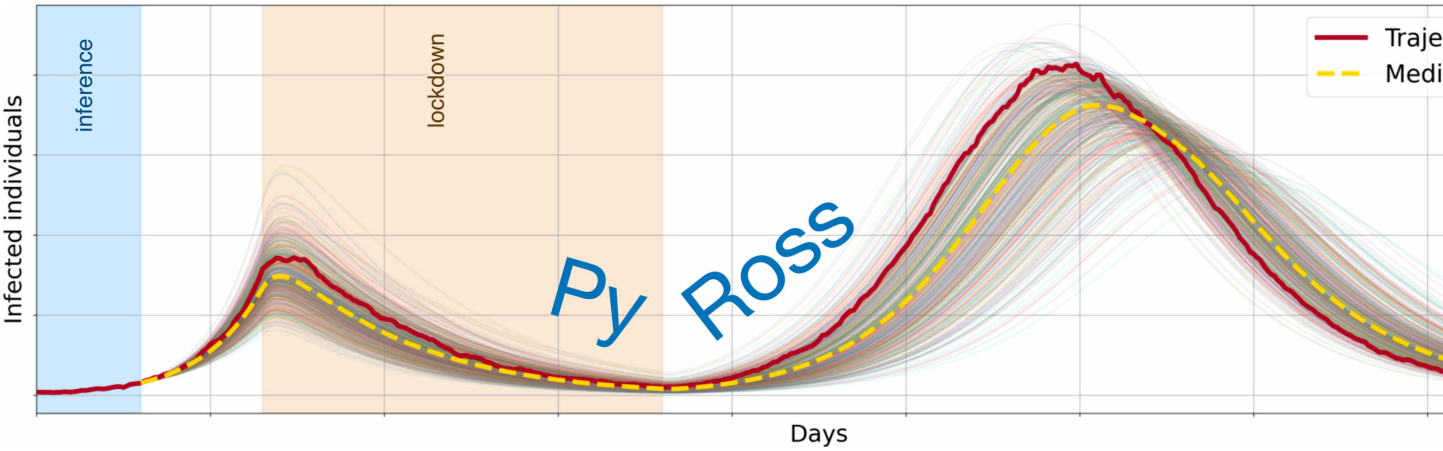
PyRoss

Release 1.0.0

Jul 11, 2023

Contents



PyRoss is a numerical library for inference, prediction and non-pharmaceutical interventions in age-structured epidemiological compartment models. The library is designed to be model-agnostic and allows the user to define models using a Python dictionary.

The library supports models formulated stochastically (as chemical master equations) or deterministically (as systems of differential equations). Inference on pre-defined or user-defined models is performed using model-adapted Gaussian processes on the epidemiological manifold or its tangent space. This method allows for latent variable inference and fast computation of the model evidence and the Fisher information matrix. These estimates are convolved with the instrinsic stochasticty of the dynamics to provide Bayesian forecasts of the progress of the epidemic.

CHAPTER 1

Installation

1.1 From a checkout of PyRoss GitHub repository

This is the recommended way as it downloads a whole suite of examples along with the package.

1.1.1 Install PyRoss and an extended list of dependencies using

```
>> git clone https://github.com/rajeshrinet/pyross.git
>> cd pyross
>> pip install -r requirements.txt
>> python setup.py install
```

1.1.2 Install PyRoss and an extended list of dependencies, via Anaconda, in an environment named `pyross`:

```
>> git clone https://github.com/rajeshrinet/pyross.git
>> cd pyross
>> make env
>> conda activate pyross
>> make
```

1.2 Via pip

Install the latest PyPI version

```
>> pip install pyross
```

See also installation instructions and more details in the README.md on GitHub.

CHAPTER 2

Tutorial examples

Please have a look at the examples folder for Jupyter notebook examples on GitHub.

CHAPTER 3

API Reference

3.1 Deterministic simulations

Deterministic simulations with compartment models and age structure

A list of methods for deterministic simulations of age-structured compartment models along with link to notebook examples is given below.

3.1.1 Model

class pyross.deterministic.**Model**

Generic user-defined epidemic model.

...

Parameters

- **model_spec** (*dict*) – A dictionary specifying the model. See *Examples*.
- **parameters** (*dict*) – Contains the values for the parameters given in the model specification. All parameters can be float if not age-dependent, and np.array(M,) if age-dependent
- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(M,)*) – Initial number in each compartment and class
- **time_dep_param_mapping** (*python function, optional*) – A user-defined function that takes a dictionary of time-independent parameters and time as an argument, and returns a dictionary of the parameters of model_spec. Default: Identical mapping of the dictionary at all times.

Examples

An example of model_spec and parameters for SIR class with a constant influx

```
>>> model_spec = {
        "classes" : ["S", "I"],
        "S" : {
            "constant"  : [ ["k"] ],
            "infection" : [ ["I", "S", "-beta"] ]
        },
        "I" : {
            "linear"    : [ ["I", "-gamma"] ],
            "infection" : [ ["I", "S", "beta"] ]
        }
    }
>>> parameters = {
        'beta': 0.1,
        'gamma': 0.1,
        'k': 1,
    }
```

model_class_data()

Parameters **data** (*dict*) – The object returned by *simulate*.

Returns

Return type The population of class *model_class_key* as a time series

simulate()

Simulates a compartment model given initial conditions, choice of integrator and other parameters. Returns the time series data and parameters in a dict. Internally calls the method 'simulator' of CommonMethods

...

Parameters

- **x0** (*np.array or dict*) – Initial conditions. If it is an array it should have length M*(model_dimension-1), where x0[i + j*M] should be the initial value of model class i of age group j. The removed R class must be left out. If it is a dict then it should have a key corresponding to each model class, with a 1D array containing the initial condition for each age group as value. One of the classes may be left out, in which case its initial values will be inferred from the others.
- **contactMatrix** (*python function(t)*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j
- **Tf** (*float*) – Final time of integrator
- **Nf** (*Int*) – Number of time points to evaluate.
- **Ti** (*float, optional*) – Start time of integrator. The default is 0.
- **integrator** (*TYPE, optional*) – Integrator to use either from scipy.integrate or odespy. The default is 'odeint'.
- **maxNumSteps** (*int, optional*) – maximum number of steps the integrator can take. The default is 100000.
- ****kwargs** (*kwargs for integrator*) –

Returns **data** – X: output path from integrator, t : time points evaluated at, 'param': input param to integrator.

Return type dict

- Link to example notebook

3.1.2 Spp

class pyross.deterministic.**Spp**

This is a slightly more specific version of the class *Model*.

Spp is still supported for backward compatibility.

Model class is recommended over *Spp* for new users.

The *Spp* class works like *Model* but infection terms use a single class *S*

...

Parameters

- **model_spec** (*dict*) – A dictionary specifying the model. See *Examples*.
- **parameters** (*dict*) – Contains the values for the parameters given in the model specification. All parameters can be float if not age-dependent, and np.array(M,) if age-dependent
- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(M,)*) – Initial number in each compartment and class
- **time_dep_param_mapping** (*python function, optional*) – A user-defined function that takes a dictionary of time-independent parameters and time as an argument, and returns a dictionary of the parameters of model_spec. Default: Identical mapping of the dictionary at all times.

Examples

An example of model_spec and parameters for SIR class with a constant influx

```
>>> model_spec = {
        "classes" : ["S", "I"],
        "S" : {
            "constant"  : [ ["k"] ],
            "infection" : [ ["I", "-beta"] ]
        },
        "I" : {
            "linear"    : [ ["I", "-gamma"] ],
            "infection" : [ ["I", "beta"] ]
        }
    }
>>> parameters = {
        'beta': 0.1,
        'gamma': 0.1,
        'k': 1,
    }
```

- Link to example notebook

3.1.3 SppQ

class pyross.deterministic.**SppQ**

User-defined epidemic model with quarantine stage.

This is a slightly more specific version of the class *Model*.

SppQ is still supported for backward compatibility.

Model class is recommended over *SppQ* for new users.

To initialise the SppQ model, . . .

Parameters

- **model_spec** (*dict*) – A dictionary specifying the model. See *Examples*.
- **parameters** (*dict*) – Contains the values for the parameters given in the model specification. All parameters can be float if not age-dependent, and np.array(M,) if age-dependent
- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(M,)*) – Initial number in each compartment and class
- **time_dep_param_mapping** (*python function, optional*) – A user-defined function that takes a dictionary of time-independent parameters and time as an argument, and returns a dictionary of the parameters of model_spec. Default: Identical mapping of the dictionary at all times.

Examples

An example of model_spec and parameters for SIR class with random testing (without false positives/negatives) and quarantine

```
>>> model_spec = {
        "classes" : ["S", "I"],
        "S" : {
            "infection" : [ ["I", "-beta"] ]
        },
        "I" : {
            "linear"    : [ ["I", "-gamma"] ],
            "infection" : [ ["I", "beta"] ]
        },
        "test_pos"  : [ "p_falsepos", "p_truepos", "p_falsepos"] ,
        "test_freq" : [ "tf", "tf", "tf"]
    }
>>> parameters = {
        'beta': 0.1,
        'gamma': 0.1,
        'p_falsepos': 0
        'p_truepos': 1
        'tf': 1
    }
```

model_class_data()

Parameters **data** (*dict*) – The object returned by *simulate*.

Returns

Return type The population of class *model_class_key* as a time series

simulate()

Simulates a compartment model given initial conditions, choice of integrator and other parameters. Returns the time series data and parameters in a dict. Internally calls the method 'simulator' of CommonMethods

. . .

Parameters

- **x0** (*np.array or dict*) – Initial conditions. If it is an array it should have length M*(model_dimension-1), where x0[i + j*M] should be the initial value of model class i of age group j. The removed R class must be left out. If it is a dict then it should have a key corresponding to each model class, with a 1D array containing the initial condition for each age group as value. One of the classes may be left out, in which case its initial values will be inferred from the others.
- **contactMatrix** (*python function(t)*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j
- **testRate** (*python function(t)*) – The total number of PCR tests performed per day
- **Tf** (*float*) – Final time of integrator
- **Nf** (*Int*) – Number of time points to evaluate.
- **Ti** (*float, optional*) – Start time of integrator. The default is 0.
- **integrator** (*TYPE, optional*) – Integrator to use either from scipy.integrate or odespy. The default is 'odeint'.
- **maxNumSteps** (*int, optional*) – maximum number of steps the integrator can take. The default value is 100000.
- ****kwargs** (*kwargs for integrator*) –

Returns **data** – X: output path from integrator, t : time points evaluated at, 'param': input param to integrator.

Return type dict

- Link to example notebook

3.1.4 SIR

class pyross.deterministic.**SIR**

Susceptible, Infected, Removed (SIR)

- Ia: asymptomatic
- Is: symptomatic

$$\dot{S}_i = -\lambda_i(t) S_i$$

$$\dot{I}^a_i = \alpha_i \lambda_i(t) S_i - \gamma_{I^a} I^a_i,$$

$$\dot{I}^s_i = \bar{\alpha}_i \lambda_i(t) S_i - \gamma_{I^s} I^s_i,$$

$$\dot{R}_i = \gamma_{I^a} I^a_i + \gamma_{I^s} I^s_i.$$

$$\lambda_i(t) = \beta \sum_{j=1}^{M} \left(C^a_{ij}(t) \frac{I^a_j}{N_j} + C^s_{ij}(t) \frac{I^s_j}{N_j} \right), \quad i, j = 1, \ldots M$$

...

Parameters

- **parameters** (*dict*) – Contains the following keys:

  alpha: float, np.array (M,) Fraction of infected who are asymptomatic.

beta: float, np.array (M,) Rate of spread of infection.

gIa: float, np.array (M,) Rate of removal from asymptomatic individuals.

gIs: float, np.array (M,) Rate of removal from symptomatic individuals.

fsa: float, np.array (M,) Fraction by which symptomatic individuals do not self-isolate.

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(M,)*) – Initial number in each compartment and class

Examples

An example of the SIR class

```
>>> M = 1                    # SIR model with no age structure
>>> Ni = 1000*np.ones(M)     # only one age group
>>> N = np.sum(Ni)           # total population
>>>
>>> beta  = 0.2              # Infection rate
>>> gIa   = 0.1              # Removal rate of asymptomatic infectives
>>> gIs   = 0.1              # Removal rate of symptomatic infectives
>>> alpha = 0                # Fraction of asymptomatic infectives
>>> fsa   = 1                # self-isolation of symtomatic infectives
>>>
>>> Ia0 = np.array([0])      # Intial asymptomatic infectives
>>> Is0 = np.array([1])      # Initial symptomatic
>>> R0  = np.array([0])      # No removed individuals initially
>>> S0  = N-(Ia0+Is0+R0)     # S + Ia + Is + R = N
>>>
>>> # there is no contact structure
>>> def contactMatrix(t):
>>>     return np.identity(M)
>>>
>>> # duration of simulation and data file
>>> Tf = 160;  Nt=160;
>>>
>>> # instantiate model
>>> parameters = {'alpha':alpha, 'beta':beta, 'gIa':gIa, 'gIs':gIs,'fsa':fsa}
>>> model = pyross.deterministic.SIR(parameters, M, Ni)
>>>
>>> # simulate model using two possible ways
>>> data1 = model.simulate(S0, Ia0, Is0, contactMatrix, Tf, Nt)
>>> data2 = model.simulator(np.concatenate((S0, Ia0, Is0)), contactMatrix, Tf, Nt)
```

simulate()

Simulates a compartment model given initial conditions, choice of integrator and other parameters. Returns the time series data and parameters in a dict. Internally calls the method 'simulator' of CommonMethods

...

Parameters

- **S0** (*np.array*) – Initial number of susceptables.
- **Ia0** (*np.array*) – Initial number of asymptomatic infectives.
- **Is0** (*np.array*) – Initial number of symptomatic infectives.

- **contactMatrix** (*python function(t)*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j
- **Tf** (*float*) – Final time of integrator
- **Nf** (*Int*) – Number of time points to evaluate.
- **Ti** (*float, optional*) – Start time of integrator. The default is 0.
- **integrator** (*TYPE, optional*) – Integrator to use either from scipy.integrate or odespy. The default is 'odeint'.
- **maxNumSteps** (*int, optional*) – maximum number of steps the integrator can take. The default is 100000.
- ****kwargs** (*kwargs for integrator*) –

Returns X: output path from integrator, t : time points evaluated at, 'param': input param to integrator.

Return type dict

- Link to example notebook

3.1.5 SIkR

class pyross.deterministic.**SIkR**

Susceptible, Infected, Removed (SIkR). Method of k-stages of I

$$\dot{S}_i = -\lambda_i(t) S_i,$$

$$\dot{I}_i^1 = k_E \gamma_E E_i^k - k_I \gamma_I I_i^1,$$

$$\dot{I}_i^k = k_I \gamma_I I_i^{(k-1)} - k_I \gamma_I I_i^k,$$

$$\dot{R}_i = k_I \gamma_I I_i^k.$$

$$\lambda_i(t) = \beta \sum_{j=1}^{M} \sum_{n=1}^{k} C_{ij}(t) \frac{I_j^n}{N_j},$$

...

Parameters

- **parameters** (*dict*) – Contains the following keys:

  beta: float Rate of spread of infection.

  gI: float Rate of removal from infectives.

  kI: int number of stages of infection.
- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(M,)*) – Initial number in each compartment and class

simulate()

Simulates a compartment model given initial conditions, choice of integrator and other parameters. Returns the time series data and parameters in a dict. Internally calls the method 'simulator' of CommonMethods

...

Parameters

- **S0** (*np.array*) – Initial number of susceptables.
- **I0** (*np.array*) – Initial number of infectives.
- **contactMatrix** (*python function(t)*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j
- **Tf** (*float*) – Final time of integrator
- **Nf** (*Int*) – Number of time points to evaluate.
- **Ti** (*float, optional*) – Start time of integrator. The default is 0.
- **integrator** (*TYPE, optional*) – Integrator to use either from scipy.integrate or odespy. The default is 'odeint'.
- **maxNumSteps** (*int, optional*) – maximum number of steps the integrator can take. The default is 100000.
- ****kwargs** (*kwargs for integrator*) –

Returns X: output path from integrator, t : time points evaluated at, 'param': input param to integrator.

Return type dict

- Link to example notebook

3.1.6 SEIR

class pyross.deterministic.**SEIR**

Susceptible, Exposed, Infected, Removed (SEIR)

- Ia: asymptomatic
- Is: symptomatic

$$\dot{S}_i = -\lambda_i(t) S_i$$

$$\dot{E}_i = \lambda_i(t) S_i - \gamma_E E_i$$

$$\dot{I}^a_i = \alpha_i \gamma^i_E E_i - \gamma_{I^a} I^a_i,$$

$$\dot{I}^s_i = \bar{\alpha}_i \gamma^i_E E_i - \gamma_{I^s} I^s_i,$$

$$\dot{R}_i = \gamma_{I^a} I^a_i + \gamma_{I^s} I^s_i.$$

$$\lambda_i(t) = \beta \sum_{j=1}^{M} \left(C^a_{ij}(t) \frac{I^a_j}{N_j} + C^s_{ij}(t) \frac{I^s_j}{N_j} \right), \quad i, j = 1, \ldots M$$

...

Parameters

- **parameters** (*dict*) – Contains the following keys:

  alpha: float, np.array (M,) Fraction of infected who are asymptomatic.

  beta: float, np.array (M,) Rate of spread of infection.

  gE: float, np.array (M,) Rate of removal from exposed individuals.

  gIa: float, np.array (M,) Rate of removal from asymptomatic individuals.

gIs: float, np.array (M,) Rate of removal from symptomatic individuals.

fsa: float, np.array (M,) Fraction by which symptomatic individuals do not self-isolate.

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(M,)*) – Initial number in each compartment and class

simulate()

Simulates a compartment model given initial conditions, choice of integrator and other parameters. Returns the time series data and parameters in a dict. Internally calls the method 'simulator' of CommonMethods

...

Parameters

- **S0** (*np.array*) – Initial number of susceptables.
- **E0** (*np.array*) – Initial number of exposed.
- **Ia0** (*np.array*) – Initial number of asymptomatic infectives.
- **Is0** (*np.array*) – Initial number of symptomatic infectives.
- **contactMatrix** (*python function(t)*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j
- **Tf** (*float*) – Final time of integrator
- **Nf** (*Int*) – Number of time points to evaluate.
- **Ti** (*float, optional*) – Start time of integrator. The default is 0.
- **integrator** (*TYPE, optional*) – Integrator to use either from scipy.integrate or odespy. The default is 'odeint'.
- **maxNumSteps** (*int, optional*) – maximum number of steps the integrator can take. The default is 100000.
- ****kwargs** (*kwargs for integrator*) –

Returns X: output path from integrator, t : time points evaluated at, 'param': input param to integrator.

Return type dict

- Link to example notebook

3.1.7 SEkIkR

class pyross.deterministic.**SEkIkR**

Susceptible, Exposed, Infected, Removed (SEkIkR). Method of k-stages of E and I

$$\dot{S}_i = -\lambda_i(t) S_i,$$

$$\dot{E}^1_i = \lambda_i(t) S_i - k_E \gamma_E E^1_i$$

$$\dot{E}^k_i = k_E \gamma_E E^{k-1}_i - k_E \gamma_E E^k_i$$

$$\dot{I}^1_i = k_E \gamma_E E^k_i - k_I \gamma_I I^1_i,$$

$$\dot{I}^k_i = k_I \gamma_I I^{(k-1)}_i - k_I \gamma_I I^k_i,$$

$$\dot{R}_i = k_I \gamma_I I_i^k.$$

$$\lambda_i(t) = \beta \sum_{j=1}^{M} \sum_{n=1}^{k} C_{ij}(t) \frac{I_j^n}{N_j},$$

... :param parameters: Contains the following keys:

beta: float Rate of spread of infection.

gI: float Rate of removal from infected individuals.

gE: float Rate of removal from exposed individuals.

kI: int number of stages of infectives.

kE: int number of stages of exposed.

Parameters

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(M,)*) – Initial number in each compartment and class

simulate()

Simulates a compartment model given initial conditions, choice of integrator and other parameters. Returns the time series data and parameters in a dict. Internally calls the method 'simulator' of CommonMethods

... :param S0: Initial number of susceptables. :type S0: np.array :param E0: Initial number of exposeds. :type E0: np.array :param I0: Initial number of infectives. :type I0: np.array :param contactMatrix: The social contact matrix C_{ij} denotes the

average number of contacts made per day by an individual in class i with an individual in class j

Parameters

- **Tf** (*float*) – Final time of integrator
- **Nf** (*Int*) – Number of time points to evaluate.
- **Ti** (*float, optional*) – Start time of integrator. The default is 0.
- **integrator** (*TYPE, optional*) – Integrator to use either from scipy.integrate or odespy. The default is 'odeint'.
- **maxNumSteps** (*int, optional*) – maximum number of steps the integrator can take. The default is 100000.
- ****kwargs** (*kwargs for integrator*) –

Returns X: output path from integrator, t : time points evaluated at, 'param': input param to integrator.

Return type dict

- Link to example notebook

3.1.8 SEAIRQ

class pyross.deterministic.**SEAIRQ**

Susceptible, Exposed, Asymptomatic and infected, Infected, Removed, Quarantined (SEAIRQ)

- Ia: asymptomatic

- Is: symptomatic
- E: exposed
- A: asymptomatic and infectious
- Q: quarantined

$$\dot{S}_i = -\lambda_i(t) S_i$$

$$\dot{E}_i = \lambda_i(t) S_i - (\gamma_E + \tau_E) A_i$$

$$\dot{A}_i = \gamma_E E_i - (\gamma_A + \tau_A) A_i$$

$$\dot{I}^a_i = \alpha_i \gamma_A A_i - (\gamma_{I^a} + \tau_{I^a}) I^a_i,$$

$$\dot{I}^s_i = \bar{\alpha}_i \gamma_A A_i - (\gamma_{I^s} + \tau_{I^a}) I^s_i,$$

$$\dot{R}_i = \gamma_{I^a} I^a_i + \gamma_{I^s} I^s_i.$$

$$\dot{Q}_i = \tau_S S_i + \tau_E E_i + \tau_A A_i + \tau_{I^s} I^s_i + \tau_{I^a} I^a_i$$

$$\lambda_i(t) = \beta \sum_{j=1}^{M} \left(C^a_{ij} \frac{I^a_j}{N_j} + C^a_{ij} \frac{A_j}{N_j} + C^s_{ij} \frac{I^s_j}{N_j} \right),$$

...

Parameters

- **parameters** (*dict*) – Contains the following keys:

  alpha: float Fraction of infected who are asymptomatic.

  beta: float, np.array (M,) Rate of spread of infection.

  gIa: float, np.array (M,) Rate of removal from asymptomatic individuals.

  gIs: float, np.array (M,) Rate of removal from symptomatic individuals.

  gE: float, np.array (M,) Rate of removal from exposed individuals.

  gA: float, np.array (M,) Rate of removal from activated individuals.

  fsa: float, np.array (M,) tE: float, np.array (M,)

  testing rate and contact tracing of exposeds

  tA: float, np.array (M,) testing rate and contact tracing of activateds

  tIa: float, np.array (M,) testing rate and contact tracing of asymptomatics

  tIs: float, np.array (M,) testing rate and contact tracing of symptomatics

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(M,)*) – Initial number in each compartment and class

simulate()

Simulates a compartment model given initial conditions, choice of integrator and other parameters. Returns the time series data and parameters in a dict. Internally calls the method 'simulator' of CommonMethods

...

Parameters

- **S0** (*np.array*) – Initial number of susceptables.

- **E0** (*np.array*) – Initial number of exposeds.
- **A0** (*np.array*) – Initial number of activateds.
- **Ia0** (*np.array*) – Initial number of asymptomatic infectives.
- **Is0** (*np.array*) – Initial number of symptomatic infectives.
- **Q0** (*np.array*) – Initial number of quarantineds.
- **contactMatrix** (*python function(t)*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j
- **Tf** (*float*) – Final time of integrator
- **Nf** (*Int*) – Number of time points to evaluate.
- **Ti** (*float, optional*) – Start time of integrator. The default is 0.
- **integrator** (*TYPE, optional*) – Integrator to use either from scipy.integrate or odespy. The default is 'odeint'.
- **maxNumSteps** (*int, optional*) – maximum number of steps the integrator can take. The default is 100000.
- ****kwargs** (*kwargs for integrator*) –

Returns

data – contains the following keys:

- X : output path from integrator
- t : time points evaluated at,
- 'param': input param to integrator.

Return type dict

- Link to example notebook

3.1.9 CommonMethods

class pyross.deterministic.**CommonMethods**

Parent class used for all classes listed below. It includes: a) Integrators used by various deterministic models listed below. b) Method to get time series of S, etc by passing a dict of data. c) Method to set the contactMatrix array, CM

A()

Parameters **data** (*Data dict*) –

Returns **A**

Return type Activated population time series

E()

Parameters **data** (*Data dict*) –

Returns **E**

Return type Exposed population time series

I()

Parameters **data** (*Data dict*) –

Returns Ia

Return type Asymptomatics population time series

Ia()

Parameters **data** (*Data dict*) –

Returns Ia

Return type Asymptomatics population time series

Is()

Parameters **data** (*Data dict*) –

Returns Is

Return type symptomatics population time series

R()

Parameters **data** (*Data dict*) –

Returns R

Return type Removed population time series

S()

Parameters **data** (*Data dict*) –

Returns S

Return type Susceptible population time series

Sx()

Parameters **data** (*Data dict*) –

Returns

Return type Generic compartment Sx

simulator()

Simulates a compartment model given initial conditions, choice of integrator and other parameters. Returns the time series data and parameters in a dict.

...

Parameters

- **x0** (*np.array*) – Initial state vector (number of compartment values). An array of size M*(model_dimension-1), where x0[i+j*M] should be the initial value of model class i of age group j. The removed R class must be left out. If Ni is dynamical, then the last M points store Ni.
- **contactMatrix** (*python function(t)*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j
- **Tf** (*float*) – Final time of integrator
- **Nf** (*Int*) – Number of time points to evaluate.
- **Ti** (*float, optional*) – Start time of integrator. The default is 0.

- **integrator** (*TYPE, optional*) – Integrator to use either from scipy.integrate or odespy. The default is 'odeint'.
- **maxNumSteps** (*int, optional*) – maximum number of steps the integrator can take. The default is 100000.
- ****kwargs** (*kwargs for integrator*) –

Returns **data** – X: output path from integrator, t : time points evaluated at, 'param': input param to integrator.

Return type dict

3.2 Stochastic simulations

Stochastic simulations with compartment models and age structure. Has Gillespie and tau-leaping implemented.

A list of methods for stochastic simulations of age-structured compartment models along with link to notebook examples is given below.

3.2.1 Model

class pyross.stochastic.**Model**

Generic user-defined epidemic model.

...

Parameters

- **model_spec** (*dict*) – A dictionary specifying the model. See *Examples*.
- **parameters** (*dict*) – A dictionary containing the model parameters. All parameters can be float if not age-dependent, and np.array(M,) if age-dependent. An optional element with the key 'seed' may be supplied as a seed for the pseudo-random number generator.
- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(3*M,)*) – Initial number in each compartment and class
- **time_dep_param_mapping** (*python function, optional*) – A user-defined function that takes a dictionary of time-independent parameters and time as an argument, and returns a dictionary of the parameters of model_spec. Default: Identical mapping of the dictionary at all times.

Examples

An example of model_spec and parameters for SIR class with a constant influx

```
>>> model_spec = {
        "classes" : ["S", "I"],
        "S" : {
            "constant"  : [ ["k"] ],
            "infection" : [ ["I", "S", "-beta"] ]
        },
        "I" : {
            "linear"    : [ ["I", "-gamma"] ],
            "infection" : [ ["I", "S", "beta"] ]
```

(continues on next page)

(continued from previous page)

```
        }
    }
>>> parameters = {
        'beta': 0.1,
        'gamma': 0.1,
        'seed': 1234,
        'k': 1,
    }
```

model_class_data()

Parameters **data** (*dict*) – The object returned by *simulate*.

Returns

Return type The population of class *model_class_key* as a time series

simulate()

Performs the Stochastic Simulation Algorithm (SSA)

Parameters

- **x0** (*np.array*) – Initial condition.
- **contactMatrix** (*python function(t)*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j
- **Tf** (*float*) – Final time of integrator
- **Nf** (*Int*) – Number of time points to evaluate.
- **method** (*str, optional*) – SSA to use, either 'gillespie' or 'tau_leaping'. The default is 'gillespie'.
- **nc** (*TYPE, optional*) –
- **epsilon** (*float, optional*) – The acceptable relative change of the rates during each tau-leaping step, as defined in Cao et al:

  https://doi.org/10.1063/1.2159468

  The default is 0.03
- **tau_update_frequency** (*TYPE, optional*) –

Returns X: output path from integrator, t : time points evaluated at, 'event_occured' , 'param': input param to integrator.

Return type dict

- Link to example notebook

3.2.2 Spp

class pyross.stochastic.**Spp**

This is a slightly more specific version of the class *Model*.

Spp is still supported for backward compatibility.

Model class is recommended over *Spp* for new users.

The *Spp* class works like *Model* but infection terms use a single class *S*

...

Parameters

- **model_spec** (*dict*) – A dictionary specifying the model. See *Examples*.
- **parameters** (*dict*) – A dictionary containing the model parameters. All parameters can be float if not age-dependent, and np.array(M,) if age-dependent. An optional element with the key 'seed' may be supplied as a seed for the pseudo-random number generator.
- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(3*M,)*) – Initial number in each compartment and class
- **time_dep_param_mapping** (*python function, optional*) – A user-defined function that takes a dictionary of time-independent parameters and time as an argument, and returns a dictionary of the parameters of model_spec. Default: Identical mapping of the dictionary at all times.

Examples

An example of model_spec and parameters for SIR class with a constant influx

```
>>> model_spec = {
        "classes" : ["S", "I"],
        "S" : {
            "constant"  : [ ["k"] ],
            "infection" : [ ["I", "-beta"] ]
        },
        "I" : {
            "linear"    : [ ["I", "-gamma"] ],
            "infection" : [ ["I", "beta"] ]
        }
    }
>>> parameters = {
        'beta': 0.1,
        'gamma': 0.1,
        'seed': 1234,
        'k': 1,
    }
```

- Link to example notebook

3.2.3 SppQ

class pyross.stochastic.**SppQ**

User-defined epidemic model with quarantine stage.

This is a slightly more specific version of the class *Model*.

SppQ is still supported for backward compatibility.

Model class is recommended over *SppQ* for new users.

To initialise the SppQ model,

...

Parameters

- **model_spec** (*dict*) – A dictionary specifying the model. See *Examples*.
- **parameters** (*dict*) – A dictionary containing the model parameters. All parameters can be float if not age-dependent, and np.array(M,) if age-dependent. An optional element with the key 'seed' may be supplied as a seed for the pseudo-random number generator.
- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(3*M,)*) – Initial number in each compartment and class
- **time_dep_param_mapping** (*python function, optional*) – A user-defined function that takes a dictionary of time-independent parameters and time as an argument, and returns a dictionary of the parameters of model_spec. Default: Identical mapping of the dictionary at all times.

Examples

An example of model_spec and parameters for SIR class with random testing (without false positives/negatives) and quarantine

```
>>> model_spec = {
        "classes" : ["S", "I"],
        "S" : {
            "infection" : [ ["I", "-beta"] ]
        },
        "I" : {
            "linear"    : [ ["I", "-gamma"] ],
            "infection" : [ ["I", "beta"] ]
        },
        "test_pos"  : [ "p_falsepos", "p_truepos", "p_falsepos"] ,
        "test_freq" : [ "tf", "tf", "tf"]
    }
>>> parameters = {
        'beta': 0.1,
        'gamma': 0.1,
        'seed': 1234,
        'p_falsepos': 0,
        'p_truepos': 1,
        'tf': 1
    }
```

model_class_data()

Parameters **data** (*dict*) – The object returned by *simulate*.

Returns

Return type The population of class *model_class_key* as a time series

simulate()

Performs the Stochastic Simulation Algorithm (SSA)

Parameters

- **x0** (*np.array*) – Initial condition.
- **contactMatrix** (*python function(t)*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j

- **testRate** (*python function(t)*) – The total number of PCR tests performed per day
- **Tf** (*float*) – Final time of integrator
- **Nf** (*Int*) – Number of time points to evaluate.
- **method** (*str, optional*) – SSA to use, either 'gillespie' or 'tau_leaping'. The default is 'gillespie'.
- **nc** (*TYPE, optional*) –
- **epsilon** (*float, optional*) – The acceptable relative change of the rates during each tau-leaping step, as defined in Cao et al:

  https://doi.org/10.1063/1.2159468

  The default is 0.03
- **tau_update_frequency** (*TYPE, optional*) –

Returns X: output path from integrator, t : time points evaluated at, 'event_occured' , 'param': input param to integrator.

Return type dict

- Link to example notebook

3.2.4 SIR

class pyross.stochastic.**SIR**

Susceptible, Infected, Removed (SIR)

- Ia: asymptomatic
- Is: symptomatic

...

Parameters

- **parameters** (*dict*) – Contains the following keys:

  alpha: float, np.array (M,) fraction of infected who are asymptomatic.

  beta: float rate of spread of infection.

  gIa: float rate of removal from asymptomatic individuals.

  gIs: float rate of removal from symptomatic individuals.

  fsa: float Fraction by which symptomatic individuals do not self-isolate.

  seed: long seed for pseudo-random number generator (optional).
- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(3*M,)*) – Initial number in each compartment and class

rate_matrix:

Calculates the rate constant for each reaction channel.

simulate:

Performs stochastic numerical integration.

simulate()

Performs the Stochastic Simulation Algorithm (SSA)

Parameters

- **S0** (*np.array*) – Initial number of susceptables.
- **Ia0** (*np.array*) – Initial number of asymptomatic infectives.
- **Is0** (*np.array*) – Initial number of symptomatic infectives.
- **contactMatrix** (*python function(t)*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j
- **Tf** (*float*) – Final time of integrator
- **Nf** (*Int*) – Number of time points to evaluate.
- **method** (*str, optional*) – SSA to use, either 'gillespie' or 'tau_leaping'. The default is 'gillespie'.
- **nc** (*TYPE, optional*) –
- **epsilon** (*float, optional*) – The acceptable relative change of the rates during each tau-leaping step, as defined in Cao et al:

  https://doi.org/10.1063/1.2159468.

  The default is 0.03
- **tau_update_frequency** (*TYPE, optional*) –

Returns X: output path from integrator, t : time points evaluated at, 'event_occured' , 'param': input param to integrator.

Return type dict

- Link to example notebook

3.2.5 SIkR

class pyross.stochastic.**SIkR**

Susceptible, Infected, Removed (SIkR). Method of k-stages of I

...

Parameters

- **parameters** (*dict*) – Contains the following keys:

  beta: float rate of spread of infection.

  gI: float rate of removal from infectives.

  fsa: float Fraction by which symptomatic individuals do not self-isolate.

  kI: int number of stages of infection.

  seed: long seed for pseudo-random number generator (optional).
- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array((kI + 1)*M,)*) – Initial number in each compartment and class

- Link to example notebook

3.2.6 SEIR

class pyross.stochastic.**SEIR**
Susceptible, Exposed, Infected, Removed (SEIR)

- Ia: asymptomatic
- Is: symptomatic
- E: exposed

...

Parameters

- **parameters** (*dict*) – Contains the following keys:

  alpha: float, np.array (M,) fraction of infected who are asymptomatic.

  beta: float rate of spread of infection.

  gIa: float rate of removal from asymptomatic individuals.

  gIs: float rate of removal from symptomatic individuals.

  fsa: float Fraction by which symptomatic individuals do not self-isolate.

  gE: float rate of removal from exposed individuals.

  seed: long seed for pseudo-random number generator (optional).
- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(4*M,)*) – Initial number in each compartment and class

- Link to example notebook

3.2.7 SEAIRQ

class pyross.stochastic.**SEAIRQ**
Susceptible, Exposed, Asymptomatic and infected, Infected, Removed, Quarantined (SEAIRQ)

- Ia: asymptomatic
- Is: symptomatic
- E: exposed
- A: asymptomatic and infectious
- Q: quarantined

...

Parameters

- **parameters** (*dict*) – Contains the following keys:

  alpha: float, np.array(M,) fraction of infected who are asymptomatic.

  beta: float rate of spread of infection.

  gIa: float rate of removal from asymptomatic individuals.

  gIs: float rate of removal from symptomatic individuals.

  gE: float rate of removal from exposed individuals.

gA: float rate of removal from activated individuals.

fsa: float Fraction by which symptomatic individuals do not self-isolate.

tE [float] testing rate and contact tracing of exposeds

tA [float] testing rate and contact tracing of activateds

tIa: float testing rate and contact tracing of asymptomatics

tIs: float testing rate and contact tracing of symptomatics

seed: long seed for pseudo-random number generator (optional).

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(6*M,)*) – Initial number in each compartment and class

- Link to example notebook

3.2.8 SEAIRQ_testing

class pyross.stochastic.**SEAIRQ_testing**

Susceptible, Exposed, Asymptomatic and infected, Infected, Removed, Quarantined (SEAIRQ)

- E: exposed
- A: Asymptomatic and infectious
- Ia: asymptomatic
- Is: symptomatic
- Q: quarantined

...

Parameters

- **parameters** (*dict*) – Contains the following keys:

  alpha: float, np.array(M,) fraction of infected who are asymptomatic.

  beta: float rate of spread of infection.

  gIa: float rate of removal from asymptomatic individuals.

  gIs: float rate of removal from symptomatic individuals.

  gE: float rate of removal from exposed individuals.

  gA: float rate of removal from activated individuals.

  fsa: float Fraction by which symptomatic individuals do not self-isolate.

  ars: float fraction of population admissible for random and symptomatic tests

  kapE: float fraction of positive tests for exposed individuals

  seed: long seed for pseudo-random number generator (optional).
- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(6*M,)*) – Initial number in each compartment and class
- **testRate** (*python function(t)*) – number of tests per day and age group

3.2.9 stochastic_integration

class pyross.stochastic.**stochastic_integration**

Integrators used by stochastic models: Gillespie and tau-leaping

A()

Parameters **data** (*Data dict*) –

Returns A

Return type Activated population time series

E()

Parameters **data** (*Data dict*) –

Returns E

Return type Exposed population time series

I()

Parameters **data** (*Data dict*) –

Returns Ia

Return type Asymptomatics population time series

Ia()

Parameters **data** (*Data dict*) –

Returns Ia

Return type Asymptomatics population time series

Is()

Parameters **data** (*Data dict*) –

Returns Is

Return type symptomatics population time series

R()

Parameters **data** (*Data dict*) –

Returns R

Return type Removed population time series

S()

Parameters **data** (*Data dict*) –

Returns S

Return type Susceptible population time series

Sx()

Parameters **data** (*Data dict*) –

Returns

Return type Generic compartment Sx

check_for_event()

simulate_gillespie()

Performs the stochastic simulation using the Gillespie algorithm.

1. Rates for each reaction channel r_i calculated from current state.
2. The timestep tau is chosen randomly from an exponential distribution P ~ e^(-W tau).
3. A single reaction occurs with probablity proportional to its fractional rate constant r_i/W.
4. The state is updated to reflect this reaction occuring and time is propagated forward by tau

Stops if population becomes too small.

Parameters

- **contactMatrix** (*python function(t)*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j
- **Tf** (*float*) – Final time of integrator
- **Nf** (*Int*) – Number of time points to evaluate.

Returns

- **t_arr** (*np.array(Nf,)*) – Array of time points at which the integrator was evaluated.
- **out_arr** (*np.array*) – Output path from integrator.

simulate_tau_leaping()

Tau leaping algorithm for producing stochastically correct trajectories Based on Cao et al (2006): https://doi.org/10.1063/1.2159468 This method can run much faster than the Gillespie algorithm

1. Rates for each reaction channel r_i calculated from current state.
2. Timestep tau chosen such that Delta r_i < epsilon Sum r_i
3. Number of reactions that occur in channel i ~Poisson(r_i tau)
4. Update state by this amount

Parameters

- **contactMatrix** (*python function(t)*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j
- **Tf** (*float*) – Final time of integrator
- **Nf** (*Int*) – Number of time points to evaluate.
- **nc** (*optional*) – The default is 30
- **epsilon** (*float, optional*) – The acceptable relative change of the rates during each tau-leaping step, as defined in Cao et al. The default is 0.03
- **tau_update_frequency** (*optional*) –

Returns

- **t_arr** (*np.array(Nf,)*) – Array of time points at which the integrator was evaluated.
- **out_arr** (*np.array*) – Output path from integrator.

3.3 Hybrid simulations

Hybrid simulation scheme using a combination of stochastic and deterministic schemes.

3.3.1 SIR

class pyross.hybrid.**SIR**

Susceptible, Infected, Removed (SIR) Ia: asymptomatic Is: symptomatic

...

Parameters

- **parameters** (*dict*) –

  Contains the following keys:

  alpha: float, np.array (M,) fraction of infected who are asymptomatic.

  beta: float rate of spread of infection.

  gIa: float rate of removal from asymptomatic individuals.

  gIs: float rate of removal from symptomatic individuals.

  fsa: float fraction by which symptomatic individuals do not self-isolate.

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(3*M,)*) – Initial number in each compartment and class

simulate()

3.4 Bayesian inference

Inference for age structured compartment models using the diffusion approximation (via the van Kampen expansion). See this paper for more details on the method.

There are two ways to do inference: manifold method (sec 3.3 in the report) and tangent space method (sec 3.4 in the report). In various degrees of *less robust but fast* to *more robust but slow*:

- tangent space method.
- manifold method with few internal steps and fast integration method (*det_method* = *RK2*, *lyapunov_method* = *euler*).
- manifold method with large number of internel steps and robust integration method (*solve_ivp* from scipy library).

Methods for full data	
infer	Infers epidemiological and control parameters given all information.
infer_mcmc	Explore the posterior distribution given all information.
infer_nested_sampling	Compute the model evidence (and generate posterior samples) given all information.
obtain_minus_log_p	Computes -log(p) of a fully observed trajectory.
compute_hessian	Computes the Hessian of -log(p).
nested_sampling_inference	Compute the log-evidence and weighted samples.

Methods for partial data	
latent_infer	Infers epidemiological and control parameters and initial conditions.
latent_infer_mcmc	Explore the posterior distribution.
latent_infer_nested_sampling	Compute the model evidence (and generate posterior samples).
minus_logp_red	Computes -log(p) of a partially observed trajectory.
compute_hessian_latent	Computes the Hessian of -log(p).
nested_sampling_latent_inference	Compute the log-evidence and weighted samples.

Sensitivity analysis	
FIM	Computes the Fisher Information Matrix of the stochastic model.
FIM_det	Computes the Fisher Information Matrix of the deterministic model.
sensitivity	Computes the normalized sensitivity measure

Helper function	
integrate	A wrapper around 'simulate' in pyross.deterministic.
set_params	Sets parameters.
set_det_method	Sets the integration method of the deterministic equation
set_lyapunov_method	Sets the integration method of the Lyapunov equation
set_det_model	Sets the internal deterministic model
set_contact_matrix	Sets the contact matrix
fill_params_dict	Fills and returns a parameter dictionary
get_mean_inits	Constructs full initial conditions from the prior dict

The functions are documented under the parent class *SIR_type*.

3.4.1 SIR_type

class pyross.inference.**SIR_type**

Parent class for inference for all SIR-type classes listed below

All subclasses use the same functions to perform inference, which are documented below.

FIM()

Computes the Fisher Information Matrix (FIM) for the MAP estimates of a stochastic SIR type model.

Parameters

- **x** (*2d numpy.array*) – Observed trajectory (number of data points x (age groups * model classes))
- **Tf** (*float*) – Total time of the trajectory
- **infer_result** (*dict*) – Dictionary returned by latent_infer
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.
- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.
- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the

function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays. The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.

- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is False.
- **eps** (*float or numpy.array, optional*) – Step size for numerical differentiation of the process mean and its full covariance matrix with respect to the parameters. Must be either a scalar, or an array of length *len(infer_result['flat_params'])*. If not specified,

```
eps = 100*infer_result['flat_params']
      *numpy.divide(numpy.spacing(infer_result['log_likelihood']),
      infer_result['log_likelihood'])**(0.25)
```

is used. It is recommended to use a step-size greater or equal to *eps*. Decreasing the step size too small can result in round-off error.

- **inter_steps** (*int, optional*) – Intermediate steps for interpolation between observations for the deterministic forward Euler integration. A higher number of intermediate steps will improve the accuracy of the result, but will make computations slower. Setting *inter_steps=0* will fall back to the method accessible via *det_method* for the deterministic integration. We have found that forward Euler is generally slower, but more stable for derivatives with respect to parameters than the variable step size integrators used elsewhere in pyross. Default is 100.

Returns **FIM** – The Fisher Information Matrix

Return type 2d numpy.array

FIM_det()

Computes the Fisher Information Matrix (FIM) for the MAP estimates of a deterministic (ODE based, including a constant measurement error) SIR type model.

Parameters

- **x** (*2d numpy.array*) – Observed trajectory (number of data points x (age groups * model classes))
- **Tf** (*float*) – Total time of the trajectory
- **infer_result** (*dict*) – Dictionary returned by latent_infer
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.
- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.
- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays. The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.
- **eps** (*float or numpy.array, optional*) – Step size for numerical differentiation of the process mean and its full covariance matrix with respect to the parameters. Must

be either a scalar, or an array of length *len(infer_result['flat_params'])*. If not specified,

```
eps = 100*infer_result['flat_params']
      *numpy.divide(numpy.spacing(infer_result['log_likelihood']),
      infer_result['log_likelihood'])**(0.25)
```

is used. It is recommended to use a step-size greater or equal to *eps*. Decreasing the step size too small can result in round-off error.

- **measurement_error** (*float, optional*) – Standard deviation of measurements (uniform and independent Gaussian measurement error assumed). Default is 1e-2.
- **inter_steps** (*int, optional*) – Intermediate steps for interpolation between observations for the deterministic forward Euler integration. A higher number of intermediate steps will improve the accuracy of the result, but will make computations slower. Setting *inter_steps=0* will fall back to the method accessible via *det_method* for the deterministic integration. We have found that forward Euler is generally slower, but more stable for derivatives with respect to parameters than the variable step size integrators used elsewhere in pyross. Default is 100.

Returns **FIM** – The Fisher Information Matrix

Return type 2d numpy.array

evidence_laplace()

Compute the evidence using a Laplace approximation at the MAP estimate.

Parameters

- **x** (*2d numpy.array*) – Observed trajectory (number of data points x (age groups * model classes))
- **Tf** (*float*) – Total time of the trajectory
- **infer_result** (*dict*) – Dictionary returned by latent_infer
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.
- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.
- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays. The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.
- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is False.
- **eps** (*float or numpy.array, optional*) – Step size for finite differences computation of the hessian with respect to the parameters. Must be either a scalar, or an array of length *len(infer_result['flat_params'])*. If not specified,

```
eps = 100*infer_result['flat_params']
      *numpy.divide(numpy.spacing(infer_result['log_likelihood']),
      infer_result['log_likelihood'])**(0.25)
```

is used. For *fd_method="central"* it is recommended to use a step-size greater or equal to *eps*. Decreasing the step size too small can result in round-off error.

- **fd_method** (*str, optional*) – The type of finite-difference scheme used to compute the hessian, supports "forward" and "central". Default is "central".
- **inter_steps** (*int, optional*) – Only used if *tangent=False*. Intermediate steps for interpolation between observations for the deterministic forward Euler integration. A higher number of intermediate steps will improve the accuracy of the result, but will make computations slower. Setting *inter_steps=0* will fall back to the method accessible via *det_method* for the deterministic integration. We have found that forward Euler is generally slower, but more stable for derivatives with respect to parameters than the variable step size integrators used elsewhere in pyross. Default is 10.

Returns **log_evidence** – The log-evidence computed via Laplace approximation at the MAP estimate.

Return type float

fill_params_dict()

Returns a full dictionary for epidemiological parameters with some changed values

Parameters

- **keys** (*list of String*) – A list of names of parameters to be changed.
- **params** (*numpy.array of list*) – An array of the same size as keys for the updated value.
- **return_additional_params** (*boolean, optional (default = False)*) – Handling of parameters that are not model parameters (e.g. control parameters). False: raise exception, True: return second dictionary with other parameters

Returns **full_parameters** – A dictionary of epidemiological parameters. For parameter names specified in *keys*, set the values to be the ones in *params*; for the others, use the values stored in the class.

Return type dict

get_mean_inits()

Construct full initial conditions from the prior dict

Parameters

- **init_priors** (*dict*) – A dictionary for priors for initial conditions. Same as the *init_priors* passed to *latent_infer*. In this function, only takes the mean.
- **obs0** (*numpy.array*) – Observed initial conditions.
- **fltr0** (*numpy.array*) – Filter for the observed initial conditions.

Returns **x0** – Full initial conditions.

Return type numpy.array

hessian()

Computes the Hessian matrix for the MAP estimates of an SIR type model.

Parameters

- **x** (*2d numpy.array*) – Observed trajectory (number of data points x (age groups * model classes))
- **Tf** (*float*) – Total time of the trajectory

- **infer_result** (*dict*) – Dictionary returned by latent_infer
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.
- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.
- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays. The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.
- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is False.
- **eps** (*float or numpy.array, optional*) – Step size for finite differences computation of the hessian with respect to the parameters. Must be either a scalar, or an array of length *len(infer_result['flat_params'])*. If not specified,

```
eps = 100*infer_result['flat_params']
      *numpy.divide(numpy.spacing(infer_result['log_likelihood']),
      infer_result['log_likelihood'])**(0.25)
```

  is used. For *fd_method="central"* it is recommended to use a step-size greater or equal to *eps*. Decreasing the step size too small can result in round-off error.
- **fd_method** (*str, optional*) – The type of finite-difference scheme used to compute the hessian, supports "forward" and "central". Default is "central".
- **inter_steps** (*int, optional*) – Only used if *tangent=False*. Intermediate steps for interpolation between observations for the deterministic forward Euler integration. A higher number of intermediate steps will improve the accuracy of the result, but will make computations slower. Setting *inter_steps=0* will fall back to the method accessible via *det_method* for the deterministic integration. We have found that forward Euler is generally slower, but sometimes more stable for derivatives with respect to parameters than the variable step size integrators used elsewhere in pyross. Default is 0.

Returns **hess** – The Hessian matrix

Return type 2d numpy.array

infer()

Compute the maximum a-posteriori (MAP) estimate for all desired parameters, including control parameters, for an SIR type model with fully observed classes. If *generator* is specified, the lockdown is modelled by scaling the contact matrices for contact at work, school, and other (but not home). This function infers the scaling parameters (can be age dependent) assuming that full data on all classes is available (with latent variables, use *latent_infer*).

Parameters

- **x** (*2d numpy.array*) – Observed trajectory (number of data points x (age groups * model classes))
- **Tf** (*float*) – Total time of the trajectory

- **prior_dict** (*dict*) – A dictionary containing priors for parameters (can include both model and intervention parameters). See examples.
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.
- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.
- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays. The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.
- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is false.
- **verbose** (*bool, optional*) – Set to True to see intermediate outputs from the optimizer.
- **ftol** (*double*) – Relative tolerance of logp
- **global_max_iter** (*int, optional*) – Number of global optimisations performed.
- **local_max_iter** (*int, optional*) – Number of local optimisation performed.
- **global_atol** (*float*) – The absolute tolerance for global optimisation.
- **enable_global** (*bool, optional*) – Set to True to enable global optimisation.
- **enable_local** (*bool, optional*) – Set to True to enable local optimisation.
- **cma_processes** (*int, optional*) – Number of parallel processes used for global optimisation.
- **cma_population** (*int, optional*) – The number of samples used in each step of the CMA algorithm.
- **cma_random_seed** (*int (between 0 and 2**32-1)*) – Random seed for the optimisation algorithms. By default it is generated from numpy.random.randint.

Returns

output_dict – Dictionary of MAP estimates, containing the following keys for users:

params_dict: dict Dictionary for MAP estimates of the model parameters.

control_params_dict: dict Dictionary for MAP estimates of the control parameters (if requested).

-logp: float Value of -logp at MAP.

Return type dict

Note: This function combines the functionality of *infer_parameters* and *infer_control*, which will be deprecated. To infer model parameters only, specify a fixed *contactMatrix* function. To infer control parameters only, specify a *generator* and do not specify priors for model parameters.

Examples

An example of prior_dict to set priors for alpha and beta, where alpha is age dependent and we want to infer its scale parameters rather than each component individually. The prior distribution is assumed to be log-normal with the specified mean and standard deviation.

```
>>> prior_dict = {
        'alpha':{
            'mean': [0.5, 0.2],
            'infer_scale': True,
            'scale_factor_std': 1,
            'scale_factor_bounds': [0.1, 10],
            'prior_fun': 'truncnorm'
        },
        'beta':{
            'mean': 0.02,
            'std': 0.1,
            'bounds': [1e-4, 1],
            'prior_fun': 'lognorm'
        }
    }
```

infer_control()

Compute the maximum a-posteriori (MAP) estimate of the change of control parameters for a SIR type model in lockdown.

Parameters **infer** (*see*) –

Returns

output_dict – Dictionary of MAP estimates, containing the following keys for users:

map_dict: dict Dictionary for MAP estimates of the control parameters.

-logp: float Value of -logp at MAP.

Return type dict

Note: This function just calls *infer* with the specified generator, will be deprecated.

---

infer_mcmc()

Sample the posterior distribution of the epidemiological parameters using ensemble MCMC.

Parameters

- **x** (*2d numpy.array*) – Observed trajectory (number of data points x (age groups * model classes))
- **Tf** (*float*) – Total time of the trajectory
- **prior_dict** (*dict*) – A dictionary containing priors for parameters (can include both model and intervention parameters). See examples.
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.
- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.

- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays. The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.
- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is False.
- **verbose** (*bool, optional*) – Set to True to see a progress bar for the sample generation. Default is False.
- **sampler** (*emcee.EnsembleSampler, optional*) – Set to instance of the sampler (as returned by this function) to continue running the MCMC chains. Default is None (i.e. run a new chain).
- **nwalkers** (*int, optional*) – The number of chains in the ensemble (should be at least 2*dim). Default is 2*dim.
- **walker_pos** (*np.array, optional*) – The initial position of the walkers. If not specified, it samples random positions from the prior.
- **nsamples** (*int, optional*) – The number of samples per walker. Default is 1000.
- **nprocesses** (*int, optional*) – The number of processes used to compute the likelihood for the walkers, needs *pathos*. Default is the number of cpu cores if *pathos* is available, otherwise 1.

Returns **sampler** – This function returns the interal state of the sampler. To look at the chain of the internal flattened parameters, run *sampler.get_chain()*. Use this to judge whether the chain has sufficiently converged. Either rerun *mcmc_inference(..., sampler=sampler)* to continue the chain or *mcmc_inference_process_result(...)* to process the result.

Return type emcee.EnsembleSampler

Examples

For the structure of *prior_dict*, see the documentation of *infer*. To start sampling the posterior, run for example

```
>>> sampler = estimator.infer_mcmc(x, Tf, prior_dict,␣
↪contactMatrix=contactMatrix, verbose=True)
```

To judge the convergence of this chain, we can look at the trace plot of all the chains (for a moderate number of dimensions *dim*)

```
>>> fig, axes = plt.subplots(dim, sharex=True)
>>> samples = sampler.get_chain()
>>> for i in range(dim):
        ax = axes[i]
        ax.plot(samples[:, :, i], "k", alpha=0.3)
        ax.set_xlim(0, len(samples))
>>> axes[-1].set_xlabel("step number");
```

For more detailed convergence metrics, see the documentation of *emcee*. To continue running this chain, we can call this function again with the sampler as argument

```
>>> sampler = estimator.infer_mcmc(x, Tf, prior_dict,␣
↪contactMatrix=contactMatrix, verbose=True, sampler=sampler)
```

This procudes 1000 additional samples in each chain. To process the results, call *infer_mcmc_process_result*.

infer_mcmc_process_result()

Take the sampler generated by *pyross.inference.infer_mcmc* and produce output dictionaries for further use in the pyross framework. See *pyross.inference.infer_mcmc* for additional description of parameters.

Parameters

- **sampler** (*emcee.EnsembleSampler*) – Output of *pyross.inference.infer_mcmc*.
- **prior_dict** (*dict*) –
- **contactMatrix** (*callable, optional*) –
- **generator** (*pyross.contactMatrix, optional*) –
- **intervention_fun** (*callable, optional*) –
- **flat** (*bool, optional*) – This decides whether to return the samples as for each chain separately (False) or as as a combined list (True). Default is True.
- **discard** (*int, optional*) – The number of initial samples to discard in each chain (to account for burn-in). Default is 0.
- **thin** (*int, optional*) – Thin out the chain by taking only the n-tn element in each chain. Default is 1 (no thinning).
- ****catchall_kwargs** (*dict*) – Catched further provided arguments and ignores them.

Returns **output_samples** – The processed posterior samples.

Return type list of dict (if flat=True), or list of list of dict (if flat=False)

infer_nested_sampling()

Compute the log-evidence and weighted samples of the a-posteriori distribution of the parameters of a SIR type model using nested sampling as implemented in the *dynesty* Python package. This function assumes that full data on all classes is available.

Parameters

- **x** (*2d numpy.array*) – Observed trajectory (number of data points x (age groups * model classes))
- **Tf** (*float*) – Total time of the trajectory
- **prior_dict** (*dict*) – A dictionary containing priors for parameters (can include both model and intervention parameters). See examples.
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.
- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.
- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays.

The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.

- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is false.
- **verbose** (*bool, optional*) – Set to True to see intermediate outputs from the nested sampling procedure.
- **nprocesses** (*int, optional*) – The number of processes used for parallel evaluation of the likelihood.
- **queue_size** (*int, optional*) – Size of internal queue of likelihood values, default is nprocesses if multiprocessing is used.
- **maxiter** (*int, optional*) – The maximum number of iterations. Default is no limit.
- **maxcall** (*int, optional*) – The maximum number of calls to the likelihood function. Default no limit.
- **dlogz** (*float, optional*) – The iteration terminates if the estimated contribution of the remaining prior volume to the total evidence falls below this threshold. Default value is *1e-3 * (nlive - 1) + 0.01* if *add_live==True*, 0.01 otherwise.
- **n_effective** (*float, optional*) – The iteration terminates if the number of effective posterior samples reaches this values. Default is no limit.
- **add_live** (*bool, optional*) – Determines whether to add the remaining set of live points to the set of samples. Default is True.
- **sampler** (*dynesty.NestedSampler, optional*) – Continue running an instance of a nested sampler until the termination criteria are met.
- ****dynesty_args** – Arguments passed through to the construction of the dynesty.NestedSampler constructor. Relevant entries are (this is not comprehensive, for details see the documentation of dynesty):

  nlive: int, optional The number of live points. Default is 500.

  bound: {'none', 'single', 'multi', 'balls', 'cubes'}, optional Method used to approximately bound the prior using the current set of live points. Default is 'multi'.

  sample: {'auto', 'unif', 'rwalk', 'rstagger', 'slice', 'rslice', 'hslice', callable}, optional Method used to sample uniformly within the likelihood constraint, conditioned on the provided bounds.

Returns **sampler** – The state of the sampler after termination of the nested sampling run.

Return type dynesty.NestedSampler

infer_nested_sampling_process_result()

Take the sampler generated by *pyross.inference.infer_nested_sampling* and produce output dictionaries for further use in the pyross framework. See *pyross.inference.infer_nested_sampling* for description of parameters.

Parameters

- **sampler** (*dynesty.NestedSampler*) – The output of *pyross.inference.infer_nested_sampling*.
- **prior_dict** (*dict*) –
- **contactMatrix** (*callable, optional*) –

- **generator** (*pyross.contactMatrix, optional*) –
- **intervention_fun** (*callable, optional*) –
- ****catchall_kwargs** (*dict*) – Catched further provided arguments and ignores them.

Returns

- **result** (*dynesty.Result*) – The result of the nested sampling iteration. Relevant entries include:

  result.logz: list The progression of log-evidence estimates, use result.logz[-1] for the final estimate.

- **output_samples** (*list*) – The processed weighted posterior samples.

infer_parameters()

Infers the MAP estimates for epidemiological parameters

Parameters **infer** (*see*) –

Returns

output – Contains the following keys for users:

map_dict: dict A dictionary for MAPs. Keys are the names of the parameters and the corresponding values are its MAP estimates.

-logp: float The value of -logp at MAP.

Return type dict

Note: This function just calls *infer* with a fixed contactMatrix function, will be deprecated.

---

integrate()

An light weight integrate method similar to *simulate* in pyross.deterministic

Parameters

- **x0** (*np.array*) – Initial state of the given model
- **t1** (*float*) – Initial time of integrator
- **t2** (*float*) – Final time of integrator
- **steps** (*int*) – Number of time steps for numerical integrator evaluation.
- **max_step** (*int, optional*) – The maximum allowed step size of the integrator.

Returns sol – The state of the system evaulated at the time point specified. Only used if det_method is set to 'solve_ivp'.

Return type np.array

latent_FIM()

Computes the Fisher Information Matrix (FIM) of the stochastic model for the initial conditions and all desired parameters, including control parameters, for a SIR type model with partially observed classes. The unobserved classes are treated as latent variables.

Parameters

- **obs** (*np.array*) – The partially observed trajectory.
- **fltr** (*2d np.array*) – The filter for the observation such that $F_{ij}x_j(t) = obs_i(t)$
- **Tf** (*float*) – Total time of the trajectory

- **infer_result** (*dict*) – Dictionary returned by latent_infer
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.
- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.
- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays. The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.
- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is False.
- **eps** (*float or numpy.array, optional*) – Step size for numerical differentiation of the process mean and its full covariance matrix with respect to the parameters. Must be either a scalar, or an array of length *len(infer_result['flat_params'])*. If not specified,

```
eps = 100*infer_result['flat_params']
      *numpy.divide(numpy.spacing(infer_result['log_likelihood']),
      infer_result['log_likelihood'])**(0.25)
```

  is used. It is recommended to use a step-size greater or equal to *eps*. Decreasing the step size too small can result in round-off error.
- **inter_steps** (*int, optional*) – Intermediate steps between observations for the deterministic forward Euler integration. A higher number of intermediate steps will improve the accuracy of the result, but will make computations slower. Setting *inter_steps=0* will fall back to the method accessible via *det_method* for the deterministic integration. We have found that forward Euler is generally slower, but more stable for derivatives with respect to parameters than the variable step size integrators used elsewhere in pyross. Default is 100.

Returns **FIM** – The Fisher Information Matrix

Return type 2d numpy.array

latent_FIM_det()

Computes the Fisher Information Matrix (FIM) of the deterministic model (ODE based, including a constant measurement error) for the initial conditions and all desired parameters, including control parameters, for a SIR type model with partially observed classes. The unobserved classes are treated as latent variables.

Parameters

- **obs** (*np.array*) – The partially observed trajectory.
- **fltr** (*2d np.array*) – The filter for the observation such that $F_{ij} x_j(t) = obs_i(t)$
- **Tf** (*float*) – Total time of the trajectory
- **infer_result** (*dict*) – Dictionary returned by latent_infer
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.

- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.
- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays. The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.
- **eps** (*float or numpy.array, optional*) – Step size for numerical differentiation of the process mean and its full covariance matrix with respect to the parameters. Must be either a scalar, or an array of length *len(infer_result['flat_params'])*. If not specified,

```
eps = 100*infer_result['flat_params']
      *numpy.divide(numpy.spacing(infer_result['log_likelihood']),
      infer_result['log_likelihood'])**(0.25)
```

  is used. It is recommended to use a step-size greater or equal to *eps*. Decreasing the step size too small can result in round-off error.
- **measurement_error** (*float, optional*) – Standard deviation of measurements (uniform and independent Gaussian measurement error assumed). Default is 1e-2.
- **inter_steps** (*int, optional*) – Intermediate steps between observations for the deterministic forward Euler integration. A higher number of intermediate steps will improve the accuracy of the result, but will make computations slower. Setting inter_steps=0 will fall back to the method accessible via det_method for the deterministic integration. We have found that forward Euler is generally slower, but more stable for derivatives with respect to parameters than the variable step size integrators used elsewhere in pyross. Default is 100.

Returns **FIM_det** – The Fisher Information Matrix

Return type 2d numpy.array

latent_evidence_laplace()

Compute the evidence using a Laplace approximation at the MAP estimate for a SIR type model with partially observed classes. The unobserved classes are treated as latent variables.

Parameters

- **obs** (*np.array*) – The partially observed trajectory.
- **fltr** (*2d np.array*) – The filter for the observation such that $F_{ij} x_j(t) = obs_i(t)$
- **Tf** (*float*) – Total time of the trajectory
- **infer_result** (*dict*) – Dictionary returned by latent_infer
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.
- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.
- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays.

The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.

- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is False.
- **eps** (*float or numpy.array, optional*) – Step size for finite differences computation of the hessian with respect to the parameters. Must be either a scalar, or an array of length *len(infer_result['flat_params'])*. If not specified,

```
eps = 100*infer_result['flat_params']
      *numpy.divide(numpy.spacing(infer_result['log_likelihood']),
      infer_result['log_likelihood'])**(0.25)
```

is used. For *fd_method="central"* it is recommended to use a step-size greater or equal to *eps*. Decreasing the step size too small can result in round-off error.

- **fd_method** (*str, optional*) – The type of finite-difference scheme used to compute the hessian, supports "forward" and "central". Default is "central".
- **inter_steps** (*int, optional*) – Intermediate steps between observations for the deterministic forward Euler integration. A higher number of intermediate steps will improve the accuracy of the result, but will make computations slower. Setting *inter_steps=0* will fall back to the method accessible via *det_method* for the deterministic integration. We have found that forward Euler is generally slower, but more stable for derivatives with respect to parameters than the variable step size integrators used elsewhere in pyross. Default is 100.

Returns **log_evidence** – The log-evidence computed via Laplace approximation at the MAP estimate.

Return type float

latent_hessian()

Computes the Hessian matrix for the initial conditions and all desired parameters, including control parameters, for a SIR type model with partially observed classes. The unobserved classes are treated as latent variables.

Parameters

- **obs** (*np.array*) – The partially observed trajectory.
- **fltr** (*2d np.array*) – The filter for the observation such that $F_{ij} x_j(t) = obs_i(t)$
- **Tf** (*float*) – Total time of the trajectory
- **infer_result** (*dict*) – Dictionary returned by latent_infer
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.
- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.
- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays.

The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.

- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is False.
- **eps** (*float or numpy.array, optional*) – Step size for finite differences computation of the hessian with respect to the parameters. Must be either a scalar, or an array of length *len(infer_result['flat_params'])*. If not specified,

```
eps = 100*infer_result['flat_params']
      *numpy.divide(numpy.spacing(infer_result['log_likelihood
↪']),
      infer_result['log_likelihood'])**(0.25)
```

is used. For *fd_method="central"* it is recommended to use a step-size greater or equal to *eps*. Decreasing the step size too small can result in round-off error.

- **fd_method** (*str, optional*) – The type of finite-difference scheme used to compute the hessian, supports "forward" and "central". Default is "central".
- **inter_steps** (*int, optional*) – Intermediate steps between observations for the deterministic forward Euler integration. A higher number of intermediate steps will improve the accuracy of the result, but will make computations slower. Setting *inter_steps=0* will fall back to the method accessible via *det_method* for the deterministic integration. We have found that forward Euler is generally slower, but sometimes more stable for derivatives with respect to parameters than the variable step size integrators used elsewhere in pyross. Default is 0.

Returns **hess** – The Hessian matrix

Return type 2d numpy.array

latent_infer()

Compute the maximum a-posteriori (MAP) estimate for the initial conditions and all desired parameters, including control parameters, for a SIR type model with partially observed classes. The unobserved classes are treated as latent variables.

Parameters

- **obs** (*np.array*) – The partially observed trajectory.
- **fltr** (*2d np.array*) – The filter for the observation such that $F_{ij} x_j(t) = obs_i(t)$
- **Tf** (*float*) – Total time of the trajectory
- **param_priors** (*dict*) – A dictionary that specifies priors for parameters (including control parameters, if desired). See *infer* for further explanations.
- **init_priors** (*dict*) – A dictionary for priors for initial conditions. See below for examples
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.
- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.

- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays. The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.
- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is false.
- **verbose** (*bool, optional*) – Set to True to see intermediate outputs from the optimizer.
- **ftol** (*double*) – Relative tolerance of logp
- **global_max_iter** (*int, optional*) – Number of global optimisations performed.
- **local_max_iter** (*int, optional*) – Number of local optimisation performed.
- **local_initital_step** (*optional, float or np.array*) – Initial step size for the local optimiser. If scalar, relative to the initial guess. Default: Deterined by final state of global optimiser, or, if enable_global=False, 0.01
- **global_atol** (*float*) – The absolute tolerance for global minimisation.
- **enable_global** (*bool, optional*) – Set to True to enable global optimisation.
- **enable_local** (*bool, optional*) – Set to True to enable local optimisation.
- **cma_processes** (*int, optional*) – Number of parallel processes used for global optimisation.
- **cma_population** (*int, optional*) – The number of samples used in each step of the CMA algorithm.
- **cma_random_seed** (*int (between 0 and 2**32-1)*) – Random seed for the optimisation algorithms. By default it is generated from numpy.random.randint.
- **objective** (*string, optional*) – Objective for the minimisation. 'likelihood' (default), 'least_square' (least squares fit w.r.t. absolute compartment values), 'least_squares_diff' (least squares fit w.r.t. time-differences of compartment values)
- **alternative_guess** (*np.array, optional*) – Alternative initial quess, different form the mean of the prior. Array in the same format as 'flat_params' in the result dictionary of a previous optimisation run.
- **use_mode_as_guess** (*bool, optional*) – Initialise optimisation with mode instead of mean of the prior. Makes a difference for lognormal distributions.
- **tmp_file** (*optional, string*) – If specified, name of a file to store the temporary best estimate of the global optimiser (as backup or for inspection) as numpy array file
- **load_backup_file** (*optional, string*) – If specified, name of a file to restore the the state of the global optimiser

Returns

output_dict – A dictionary containing the following keys for users:

x0: np.array MAP estimates for the initial conditions

params_dict: dict dictionary for MAP estimates for model parameters

control_params_dict: dict dictionary for MAP estimates for control parameters (if requested)

-logp: float Value of -logp at MAP.

Return type dict

Note: This function combines the functionality of *latent_infer_parameters* and *latent_infer_control*, which will be deprecated. To infer model parameters only, specify a fixed *contactMatrix* function. To infer control parameters only, specify a *generator* and do not specify priors for model parameters.

Examples

Here we list three examples, one for inferring all initial conditions along the fastest growing linear mode, one for inferring the initial conditions individually and a mixed one.

First, suppose we only observe Is out of (S, Ia, Is) and we wish to infer all compartmental values of S and Ia independently. For two age groups with population [2500, 7500],

```
>>> init_priors = {
        'independent':{
            'fltr': [True, True, True, True, False, False],
            'mean': [2400, 7400, 50, 50],
            'std': [200, 200, 200, 200],
            'bounds': [[2000, 2500], [7000, 7500], [0, 400], [0, 400]]
        }
    }
```

In the 'fltr' entry, we need a boolean array indicating which components of the full x0 = [S0[0], S0[1], Ia0[0], Ia0[1], Is0[0], Ia0[1]] array we are inferring. By setting fltr = [True, True, True, True, False, False], the inference algorithm will know that we are inferring all components of S0 and Ia0 but not Is0. Similar to inference for parameter values, we also assume a log-normal distribution for the priors for the initial conditions.

Next, if we are happy to assume that all our initial conditions lie along the fastest growing linear mode and we will only infer the coefficient of the mode, the init_priors dict would be,

```
>>> init_priors = {
        'lin_mode_coeff':{
            'fltr': [True, True, True, True, False, False],
            'mean': 100,
            'std': 100,
            'bounds': [1, 1000]
        }
    }
```

Note that the 'fltr' entry is still the same as before because we still only want to infer S and Ia, and the initial conditions for Is is fixed by the observation.

Finally, if we want to do a mixture of both (useful when some compartments have aligned with the fastest growing mode but others haven't), we need to set the init_priors to be,

```
>>> init_priors = {
        'lin_mode_coeff': {
            'fltr': [True, True, False, False, False, False],
            'mean': 100,
            'std': 100,
            'bounds': [1, 1000]
```

(continues on next page)

(continued from previous page)

```
        },
        'independent':{
            'fltr': [False, False, True, True, False, False],
            'mean': [50, 50],
            'std': [200, 200],
            'bounds': [0, 400], [0, 400]
        }
    }
```

latent_infer_control()

Compute the maximum a-posteriori (MAP) estimate of the change of control parameters for a SIR type model in lockdown with partially observed classes.

Parameters **latent_infer** (*see*) –

Returns

output_dict – A dictionary containing the following keys for users:

map_params_dict: dict dictionary for MAP estimates for control parameters

map_x0: np.array MAP estimates for the initial conditions

-logp: float Value of -logp at MAP.

Return type dict

Note: This function just calls *latent_infer* (with the specified *generator*), will be deprecated.

---

latent_infer_mcmc()

Sample the posterior distribution of the initial conditions and all desired parameters, including control parameters, using ensemble MCMC. This requires the optional dependency *emcee*.

Parameters

- **obs** (*2d numpy.array*) – The observed trajectories with reduced number of variables (number of data points, (age groups * observed model classes))
- **fltr** (*2d numpy.array*) – A matrix of shape (no. observed variables, no. total variables), such that obs_{ti} = fltr_{ij} * X_{tj}
- **Tf** (*float*) – Total time of the trajectory
- **contactMatrix** (*callable*) – A function that returns the contact matrix at time t (input).
- **param_priors** (*dict*) – A dictionary for priors for the model parameters. See *latent_infer_parameters* for further explanations.
- **init_priors** (*dict*) – A dictionary for priors for the initial conditions. See *latent_infer_parameters* for further explanations.
- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is False.
- **verbose** (*bool, optional*) – Set to True to see a progress bar for the sample generation. Default is False.
- **sampler** (*emcee.EnsembleSampler, optional*) – Set to instance of the sampler (as returned by this function) to continue running the MCMC chains. Default is None (i.e. run a new chain).

- **nwalkers** (*int, optional*) – The number of chains in the ensemble (should be at least 2*dim). Default is 2*dim.
- **walker_pos** (*np.array, optional*) – The initial position of the walkers. If not specified, the function samples random positions from the prior.
- **nsamples** (*int, optional*) – The number of samples per walker. Default is 1000.
- **nprocesses** (*int, optional*) – The number of processes used to compute the likelihood for the walkers, needs *pathos* for values > 1. Default is the number of cpu cores if *pathos* is available, otherwise 1.

Returns **sampler** – This function returns the state of the sampler. To look at the chain of the internal flattened parameters, run *sampler.get_chain()*. Use this to judge whether the chain has sufficiently converged. Either rerun *latent_infer_mcmc(..., sampler=sampler)* to continue the chain or *latent_infer_mcmc_process_result(...)* to process the result.

Return type emcee.EnsembleSampler

Examples

For the structure of the model input parameters, in particular *param_priors, init_priors*, see the documentation of *latent_infer*. To start sampling the posterior, run for example

```
>>> sampler = estimator.latent_infer_mcmc(obs, fltr, Tf, param_priors, init_
↪priors, contactMatrix=contactMatrix,
                                          verbose=True)
```

To judge the convergence of this chain, we can look at the trace plot of all the chains (for a moderate number of dimensions *dim*)

```
>>> fig, axes = plt.subplots(dim, sharex=True)
>>> samples = sampler.get_chain()
>>> for i in range(dim):
        ax = axes[i]
        ax.plot(samples[:, :, i], "k", alpha=0.3)
        ax.set_xlim(0, len(samples))
>>> axes[-1].set_xlabel("step number");
```

For more detailed convergence metrics, see the documentation of *emcee*. To continue running this chain, we can call this function again with the sampler as argument

```
>>> sampler = estimator.latent_infer_mcmc(obs, fltr, Tf, param_priors, init_
↪priors, contactMatrix=contactMatrix,
                                          verbose=True, sampler=sampler)
```

This procudes 1000 additional samples in each chain. To process the results, call *pyross.inference.latent_infer_mcmc_process_result*.

latent_infer_mcmc_process_result()

Take the sampler generated by *pyross.inference.latent_infer_mcmc* and produce output dictionaries for further use in the pyross framework.

Parameters

- **sampler** (*emcee.EnsembleSampler*) – Output of *mcmc_latent_inference*.
- **obs** (*2d numpy.array*) –
- **fltr** (*2d numpy.array*) –

- **param_priors** (*dict*) –
- **init_priors** (*dict*) –
- **contactMatrix** (*callable, optional*) –
- **generator** (*pyross.contactMatrix, optional*) –
- **intervention_fun** (*callable, optional*) –
- **flat** (*bool, optional*) – This decides whether to return the samples as for each chain separately (False) or as as a combined list (True). Default is True.
- **discard** (*int, optional*) – The number of initial samples to discard in each chain (to account for burn-in). Default is 0.
- **thin** (*int, optional*) – Thin out the chain by taking only the n-tn element in each chain. Default is 1 (no thinning).
- ****catchall_kwargs** (*dict*) – Catches further provided arguments and ignores them.

Returns **output_samples** – The processed posterior samples.

Return type list of dict (if flat=True), or list of list of dict (if flat=False)

latent_infer_nested_sampling()

Compute the log-evidence and weighted samples for the initial conditions and all desired parameters, including control parameters, for a SIR type model with partially observed classes. This function uses nested sampling as implemented in the *dynesty* Python package.

Parameters

- **obs** (*2d numpy.array*) – The observed trajectories with reduced number of variables (number of data points, (age groups * observed model classes))
- **fltr** (*2d numpy.array*) – A matrix of shape (no. observed variables, no. total variables), such that obs_{ti} = fltr_{ij} * X_{tj}
- **Tf** (*float*) – Total time of the trajectory
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input).
- **param_priors** (*dict*) – A dictionary for priors for the model parameters. See *latent_infer* for further explanations.
- **init_priors** (*dict*) – A dictionary for priors for the initial conditions. See *latent_infer* for further explanations.
- **contactMatrix** – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.
- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.
- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays. The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.

- **tangent** (*bool, optional*) – Set to True to do inference in tangent space (might be less robust but a lot faster). Default is False.
- **verbose** (*bool, optional*) – Set to True to see intermediate outputs from the nested sampling procedure.
- **nprocesses** (*int, optional*) – The number of processes used for parallel evaluation of the likelihood.
- **queue_size** (*int, optional*) – Size of internal queue of likelihood values, default is nprocesses if multiprocessing is used.
- **maxiter** (*int, optional*) – The maximum number of iterations. Default is no limit.
- **maxcall** (*int, optional*) – The maximum number of calls to the likelihood function. Default no limit.
- **dlogz** (*float, optional*) – The iteration terminates if the estimated contribution of the remaining prior volume to the total evidence falls below this threshold. Default value is *1e-3 * (nlive - 1) + 0.01* if *add_live==True*, 0.01 otherwise.
- **n_effective** (*float, optional*) – The iteration terminates if the number of effective posterior samples reaches this values. Default is no limit.
- **add_live** (*bool, optional*) – Determines whether to add the remaining set of live points to the set of samples. Default is True.
- **sampler** (*dynesty.NestedSampler, optional*) – Continue running an instance of a nested sampler until the termination criteria are met.
- ****dynesty_args** – Arguments passed through to the construction of the dynesty.NestedSampler constructor. Relevant entries are (this is not comprehensive, for details see the documentation of dynesty):

  nlive: int, optional The number of live points. Default is 500.

  bound: {'none', 'single', 'multi', 'balls', 'cubes'}, optional Method used to approximately bound the prior using the current set of live points. Default is 'multi'.

  sample: {'auto', 'unif', 'rwalk', 'rstagger', 'slice', 'rslice', 'hslice', callable}, optional Method used to sample uniformly within the likelihood constraint, conditioned on the provided bounds.

Returns **sampler** – The state of the sampler after termination of the nested sampling run.

Return type dynesty.NestedSampler

latent_infer_nested_sampling_process_result()

Take the sampler generated by *pyross.inference.latent_infer_nested_sampling* and produce output dictionaries for further use in the pyross framework. See there for additional description of parameters.

Parameters

- **sampler** (*dynesty.NestedSampler*) – Output of *pyross.inference.latent_infer_nested_sampling*.
- **obs** (*2d numpy.array*) –
- **fltr** (*2d numpy.array*) –
- **param_priors** (*dict*) –
- **init_priors** (*dict*) –
- **contactMatrix** (*callable, optional*) –

- **generator** (*pyross.contactMatrix, optional*) –
- **intervention_fun** (*callable, optional*) –
- ****catchall_kwargs** (*dict*) – Catches further provided arguments and ignores them.

Returns

- **result** (*dynesty.Result*) – The result of the nested sampling iteration. Relevant entries include:

  result.logz: list The progression of log-evidence estimates, use result.logz[-1] for the final estimate.

- **output_samples** (*list*) – The processed weighted posterior samples.

latent_infer_parameters()

Compute the maximum a-posteriori (MAP) estimate of the parameters and the initial conditions of a SIR type model when the classes are only partially observed. Unobserved classes are treated as latent variables.

Parameters **latent_infer** (*see*) –

Returns

output – Contains the following keys for users:

map_params_dict: dict A dictionary for the MAP estimates for parameter values. The keys are the names of the parameters.

map_x0: np.array The MAP estimate for the initial conditions.

-logp: float The value of -logp at MAP.

Return type dict

Note: This function just calls latent_infer (with fixed *contactMatrix*), will be deprecated.

---

latent_param_slice()

Samples the posterior and prior along a one-dimensional slice of the parameter space

Parameters

- **obs** (*np.array*) – The partially observed trajectory.
- **fltr** (*2d np.array*) – The filter for the observation such that $F_{ij}x_j(t) = obs_i(t)$
- **Tf** (*float*) – The total time of the trajectory.
- **infer_result** (*dict*) – Dictionary returned by latent_infer
- **pos** (*np.array*) – Position in parameter space around which the parameter slice is computed
- **direction** (*np.array*) – Direction in parameter space in which the parameter slice is computed
- **scale** (*np.array*) – Values by which the direction vector is scaled. Points evaluated are pos + scale * direction
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.

- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.
- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays. The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.
- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is False.
- **inter_steps** (*int, optional*) – Intermediate steps between observations for the deterministic forward Euler integration. A higher number of intermediate steps will improve the accuracy of the result, but will make computations slower. Setting *inter_steps=0* will fall back to the method accessible via *det_method* for the deterministic integration.
- **nprocesses** (*int, optional*) – The number of processes used to compute the likelihood for the walkers, needs *pathos*. Default is the number of cpu cores if *pathos* is available, otherwise 1.

Returns

- **posterior** (*np.array*) – posterior evaluated along the 1d slice
- **prior** (*np.array*) – prior evaluated along the 1d slice

mcmc_inference()

Sample the posterior distribution of the epidemiological parameters using ensemble MCMC.

Note: This function has been replaced by *pyross.inference.infer_mcmc* and will be deleted in a future version of pyross. See there for a documentation of the function parameters.

---

mcmc_inference_process_result()

Take the sampler generated by *mcmc_inference* and produce output dictionaries for further use in the pyross framework.

Note: This function has been replaced by *pyross.inference.infer_mcmc_process_result* and will be deleted in a future version of pyross. See there for a documentation of the function parameters.

---

mcmc_latent_inference()

Sample the posterior distribution of the epidemiological parameters using ensemble MCMC.

Note: This function has been replaced by *pyross.inference.latent_infer_mcmc* and will be deleted in a future version of pyross. See there for a documentation of the function parameters.

---

mcmc_latent_inference_process_result()

Take the sampler generated by *mcmc_latent_inference* and produce output dictionaries for further use in the pyross framework.

Note: This function has been replaced by *pyross.inference.latent_infer_mcmc_process_results* and will

be deleted in a future version of pyross. See there for a documentation of the function parameters.

minus_logp_red()
Computes -logp for a latent trajectory

Parameters

- **parameters** (*dict*) – A dictionary of parameter values, same as the ones required for initialisation.
- **x0** (*numpy.array*) – Initial conditions
- **obs** (*numpy.array*) – The observed trajectory without the initial datapoint
- **fltr** (*boolean sequence or array*) – True for observed and False for unobserved. e.g. if only Is is known for SIR with one age group, fltr = [False, False, True]
- **Tf** (*float*) – The total time of the trajectory
- **contactMatrix** (*callable*) – A function that returns the contact matrix at time t (input).
- **tangent** (*bool, optional*) – Set to True to do inference in tangent space (might be less robust but a lot faster). Default is False.

Returns **minus_logp** – -log(p) for the observed trajectory with the given parameters and initial conditions

Return type float

nested_sampling_inference()
Run nested sampling for model parameters without latent variables.

Note: This function has been replaced by *pyross.inference.infer_nested_sampling* and will be deleted in a future version of pyross. See there for a documentation of the function parameters.

nested_sampling_inference_process_result()
Take the sampler generated by *nested_sampling_inference* and produce output dictionaries for further use in the pyross framework.

Note: This function has been replaced by *pyross.inference.infer_nested_sampling_process_result* and will be deleted in a future version of pyross. See there for a documentation of the function parameters.

nested_sampling_latent_inference()
Compute the log-evidence and weighted samples of the a-posteriori distribution of the parameters of a SIR type model with latent variables using nested sampling as implemented in the *dynesty* Python package.

Note: This function has been replaced by *pyross.inference.latent_infer_nested_sampling* and will be deleted in a future version of pyross. See there for a documentation of the function parameters.

nested_sampling_latent_inference_process_result()
Take the sampler generated by *nested_sampling_latent_inference* and produce output dictionaries for further use in the pyross framework.

Note: This function has been replaced by *pyross.inference.latent_infer_nested_sampling_process_result* and will be deleted in a future version of pyross. See there for a documentation of the function parameters.

obtain_minus_log_p()
Computes -logp of a full trajectory :param parameters: A dictionary for the model parameters. :type parameters: dict :param x: The full trajectory. :type x: np.array :param Tf: The time duration of the trajectory. :type Tf: float :param contactMatrix: A function that takes time (t) as an argument and returns the contactMatrix :type contactMatrix: callable :param tangent: Set to True to use tangent space inference. :type tangent: bool, optional

Returns **minus_logp** – Value of -logp

Return type float

robustness()
Robustness analysis in a two-dimensional slice of the parameter space, revealing neutral spaces as in https://doi.org/10.1073/pnas.1015814108.

Parameters

- **FIM** (*2d numpy.array*) – Fisher Information matrix of a stochastic model
- **FIM_det** (*2d numpy.array*) – Fisher information matrix of the corresponding deterministic model
- **infer_result** (*dict*) – Dictionary returned by *latent_infer*
- **param_pos_1** (*int*) – Position of 'parameter 1' in map_dict['flat_map'] for x-axis
- **param_pos_2** (*int*) – Position of 'parameter 2' in map_dict['flat_map'] for y-axis
- **range_1** (*float*) – Symmetric interval around parameter 1 for which robustness will be analysed. Absolute interval: 'parameter 1' +/- range_1
- **range_2** (*float*) – Symmetric interval around parameter 2 for which robustness will be analysed. Absolute interval: 'parameter 2' +/- range_2
- **resolution_1** (*int*) – Resolution of the meshgrid in x direction.
- **resolution_2** (*int*) – Resolution of the meshgrid in y direction. Default is resolution_2=resolution_1.

Returns

- **ff** (*2d numpy.array*) – shape=resolution_1 x resolution_2, meshgrid for x-axis
- **ss** (*2d numpy.array*) – shape=resolution_1 x resolution_2, meshgrid for y-axis
- **Z_sto** (*2d numpy.array*) – shape=resolution_1 x resolution_2, expected quadratic coefficient in the Taylor expansion of the likelihood of the stochastic model
- **Z_det** (*2d numpy.array*) – shape=resolution_1 x resolution_2, expected quadratic coefficient in the Taylor expansion of the likelihood of the deterministic model

Examples

```
>>> from matplotlib import pyplot as plt
>>> from matplotlib import cm
>>>
>>> # positions 0 and 1 of infer_result['flat_params'] correspond to a scale␣
↪parameter for alpha, and beta, respectively.
```

(continues on next page)

(continued from previous page)

```
>>> ff, ss, Z_sto, Z_det = estimator.robustness(FIM, FIM_det, map_dict, 0, 1,␣
↪0.5, 0.01, 20)
>>> cmap = plt.cm.PuBu_r
>>> levels=11
>>> colors='black'
>>>
>>> c = plt.contourf(ff, ss, Z_sto, cmap=cmap, levels=levels) # heat map for␣
↪the stochastic coefficient
>>> plt.contour(ff, ss, Z_sto, colors='black', levels=levels, linewidths=0.25)
>>> plt.contour(ff, ss, Z_det, colors=colors, levels=levels) # contour plot␣
↪for the deterministic model
>>> plt.plot(infer_result['flat_params'][0], infer_result['flat_params'][1],
↪'o',
            color="#A60628", markersize=6) # the MAP estimate
>>> plt.colorbar(c)
>>> plt.xlabel(r'$lpha$ scale', fontsize=20, labelpad=10)
>>> plt.ylabel(r'$eta$', fontsize=20, labelpad=10)
>>> plt.show()
```

sample_gaussian()

Sample *N* samples of the parameters from the Gaussian centered at the MAP estimate with specified covariance *cov*.

Parameters

- **N** (*int*) – The number of samples.
- **map_estimate** (*dict*) – The MAP estimate, e.g. as computed by *inference.infer_parameters*.
- **cov** (*np.array*) – The covariance matrix of the flat parameters.
- **x** (*np.array*) – The full trajectory.
- **Tf** (*float*) – The total time of the trajectory.
- **contactMatrix** (*callable*) – A function that returns the contact matrix at time t (input).
- **prior_dict** (*dict*) – A dictionary containing priors. See examples.
- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is False.

Returns **samples** – N samples of the Gaussian distribution.

Return type list of dict

sample_gaussian_latent()

Sample *N* samples of the parameters from the Gaussian centered at the MAP estimate with specified covariance *cov*.

Parameters

- **N** (*int*) – The number of samples.
- **obs** (*np.array*) – The partially observed trajectory.
- **fltr** (*2d np.array*) – The filter for the observation such that $F_{ij}x_j(t) = obs_i(t)$
- **Tf** (*float*) – The total time of the trajectory.
- **infer_result** (*dict*) – Dictionary returned by latent_infer

- **invcov** (*np.array*) – The inverse covariance matrix of the flat parameters.
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.
- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.
- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays. The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.
- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is False.
- **allow_negative** (*bool, optional*) – Allow negative values of the sample parameters. If False, samples with negative paramters values are discarded and additional samples are drawn until the specified number *N* of samples is reached. Default is False.
- **inter_steps** (*int, optional*) – Intermediate steps between observations for the deterministic forward Euler integration. A higher number of intermediate steps will improve the accuracy of the result, but will make computations slower. Setting *inter_steps=0* will fall back to the method accessible via *det_method* for the deterministic integration.
- **nprocesses** (*int, optional*) – The number of processes used to compute the likelihood for the walkers, needs *pathos*. Default is the number of cpu cores if *pathos* is available, otherwise 1.

Returns

- **samples** (*list of np.array's*) – N samples of the Gaussian distribution (flat parameters).
- **posterior** (*np.array*) – posterior evaluated along the 1d slice
- **prior** (*np.array*) – prior evaluated along the 1d slice

sample_latent()

Samples the posterior and prior

Parameters

- **obs** (*np.array*) – The partially observed trajectory.
- **fltr** (*2d np.array*) – The filter for the observation such that $F_{ij} x_j(t) = obs_i(t)$
- **Tf** (*float*) – The total time of the trajectory.
- **infer_result** (*dict*) – Dictionary returned by latent_infer
- **flat_params_list** (*list of np.array's*) – Parameters for which the prior and posterior are sampled
- **contactMatrix** (*callable, optional*) – A function that returns the contact matrix at time t (input). If specified, control parameters are not inferred. Either a contactMatrix or a generator must be specified.
- **generator** (*pyross.contactMatrix, optional*) – A pyross.contactMatrix object that generates a contact matrix function with specified lockdown parameters. Either a contactMatrix or a generator must be specified.

- **intervention_fun** (*callable, optional*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO are (2, M) arrays. The contact matrices are then rescaled as $aW[0]_i CW_{ij} aW[1]_j$ etc. If not set, assume intervention that's constant in time. See *contactMatrix.constant_contactMatrix* for details on the keyword parameters.
- **tangent** (*bool, optional*) – Set to True to use tangent space inference. Default is False.
- **inter_steps** (*int, optional*) – Intermediate steps between observations for the deterministic forward Euler integration. A higher number of intermediate steps will improve the accuracy of the result, but will make computations slower. Setting *inter_steps=0* will fall back to the method accessible via *det_method* for the deterministic integration.
- **nprocesses** (*int, optional*) – The number of processes used to compute the likelihood for the walkers, needs *pathos*. Default is the number of cpu cores if *pathos* is available, otherwise 1.

Returns

- **posterior** (*np.array*) – posterior evaluated along the 1d slice
- **prior** (*np.array*) – prior evaluated along the 1d slice

sensitivity()

Computes sensitivity measures (not normalised) for 1) each individual parameter: from the diagonal elements of the FIM 2) incorporating parametric interactions: from the standard deviations derived from the FIM More on these interpretations can be found here: https://doi.org/10.1529/biophysj.104.053405 A larger entry translates into greater anticipated model sensitivity to changes in the parameter of interest.

Parameters **FIM** (*2d numpy.array*) – The Fisher Information Matrix

Returns

- **sensitivity_individual** (*numpy.array*) – Sensitivity measure for individual parameters.
- **sensitivity_correlated** (*numpy.array*) – Sensitivity measure incorporating parametric interactions.

set_contact_matrix()

Sets the internal contact matrix

Parameters **contactMatrix** (*callable*) – A function that returns the contact matrix given time, with call signature contactMatrix(t).

set_det_method()

Sets the method used for deterministic integration for the SIR_type model

Parameters

- **det_method** (*str*) – The name of the integration method. Choose between 'LSODA' and 'RK45'.
- **rtol** (*double, optional*) – relative tolerance of the integrator (default 1e-3)
- **max_steps** (*int, optional*) – Maximum number of integration steps (total) for the integrator. Default: unlimited (represented as 0) Parameters for which the integrator reaches max_steps are disregarded by the optimiser.

set_det_model()

Sets the internal deterministic model with given epidemiological parameters

Parameters **parameters** (*dict*) – A dictionary of parameter values, same as the ones required for initialisation.

set_lyapunov_method()
Sets the method used for deterministic integration for the SIR_type model

Parameters

- **lyapunov_method** (*str*) – The name of the integration method. Choose between 'LSODA', 'RK45', 'RK2', 'RK4', and 'euler'.
- **rtol** (*double, optional*) – relative tolerance of the integrator (default 1e-3)
- **max_steps** (*int*) – Maximum number of integration steps (total) for the integrator. Default: unlimited (represented as 0) Parameters for which the integrator reaches max_steps are disregarded by the optimiser.

set_params()
Sets epidemiological parameters used for evaluating -log(p)

Parameters **parameters** (*dict*) – A dictionary containing all epidemiological parameters. Same keys as the one used to initialise the class.

Notes

Can use *fill_params_dict* to generate the full dictionary if only a few parameters are changed

3.4.2 Model

class pyross.inference.**Model**
Generic user-defined epidemic model.

To initialise the Model,

Parameters

- **model_spec** (*dict*) – A dictionary specifying the model. See *Examples*.
- **parameters** (*dict*) – A dictionary containing the model parameters. All parameters can be float if not age-dependent, and np.array(M,) if age-dependent
- **M** (*int*) – Number of age groups.
- **fi** (*np.array(M) or list*) – Fraction of each age group.
- **Omega** (*int*) – Total population.
- **steps** (*int, optional*) – The number of internal integration steps performed between the observed points (not used in tangent space inference). For robustness, set steps to be large, lyapunov_method='LSODA'. For speed, set steps to be small (~4), lyapunov_method='euler'. For a combination of the two, choose something in between.
- **det_method** (*str, optional*) – The integration method used for deterministic integration. Choose one of 'LSODA' and 'RK45'. Default is 'LSODA'.
- **lyapunov_method** (*str, optional*) – The integration method used for the integration of the Lyapunov equation for the covariance. Choose one of 'LSODA', 'RK45', 'RK2', 'RK4' and 'euler'. Default is 'LSODA'.
- **rtol_det** (*float, optional*) – relative tolerance for the deterministic integrator (default 1e-3)

- **rtol_lyapunov** (*float, optional*) – relative tolerance for the Lyapunov-type integrator (default 1e-3)
- **max_steps_det** (*int, optional*) – Maximum number of integration steps (total) for the deterministic integrator. Default: unlimited (represented as 0) Parameters for which the integrator reaches max_steps_det are disregarded by the optimiser.
- **max_steps_lyapunov** (*int, optional*) – Maximum number of integration steps (total) for the Lyapunov-type integrator. Default: unlimited (represented as 0) Parameters for which the integrator reaches max_steps_lyapunov are disregarded by the optimiser.
- **parameter_mapping** (*python function, optional*) – A user-defined function that maps the dictionary the parameters used for inference to a dictionary of parameters used in model_spec. Default is an identical mapping.
- **time_dep_param_mapping** (*python function, optional*) – As parameter_mapping, but time-dependent. The user-defined function takes time as a second argument.
- **SIR_type for a table of all the methods** (*See*) –

Examples

An example of model_spec and parameters for SIR class with a constant influx

```
>>> model_spec = {
        "classes" : ["S", "I"],
        "S" : {
            "constant"  : [ ["k"] ],
            "infection" : [ ["I", "S", "-beta"] ]
        },
        "I" : {
            "linear"    : [ ["I", "-gamma"] ],
            "infection" : [ ["I", "S", "beta"] ]
        }
    }
>>> parameters = {
        'beta': 0.1,
        'gamma': 0.1,
        'k': 1,
    }
```

- Link to example notebook

3.4.3 Spp

class pyross.inference.**Spp**

This is a slightly more specific version of the class *Model*.

Spp is still supported for backward compatibility.

Model class is recommended over *Spp* for new users.

The *Spp* class works like *Model* but infection terms use a single class *S* ...

Parameters

- **model_spec** (*dict*) – A dictionary specifying the model. See *Examples*.

- **parameters** (*dict*) – A dictionary containing the model parameters. All parameters can be float if not age-dependent, and np.array(M,) if age-dependent
- **M** (*int*) – Number of age groups.
- **fi** (*np.array(M) or list*) – Fraction of each age group.
- **Omega** (*int*) – Total population.
- **steps** (*int, optional*) – The number of internal integration steps performed between the observed points (not used in tangent space inference). For robustness, set steps to be large, lyapunov_method='LSODA'. For speed, set steps to be small (~4), lyapunov_method='euler'. For a combination of the two, choose something in between.
- **det_method** (*str, optional*) – The integration method used for deterministic integration. Choose one of 'LSODA' and 'RK45'. Default is 'LSODA'.
- **lyapunov_method** (*str, optional*) – The integration method used for the integration of the Lyapunov equation for the covariance. Choose one of 'LSODA', 'RK45', 'RK2' and 'euler'. Default is 'LSODA'.
- **rtol_det** (*float, optional*) – relative tolerance for the deterministic integrator (default 1e-3)
- **rtol_lyapunov** (*float, optional*) – relative tolerance for the Lyapunov-type integrator (default 1e-3)
- **max_steps_det** (*int, optional*) – Maximum number of integration steps (total) for the deterministic integrator. Default: unlimited (represented as 0) Parameters for which the integrator reaches max_steps_det are disregarded by the optimiser.
- **max_steps_lyapunov** (*int, optional*) – Maximum number of integration steps (total) for the Lyapunov-type integrator. Default: unlimited (represented as 0) Parameters for which the integrator reaches max_steps_lyapunov are disregarded by the optimiser.
- **parameter_mapping** (*python function, optional*) – A user-defined function that maps the dictionary the parameters used for inference to a dictionary of parameters used in model_spec. Default is an identical mapping.
- **time_dep_param_mapping** (*python function, optional*) – As parameter_mapping, but time-dependent. The user-defined function takes time as a second argument.
- **SIR_type for a table of all the methods** (*See*) –

Examples

An example of model_spec and parameters for SIR class with a constant influx

```
>>> model_spec = {
        "classes" : ["S", "I"],
        "S" : {
            "constant"  : [ ["k"] ],
            "infection" : [ ["I", "-beta"] ]
        },
        "I" : {
            "linear"    : [ ["I", "-gamma"] ],
            "infection" : [ ["I", "beta"] ]
        }
    }
```

(continues on next page)

(continued from previous page)

```
>>> parameters = {
        'beta': 0.1,
        'gamma': 0.1,
        'k': 1,
    }
```

- Link to example notebook

3.4.4 SIR

class pyross.inference.**SIR**

Susceptible, Infected, Removed (SIR)

- Ia: asymptomatic
- Is: symptomatic

To initialise the SIR class,

Parameters

- **parameters** (*dict*) – Contains the following keys:

  alpha: float Ratio of asymptomatic carriers

  beta: float Infection rate upon contact

  gIa: float Recovery rate for asymptomatic

  gIs: float Recovery rate for symptomatic

  fsa: float Fraction by which symptomatic individuals do not self-isolate.

- **M** (*int*) – Number of age groups
- **fi** (*float numpy.array*) – Number of people in each age group divided by Omega.
- **Omega** (*float, optional*) – System size parameter, e.g. total population. Default to 1.
- **steps** (*int, optional*) – The number of internal integration steps performed between the observed points (not used in tangent space inference). For robustness, set steps to be large, lyapunov_method='LSODA'. For speed, set steps to be small (~4), lyapunov_method='euler'. For a combination of the two, choose something in between.
- **det_method** (*str, optional*) – The integration method used for deterministic integration. Choose one of 'LSODA' and 'RK45'. Default is 'LSODA'.
- **lyapunov_method** (*str, optional*) – The integration method used for the integration of the Lyapunov equation for the covariance. Choose one of 'LSODA', 'RK45', 'RK2', 'RK4' and 'euler'. Default is 'LSODA'.
- **rtol_det** (*float, optional*) – relative tolerance for the deterministic integrator (default 1e-4)
- **rtol_lyapunov** (*float, optional*) – relative tolerance for the Lyapunov-type integrator (default 1e-3)
- **max_steps_det** (*int, optional*) – Maximum number of integration steps (total) for the deterministic integrator. Default: unlimited (represented as 0). Parameters for which the integrator reaches max_steps_det are disregarded by the optimiser.

- **max_steps_lyapunov** (*int, optional*) – Maximum number of integration steps (total) for the Lyapunov-type integrator. Default: unlimited (represented as 0) Parameters for which the integrator reaches max_steps_lyapunov are disregarded by the optimiser.

3.4.5 SEIR

class pyross.inference.**SEIR**

Susceptible, Exposed, Infected, Removed (SEIR)

- Ia: asymptomatic
- Is: symptomatic

To initialise the SEIR class,

Parameters

- **parameters** (*dict*) – Contains the following keys:

  alpha: float or np.array(M) Fraction of infected who are asymptomatic.

  beta: float Rate of spread of infection.

  gIa: float Rate of removal from asymptomatic individuals.

  gIs: float Rate of removal from symptomatic individuals.

  fsa: float Fraction by which symptomatic individuals do not self-isolate.

  gE: float rate of removal from exposed individuals.

- **M** (*int*) – Number of age groups
- **fi** (*float numpy.array*) – Number of people in each compartment divided by Omega
- **Omega** (*float, optional*) – System size, e.g. total population. Default is 1.
- **steps** (*int, optional*) – The number of internal integration steps performed between the observed points (not used in tangent space inference). For robustness, set steps to be large, lyapunov_method='LSODA'. For speed, set steps to be small (~4), lyapunov_method='euler'. For a combination of the two, choose something in between.
- **det_method** (*str, optional*) – The integration method used for deterministic integration. Choose one of 'LSODA' and 'RK45'. Default is 'LSODA'.
- **lyapunov_method** (*str, optional*) – The integration method used for the integration of the Lyapunov equation for the covariance. Choose one of 'LSODA', 'RK45', 'RK2', 'RK4' and 'euler'. Default is 'LSODA'.
- **rtol_det** (*float, optional*) – relative tolerance for the deterministic integrator (default 1e-3)
- **rtol_lyapunov** (*float, optional*) – relative tolerance for the Lyapunov-type integrator (default 1e-3)
- **max_steps_det** (*int, optional*) – Maximum number of integration steps (total) for the deterministic integrator. Default: unlimited (represented as 0) Parameters for which the integrator reaches max_steps_det are disregarded by the optimiser.
- **max_steps_lyapunov** (*int, optional*) – Maximum number of integration steps (total) for the Lyapunov-type integrator. Default: unlimited (represented as 0) Parameters for which the integrator reaches max_steps_lyapunov are disregarded by the optimiser.

3.4.6 SEAIRQ

class pyross.inference.**SEAIRQ**

Susceptible, Exposed, Asymptomatic and infected, Infected, Removed, Quarantined (SEAIRQ)

- Ia: asymptomatic
- Is: symptomatic
- E: exposed
- A: asymptomatic and infectious
- Q: quarantined

To initialise the SEAIRQ class,

Parameters

- **parameters** (*dict*) – Contains the following keys:

  alpha: float or np.array(M) Fraction of infected who are asymptomatic.

  beta: float Rate of spread of infection.

  gIa: float Rate of removal from asymptomatic individuals.

  gIs: float Rate of removal from symptomatic individuals.

  gE: float rate of removal from exposed individuals.

  gA: float rate of removal from activated individuals.

  fsa: float Fraction by which symptomatic individuals do not self-isolate.

  tE: float testing rate and contact tracing of exposeds

  tA: float testing rate and contact tracing of activateds

  tIa: float testing rate and contact tracing of asymptomatics

  tIs: float testing rate and contact tracing of symptomatics

- **M** (*int*) – Number of compartments
- **fi** (*float numpy.array*) – Number of people in each compartment divided by Omega.
- **Omega** (*float, optional*) – System size, e.g. total population. Default is 1.
- **steps** (*int, optional*) – The number of internal integration steps performed between the observed points (not used in tangent space inference). For robustness, set steps to be large, lyapunov_method='LSODA'. For speed, set steps to be small (~4), lyapunov_method='euler'. For a combination of the two, choose something in between.
- **det_method** (*str, optional*) – The integration method used for deterministic integration. Choose one of 'LSODA' and 'RK45'. Default is 'LSODA'.
- **lyapunov_method** (*str, optional*) – The integration method used for the integration of the Lyapunov equation for the covariance. Choose one of 'LSODA', 'RK45', 'RK2', 'RK4' and 'euler'. Default is 'LSODA'.
- **rtol_det** (*float, optional*) – relative tolerance for the deterministic integrator (default 1e-3)
- **rtol_lyapunov** (*float, optional*) – relative tolerance for the Lyapunov-type integrator (default 1e-3)

- **max_steps_det** (*int, optional*) – Maximum number of integration steps (total) for the deterministic integrator. Default: unlimited (represented as 0) Parameters for which the integrator reaches max_steps_det are disregarded by the optimiser.
- **max_steps_lyapunov** (*int, optional*) – Maximum number of integration steps (total) for the Lyapunov-type integrator. Default: unlimited (represented as 0) Parameters for which the integrator reaches max_steps_lyapunov are disregarded by the optimiser.

3.4.7 SppQ

class pyross.inference.**SppQ**

User-defined epidemic model with quarantine stage.

This is a slightly more specific version of the class *Model*.

SppQ is still supported for backward compatibility.

Model class is recommended over *SppQ* for new users.

To initialise the SppQ model, . . .

Parameters

- **model_spec** (*dict*) – A dictionary specifying the model. See *Examples*.
- **parameters** (*dict*) – A dictionary containing the model parameters. All parameters can be float if not age-dependent, and np.array(M,) if age-dependent
- **testRate** (*python function*) – number of tests per day and age group
- **M** (*int*) – Number of age groups.
- **fi** (*np.array(M) or list*) – Fraction of each age group.
- **Omega** (*int*) – Total population.
- **steps** (*int, optional*) – The number of internal integration steps performed between the observed points (not used in tangent space inference). For robustness, set steps to be large, lyapunov_method='LSODA'. For speed, set steps to be small (~4), lyapunov_method='euler'. For a combination of the two, choose something in between.
- **det_method** (*str, optional*) – The integration method used for deterministic integration. Choose one of 'LSODA' and 'RK45'. Default is 'LSODA'.
- **lyapunov_method** (*str, optional*) – The integration method used for the integration of the Lyapunov equation for the covariance. Choose one of 'LSODA', 'RK45', 'RK2', 'RK4' and 'euler'. Default is 'LSODA'.
- **rtol_det** (*float, optional*) – relative tolerance for the deterministic integrator (default 1e-3)
- **rtol_lyapunov** (*float, optional*) – relative tolerance for the Lyapunov-type integrator (default 1e-3)
- **max_steps_det** (*int, optional*) – Maximum number of integration steps (total) for the deterministic integrator. Default: unlimited (represented as 0) Parameters for which the integrator reaches max_steps_det are disregarded by the optimiser.
- **max_steps_lyapunov** (*int, optional*) – Maximum number of integration steps (total) for the Lyapunov-type integrator. Default: unlimited (represented as 0) Parameters for which the integrator reaches max_steps_lyapunov are disregarded by the optimiser.

- **parameter_mapping** (*python function, optional*) – A user-defined function that maps the dictionary the parameters used for inference to a dictionary of parameters used in model_spec. Default is an identical mapping.
- **time_dep_param_mapping** (*python function, optional*) – As parameter_mapping, but time-dependent. The user-defined function takes time as a second argument.
- **SIR_type for a table of all the methods** (*See*) –

Examples

An example of model_spec and parameters for SIR class with random testing (without false positives/negatives) and quarantine

```
>>> model_spec = {
        "classes" : ["S", "I"],
        "S" : {
            "infection" : [ ["I", "-beta"] ]
        },
        "I" : {
            "linear"    : [ ["I", "-gamma"] ],
            "infection" : [ ["I", "beta"] ]
        },
        "test_pos"  : [ "p_falsepos", "p_truepos", "p_falsepos"] ,
        "test_freq" : [ "tf", "tf", "tf"]
    }
>>> parameters = {
        'beta': 0.1,
        'gamma': 0.1,
        'p_falsepos': 0
        'p_truepos': 1
        'tf': 1
    }
```

3.5 Control with NPIs

Non-pharmaceutical interventions (NPIs) are strategies that mitigate the spread of a disease by suppressing its normal pathways for transmission. These include social distancing, wearing masks, working from home, and isolation of vulnerable populations. In contrast to pharmaceutical interventions, which are slow to develop but effective in the long term, NPIs can be rapidly implemented but are generally too costly to maintain indefinitely. In the modelling framework of PyRoss, we represent NPIs as modifications to the contact matrix.

3.5.1 control_integration

class pyross.control.**control_integration**

Integrator class to implement control through changing the contact matrix as a function of the current state.

simulate_deteministic : Performs a deterministic simulation.

simulate_deterministic()

Performs detemrinistic numerical integration

Parameters

- **x0** (*np.array*) – Inital state of the system.
- **events** (*list*) – List of events that the current state can satisfy to change behaviour of the contact matrix. contactMatricies
- **contactMatricies** (*list of python functions*) – New contact matrix after the corresponding event occurs
- **Tf** (*float*) – End time for integrator.
- **Nf** (*Int*) – Number of time points to evaluate at.
- **Ti** (*float, optional*) – Start time for integrator. The default is 0.
- **events_repeat** (*bool, optional*) – Wheither events is periodic in time. The default is false.
- **events_subsequent** (*bool, optional*) – TODO

Returns

- **x_eval** (*np.array(len(t), len(x0))*) – Numerical integration solution.
- **t_eval** (*np.array*) – Corresponding times at which X is evaluated at.
- **event_out** (*list*) – List of events that occured during the run.

3.5.2 SIR

class pyross.control.**SIR**

Susceptible, Infected, Removed (SIR) Ia: asymptomatic Is: symptomatic

...

Parameters

- **parameters** (*dict*) –

  Contains the following keys:

  alpha: float, np.array (M,) fraction of infected who are asymptomatic.

  beta: float rate of spread of infection.

  gIa: float rate of removal from asymptomatic individuals.

  gIs: float rate of removal from symptomatic individuals.

  fsa: float fraction by which symptomatic individuals do not self-isolate.

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(3*M,)*) – Initial number in each compartment and class

simulate()

3.5.3 SEkIkIkR

3.5.4 SIRS

3.5.5 SEIR

class pyross.control.**SEIR**

Susceptible, Exposed, Infected, Removed (SEIR) Ia: asymptomatic Is: symptomatic :param parameters:

Contains the following keys:

alpha: float, np.array (M,) fraction of infected who are asymptomatic.

beta: float rate of spread of infection.

gIa: float rate of removal from asymptomatic individuals.

gIs: float rate of removal from symptomatic individuals.

fsa: float fraction by which symptomatic individuals do not self-isolate.

gE: float rate of removal from exposed individuals.

Parameters

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(4*M,)*) – Initial number in each compartment and class

simulate()

3.5.6 SIkR

class pyross.control.**SIkR**

Susceptible, Infected, Removed (SIkR) method of k-stages of I :param parameters:

Contains the following keys:

alpha: float fraction of infected who are asymptomatic.

beta: float rate of spread of infection.

gI: float rate of removal from infectives.

kI: int number of stages of infection.

Parameters

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array((kI + 1)*M,)*) – Initial number in each compartment and class

simulate()

3.5.7 SEkIkR

class pyross.control.**SEkIkR**

Susceptible, Infected, Removed (SIkR) method of k-stages of I See: Lloyd, Theoretical Population Biology 60, 5971 (2001), doi:10.1006tpbi.2001.1525. :param parameters:

Contains the following keys:

alpha: float fraction of infected who are asymptomatic.

beta: float rate of spread of infection.

gI: float rate of removal from infected individuals.

gE: float rate of removal from exposed individuals.

ki: int number of stages of infectives.

ke: int number of stages of exposed.

Parameters

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array((kI = kE +1)*M,)*) – Initial number in each compartment and class

simulate()

3.5.8 SEAIR

3.5.9 SEAIRQ

class pyross.control.**SEAIRQ**

Susceptible, Exposed, Asymptomatic and infected, Infected, Removed, Quarantined (SEAIRQ) Ia: asymptomatic Is: symptomatic A: Asymptomatic and infectious

Parameters

- **parameters** (*dict*) –

  Contains the following keys:

  alpha: float fraction of infected who are asymptomatic.

  beta: float rate of spread of infection.

  gIa: float rate of removal from asymptomatic individuals.

  gIs: float rate of removal from symptomatic individuals.

  gE: float rate of removal from exposed individuals.

  gA: float rate of removal from activated individuals.

  fsa: float fraction by which symptomatic individuals do not self-isolate.

  tE [float] testing rate and contact tracing of exposeds

  tA [float] testing rate and contact tracing of activateds

  tIa: float testing rate and contact tracing of asymptomatics

  tIs: float testing rate and contact tracing of symptomatics

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(6*M,)*) – Initial number in each compartment and class

simulate()

3.6 Contact matrix

Classes and methods to compute contact matrix of a meta-population. The contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j. Clearly, the total number of contacts between group i to group j must equal the total number of contacts from group j to group i, and thus, for populations of fixed size the contact matrices obey the reciprocity relation $N_i C_{ij} = N_j C_{ji}$. Here N_i is the population in group i.

3.6.1 Contact Matrix Function

Generates contact matrix for given interventions

class pyross.contactMatrix.**ContactMatrixFunction**

Generates a time dependent contact matrix

For prefactors $a_{W1}, a_{W2}, a_{S1}, a_{S2}, a_{O1}, a_{O2}$ that multiply the contact matrices CW, CS, and CO. the final contact matrix is computed as

$$CM_{ij} = CH_{ij} + (a_{W1})_i CW_{ij} (a_{W2})_j + (a_{S1})_i CW_{ij} (a_{S2})_j + (a_{O1})_i CO_{ij} (a_{O2})_j$$

For all the intervention functions, if a prefactor is passed as scalar, it is set to be an M (=no. of metapopulation groups) dimensional vector with all entries equal to the scalar.

Parameters

- **CH** (*2d np.array*) – Contact matrix at home
- **CW** (*2d np.array*) – Contact matrix at work
- **CS** (*2d np.array*) – Contact matrix at school
- **CO** (*2d np.array*) – Contact matrix at other locations

constant_contactMatrix()

Constant contact matrix

Parameters

- **aW** (*float or array of size M, optional*) – Fraction of work contact per receiver of infection. Default is 1.
- **aS** (*float or array of size M, optional*) – Fraction of school contact per receiver of infection. Default is 1.
- **aO** (*float or array of size M, optional*) – Fraction of other contact per receiver of infection. Default is 1.
- **aW2** (*float or array of size M or None, optional*) – Fraction of work contact per giver of infection. If set to None, aW2 = aW.
- **aS2** (*float or array of size M or None, optional*) – Fraction of school contact per giver of infection. If set to None, aS2 = aS.
- **aO2** (*float or array of size M or None, optional*) – Fraction of other contact per giver of infection. If set to None, aO2 = aO.

Returns **contactMatrix** – A function that takes t as an argument and outputs the contact matrix

Return type callable

get_individual_contactMatrices()

Returns the internal CH, CW, CS and CO

intervention_custom_temporal()
Custom temporal interventions

Parameters

- **intervention_func** (*callable*) – The calling signature is *intervention_func(t, **kwargs)*, where t is time and kwargs are other keyword arguments for the function. The function must return (aW, aS, aO), where aW, aS and aO must be of shape (2, M)
- **kwargs** (*dict*) – Keyword arguments for the function.

Returns **contactMatrix** – A function that takes t as an argument and outputs the contact matrix.

Return type callable

Examples

An example for an custom temporal intervetion that allows for some anticipation and reaction time

```
>>> def fun(t, M, width=1, loc=0) # using keyword arguments for parameters of␣
↪the intervention
>>>     a = (1-np.tanh((t-loc)/width))/2
>>>     a_full = np.full((2, M), a)
>>>     return a_full, a_full, a_full
>>>
>>> contactMatrix = generator.intervention_custom_temporal(fun, width=5,␣
↪loc=10)
```

interventions_temporal()
Temporal interventions

Parameters

- **time** (*np.array*) – Ordered array with temporal boundaries between the different interventions.
- **interventions** (*np.array*) – Ordered matrix with prefactors aW, aS, aO such that aW1=aW2=aW during the different time intervals. Note that len(interventions) = len(times) + 1

Returns **contactMatrix** – A function that takes t as an argument and outputs the contact matrix

Return type callable

interventions_threshold()
Temporal interventions

Parameters

- **threshold** (*np.array*) – Ordered array with temporal boundaries between the different interventions.
- **interventions** (*np.array*) – Array of shape [K+1,3, ..] with prefactors during different phases of intervention The current state of the intervention is defined by the largest integer "index" such that state[j] >= thresholds[index,j] for all j.

Returns **contactMatrix** – A function that takes t as an argument and outputs the contact matrix

Return type callable

3.6.2 Spatial Contact Matrix

Approximates the spatial contact matrix given the locations, populations and areas of the geographical regions and the overall age structured contact matrix.

class pyross.contactMatrix.**SpatialContactMatrix**
A class for generating a spatial compartmental model with commute data

Let μ, ν denote spatial index and i, j denote age group index.

$$C_{ij}^{\mu\nu} = \frac{1}{N_i^\mu} \widetilde{C}_{ij}^{\mu\nu}$$

Parameters

- **b** (*float*) – Parameter b in the above equation
- **populations** (*np.array(n_loc, M)*) – Populations of regions by age groups. Here n_loc is the number of regions and M is the number of age groups.
- **areas** (*np.array(n_loc)*) – Areas of the geographical regions.
- **commutes** (*np.array(n_loc, n_loc, M)*) – Each entry commute[mu, nu, i] needs to be the number of people of age group i commuting from μ to ν. Entries with $\mu = \nu$ are ignored.

contactMatrix.**characterise_transient**()
The maximal eigenvalue (spectral abcissa), initial groth rate (numerical abcissa), the Kreiss constant (minimum bound of transient) and time of transient growth

Parameters

- **A** (*an MxM matrix*) –
- **tol** (*Used to find a first estimate of the pseudospectrum*) –
- **theta** (*normalizing factor found in Townley et al 2007, default 0*) –
- **ord** (*default 2, order of matrix norm*) –

Returns

- *[spectral abcissa, numerical abcissa, Kreiss constant,*
- *duration of transient, henrici's departure from normalcy]*

3.7 Forecasting

Forecasting with the inferred parameters, error bars and, if there are latent variables, inferred initial conditions.

3.7.1 SIR

class pyross.forecast.**SIR**
Susceptible, Infected, Removed (SIR) Ia: asymptomatic Is: symptomatic

...

Parameters

- **parameters** (*dict*) –

Contains the following keys:

alpha: float Estimate mean value of fraction of infected who are asymptomatic.

beta: float Estimate mean value of rate of spread of infection.

gIa: float Estimate mean value of rate of removal from asymptomatic individuals.

gIs: float Estimate mean value of rate of removal from symptomatic individuals.

fsa: float fraction by which symptomatic individuals do not self-isolate.

cov: np.array( ) covariance matrix for all the estimated parameters.

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(3*M,)*) – Initial number in each compartment and class

simulate()

simulate()

Parameters

- **S0** (*np.array(M,)*) – Initial number of susceptables.
- **Ia0** (*np.array(M,)*) – Initial number of asymptomatic infectives.
- **Is0** (*np.array(M,)*) – Initial number of symptomatic infectives.
- **contactMatrix** (*python function(t), optional*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j The default is None.
- **Tf** (*float, optional*) – Final time of integrator. The default is 100.
- **Nf** (*Int, optional*) – Number of time points to evaluate.The default is 101,
- **Ns** (*int, optional*) – Number of samples of parameters to take. The default is 1000.
- **nc** (*int, optional*) –
- **epsilon** (*np.float64, optional*) – Acceptable error in leap. The default is 0.03.
- **tau_update_frequency** (*int, optional*) –
- **verbose** (*bool, optional*) – Verbosity of output. The default is False.
- **Ti** (*float, optional*) – Start time of integrator. The default is 0.
- **method** (*str, optional*) – Pyross integrator to use. The default is "deterministic".
- **events** (*list of python functions, optional*) – List of events that the current state can satisfy to change behaviour of the contact matrix. Event occurs when the value of the function changes sign. Event.direction determines which direction triggers the event, takign values {+1,-1}. The default is [].
- **contactMatricies** (*list of python functions*) – New contact matrix after the corresponding event occurs The default is [].
- **events_repeat** (*bool, optional*) – Wheither events is periodic in time. The default is false.
- **events_subsequent** (*bool, optional*) – TODO

Returns

out_dict –

Dictionary containing the following keys:

X: list List of resultant trajectories

t: list List of times at which X is evaluated.

X_mean [list] Mean trajectory of X

X_std [list] Standard devation of trajectories of X at each time point.

<init params> : Initial parameters passed at object instantiation. sample_parameters : list of parameters sampled to make trajectories.

Return type dict

3.7.2 SIR_latent

class pyross.forecast.**SIR_latent**

Susceptible, Infected, Removed (SIR) Ia: asymptomatic Is: symptomatic

Latent inference class to be used when observed data is incomplete. ...

Parameters

- **parameters** (*dict*) –

  Contains the following keys:

  alpha: float Estimate mean value of fraction of infected who are asymptomatic.

  beta: float Estimate mean value of rate of spread of infection.

  gIa: float Estimate mean value of rate of removal from asymptomatic individuals.

  gIs: float Estimate mean value of rate of removal from symptomatic individuals.

  fsa: float fraction by which symptomatic individuals do not self-isolate.

  cov: np.array( ) Covariance matrix for all the estimated parameters.

  S0: np.array(M,) Estimate initial number of susceptables.

  Ia0: np.array(M,) Estimate initial number of asymptomatic infectives.

  Is0: np.array(M,) Estimate initial number of symptomatic infectives.

  cov_init [np.array((3*M, 3*M)) :] Covariance matrix for the initial state.

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(3*M,)*) – Initial number in each compartment and class

simulate()

simulate()

Parameters

- **contactMatrix** (*python function(t), optional*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j The default is None.
- **Tf** (*float*) – Final time of integrator.
- **Nf** (*Int*) – Number of time points to evaluate.
- **Ns** (*int*) – Number of samples of parameters to take.

- **nc** (*int, optional*) –
- **epsilon** (*np.float64, optional*) – Acceptable error in leap. The default is 0.03.
- **tau_update_frequency** (*int, optional*) – TODO
- **verbose** (*bool, optional*) – Verbosity of output. The default is False.
- **Ti** (*float, optional*) – Start time of integrator. The default is 0.
- **method** (*str, optional*) – Pyross integrator to use. The default is "deterministic".
- **events** (*list of python functions, optional*) – List of events that the current state can satisfy to change behaviour of the contact matrix. Event occurs when the value of the function changes sign. Event.direction determines which direction triggers the event, takign values {+1,-1}. The default is [].
- **contactMatricies** (*list of python functions*) – New contact matrix after the corresponding event occurs The default is [].
- **events_repeat** (*bool, optional*) – Wheither events is periodic in time. The default is false.
- **events_subsequent** (*bool, optional*) – TODO

Returns

out_dict –

Dictionary containing the following keys:

X: list List of resultant trajectories

t: list List of times at which X is evaluated.

X_mean [list] Mean trajectory of X

X_std [list] Standard devation of trajectories of X at each time point.

<init params> : Initial parameters passed at object instantiation. sample_parameters : list of parameters sampled to make trajectories. sample_inits : List of initial state vectors tried.

Return type dict

3.7.3 SEIR

class pyross.forecast.**SEIR**

Susceptible, Exposed, Infected, Removed (SEIR) Ia: asymptomatic Is: symptomatic :param parameters:

Contains the following keys:

alpha: float Estimate mean value of fraction of infected who are asymptomatic.

beta: float Estimate mean value of rate of spread of infection.

gIa: float Estimate mean value of rate of removal from asymptomatic individuals.

gIs: float Estimate mean value of rate of removal from symptomatic individuals.

fsa: float fraction by which symptomatic individuals do not self-isolate.

gE: float Estimated mean value of rate of removal from exposed individuals.

cov: np.array( ) covariance matrix for all the estimated parameters.

Parameters

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(4*M,)*) – Initial number in each compartment and class

simulate()

simulate()

Parameters

- **S0** (*np.array(M,)*) – Initial number of susceptables.
- **E0** (*np.array(M,)*) – Initial number of exposed.
- **Ia0** (*np.array(M,)*) – Initial number of asymptomatic infectives.
- **Is0** (*np.array(M,)*) – Initial number of symptomatic infectives.
- **contactMatrix** (*python function(t), optional*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j The default is None.
- **Tf** (*float, optional*) – Final time of integrator. The default is 100.
- **Nf** (*Int, optional*) – Number of time points to evaluate.The default is 101,
- **Ns** (*int, optional*) – Number of samples of parameters to take. The default is 1000.
- **nc** (*int, optional*) –
- **epsilon** (*np.float64, optional*) – Acceptable error in leap. The default is 0.03.
- **tau_update_frequency** (*int, optional*) –
- **verbose** (*bool, optional*) – Verbosity of output. The default is False.
- **Ti** (*float, optional*) – Start time of integrator. The default is 0.
- **method** (*str, optional*) – Pyross integrator to use. The default is "deterministic".
- **events** (*list of python functions, optional*) – List of events that the current state can satisfy to change behaviour of the contact matrix. Event occurs when the value of the function changes sign. Event.direction determines which direction triggers the event, takign values {+1,-1}. The default is [].
- **contactMatricies** (*list of python functions*) – New contact matrix after the corresponding event occurs The default is [].
- **events_repeat** (*bool, optional*) – Wheither events is periodic in time. The default is false.
- **events_subsequent** (*bool, optional*) – TODO

Returns

out_dict –

Dictionary containing the following keys:

X: list List of resultant trajectories

t: list List of times at which X is evaluated.

X_mean [list] Mean trajectory of X

X_std [list] Standard devation of trajectories of X at each time point.

<init params> : Initial parameters passed at object instantiation. sample_parameters : list of parameters sampled to make trajectories.

Return type dict

3.7.4 SEIR_latent

class pyross.forecast.**SEIR_latent**

Susceptible, Exposed, Infected, Removed (SEIR) Ia: asymptomatic Is: symptomatic

Latent inference class to be used when observed data is incomplete.

Parameters

- **parameters** (*dict*) –

  Contains the following keys:

  alpha: float Estimate mean value of fraction of infected who are asymptomatic.

  beta: float Estimate mean value of rate of spread of infection.

  gIa: float Estimate mean value of rate of removal from asymptomatic individuals.

  gIs: float Estimate mean value of rate of removal from symptomatic individuals.

  fsa: float fraction by which symptomatic individuals do not self-isolate.

  gE: float Estimated mean value of rate of removal from exposed individuals.

  cov: np.array( ) covariance matrix for all the estimated parameters.

  S0: np.array(M,) Estimate initial number of susceptables.

  E0: np.array(M,) Estimate initial number of exposed.

  Ia0: np.array(M,) Estimate initial number of asymptomatic infectives.

  Is0: np.array(M,) Estimate initial number of symptomatic infectives.

  cov_init [np.array((3*M, 3*M)) :] Covariance matrix for the initial state.

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(4*M,)*) – Initial number in each compartment and class

simulate()

simulate()

Parameters

- **contactMatrix** (*python function(t), optional*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j The default is None.
- **Tf** (*float, optional*) – Final time of integrator. The default is 100.
- **Nf** (*Int, optional*) – Number of time points to evaluate.The default is 101,
- **Ns** (*int, optional*) – Number of samples of parameters to take. The default is 1000.
- **nc** (*int, optional*) –
- **epsilon** (*np.float64, optional*) – Acceptable error in leap. The default is 0.03.
- **tau_update_frequency** (*int, optional*) –

- **verbose** (*bool, optional*) – Verbosity of output. The default is False.
- **Ti** (*float, optional*) – Start time of integrator. The default is 0.
- **method** (*str, optional*) – Pyross integrator to use. The default is "deterministic".
- **events** (*list of python functions, optional*) – List of events that the current state can satisfy to change behaviour of the contact matrix. Event occurs when the value of the function changes sign. Event.direction determines which direction triggers the event, takign values {+1,-1}. The default is [].
- **contactMatricies** (*list of python functions*) – New contact matrix after the corresponding event occurs The default is [].
- **events_repeat** (*bool, optional*) – Wheither events is periodic in time. The default is false.
- **events_subsequent** (*bool, optional*) – TODO

Returns

out_dict –

Dictionary containing the following keys:

X: list List of resultant trajectories

t: list List of times at which X is evaluated.

X_mean [list] Mean trajectory of X

X_std [list] Standard devation of trajectories of X at each time point.

<init params> : Initial parameters passed at object instantiation. sample_parameters : list of parameters sampled to make trajectories. sample_inits : List of initial state vectors tried.

Return type dict

3.7.5 SEAIRQ

class pyross.forecast.**SEAIRQ**

Susceptible, Exposed, Infected, Removed (SEIR) Ia: asymptomatic Is: symptomatic A: Asymptomatic and infectious :param parameters:

Contains the following keys:

alpha: float Estimate mean value of fraction of infected who are asymptomatic.

beta: float Estimate mean value of rate of spread of infection.

gIa: float Estimate mean value of rate of removal from asymptomatic individuals.

gIs: float Estimate mean value of rate of removal from symptomatic individuals.

fsa: float fraction by which symptomatic individuals do not self-isolate.

gE: float Estimated mean value of rate of removal from exposed individuals.

gA: float Estimated mean value of rate of removal from activated individuals.

cov: np.array( ) covariance matrix for all the estimated parameters.

tE [float] testing rate and contact tracing of exposeds

tA [float] testing rate and contact tracing of activateds

tIa: float testing rate and contact tracing of asymptomatics

tIs: float testing rate and contact tracing of symptomatics

Parameters

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(4*M,)*) – Initial number in each compartment and class

simulate()

simulate()

Parameters

- **S0** (*np.array*) – Initial number of susceptables.
- **E0** (*np.array*) – Initial number of exposeds.
- **A0** (*np.array*) – Initial number of activateds.
- **Ia0** (*np.array*) – Initial number of asymptomatic infectives.
- **Is0** (*np.array*) – Initial number of symptomatic infectives.
- **Q0** (*np.array*) – Initial number of quarantineds.
- **contactMatrix** (*python function(t), optional*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j The default is None.
- **Tf** (*float, optional*) – Final time of integrator. The default is 100.
- **Nf** (*Int, optional*) – Number of time points to evaluate.The default is 101,
- **Ns** (*int, optional*) – Number of samples of parameters to take. The default is 1000.
- **nc** (*int, optional*) –
- **epsilon** (*np.float64, optional*) – Acceptable error in leap. The default is 0.03.
- **tau_update_frequency** (*int, optional*) –
- **verbose** (*bool, optional*) – Verbosity of output. The default is False.
- **Ti** (*float, optional*) – Start time of integrator. The default is 0.
- **method** (*str, optional*) – Pyross integrator to use. The default is "deterministic".
- **events** (*list of python functions, optional*) – List of events that the current state can satisfy to change behaviour of the contact matrix. Event occurs when the value of the function changes sign. Event.direction determines which direction triggers the event, takign values {+1,-1}. The default is [].
- **contactMatricies** (*list of python functions*) – New contact matrix after the corresponding event occurs The default is [].
- **events_repeat** (*bool, optional*) – Wheither events is periodic in time. The default is false.
- **events_subsequent** (*bool, optional*) – TODO

Returns

out_dict –

Dictionary containing the following keys:

X: list List of resultant trajectories

t: list List of times at which X is evaluated.

X_mean [list] Mean trajectory of X

X_std [list] Standard devation of trajectories of X at each time point.

<init params> : Initial parameters passed at object instantiation. sample_parameters : list of parameters sampled to make trajectories.

Return type dict

3.7.6 SEAIRQ_latent

class pyross.forecast.**SEAIRQ_latent**

Susceptible, Exposed, Infected, Removed (SEIR) Ia: asymptomatic Is: symptomatic A: Asymptomatic and infectious

Latent inference class to be used when observed data is incomplete. :param parameters:

Contains the following keys:

alpha: float Estimate mean value of fraction of infected who are asymptomatic.

beta: float Estimate mean value of rate of spread of infection.

gIa: float Estimate mean value of rate of removal from asymptomatic individuals.

gIs: float Estimate mean value of rate of removal from symptomatic individuals.

fsa: float fraction by which symptomatic individuals do not self-isolate.

gE: float Estimated mean value of rate of removal from exposed individuals.

gA: float Estimated mean value of rate of removal from activated individuals.

cov: np.array( ) covariance matrix for all the estimated parameters.

tE [float] testing rate and contact tracing of exposeds

tA [float] testing rate and contact tracing of activateds

tIa: float testing rate and contact tracing of asymptomatics

tIs: float testing rate and contact tracing of symptomatics

S0: np.array(M,) Estimate initial number of susceptables.

E0: np.array(M,) Estimate initial number of exposed.

A0: np.array(M,) Estimate initial number of activated.

Ia0: np.array(M,) Estimate initial number of asymptomatic infectives.

Is0: np.array(M,) Estimate initial number of symptomatic infectives.

Q0: np.array(M,) Estimate initial number of quarantined.

cov_init [np.array((3*M, 3*M)) :] Covariance matrix for the initial state.

Parameters

- **M** (*int*) – Number of compartments of individual for each class. I.e len(contactMatrix)
- **Ni** (*np.array(4*M,)*) – Initial number in each compartment and class

simulate()

simulate()

Parameters

- **contactMatrix** (*python function(t), optional*) – The social contact matrix C_{ij} denotes the average number of contacts made per day by an individual in class i with an individual in class j The default is None.
- **Tf** (*float, optional*) – Final time of integrator. The default is 100.
- **Nf** (*Int, optional*) – Number of time points to evaluate.The default is 101,
- **Ns** (*int, optional*) – Number of samples of parameters to take. The default is 1000.
- **nc** (*int, optional*) –
- **epsilon** (*np.float64, optional*) – Acceptable error in leap. The default is 0.03.
- **tau_update_frequency** (*int, optional*) –
- **verbose** (*bool, optional*) – Verbosity of output. The default is False.
- **Ti** (*float, optional*) – Start time of integrator. The default is 0.
- **method** (*str, optional*) – Pyross integrator to use. The default is "deterministic".
- **events** (*list of python functions, optional*) – List of events that the current state can satisfy to change behaviour of the contact matrix. Event occurs when the value of the function changes sign. Event.direction determines which direction triggers the event, takign values {+1,-1}. The default is [].
- **contactMatricies** (*list of python functions*) – New contact matrix after the corresponding event occurs The default is [].
- **events_repeat** (*bool, optional*) – Wheither events is periodic in time. The default is false.
- **events_subsequent** (*bool, optional*) – TODO

Returns

out_dict –

Dictionary containing the following keys:

X: list List of resultant trajectories

t: list List of times at which X is evaluated.

X_mean [list] Mean trajectory of X

X_std [list] Standard devation of trajectories of X at each time point.

<init params> : Initial parameters passed at object instantiation. sample_parameters : list of parameters sampled to make trajectories. sample_inits : List of initial state vectors tried.

Return type dict

3.8 Evidence

Additional functions for computing the evidence of a pyross compartment model.

This is an extension of pyross.inference. Evidence computation via nested sampling is already directly implemented in the inference module. However, for large-scale (high-dimensional) inference problems, nested sampling can become very slow. In this module, we implement two additional ways to compute the evidence that work whenever the MCMC

simulation of the posterior distribution is feasible. See the ex-evidence.ipynb notebook for a code example of all ways to compute the evidence.

pyross.evidence.**get_parameters**(*estimator*, *x*, *Tf*, *prior_dict*, *contactMatrix=None*, *generator=None*, *intervention_fun=None*, *tangent=False*)

Process an estimator from *pyross.inference* to generate input arguments for the evidence computations *pyross.evidence.evidence_smc* and *pyross.evidence.evidence_path_sampling* for estimation problems without latent variables. The input has the same structure as the input of *pyross.inference.infer*, see there for a detailed documentation of the arguments.

Parameters

- **estimator** (pyross.inference.SIR_type) – The estimator object of the underlying (non-latent) inference problem.
- **x** (*2d numpy.array*) –
- **Tf** (*float*) –
- **prior_dict** (*dict*) –
- **contactMatrix** (*callable, optional*) –
- **generator** (*pyross.contactMatrix, optional*) –
- **intervention_fun** (*callable, optional*) –
- **tangent** (*bool, optional*) –

Returns

- *logl* – The log-likelihood of the inference problem.
- *prior* – The prior distribution of the parameters.
- *ndim* – The number of (flat) parameters.

pyross.evidence.**latent_get_parameters**(*estimator*, *obs*, *fltr*, *Tf*, *param_priors*, *init_priors*, *contactMatrix=None*, *generator=None*, *intervention_fun=None*, *tangent=False*, *smooth_penalty=False*, *disable_bounds=False*)

Process an estimator from *pyross.inference* to generate input arguments for the evidence computations *pyross.evidence.evidence_smc* and *pyross.evidence.evidence_path_sampling* for estimation problems with latent variables. The input has the same structure as the input of *pyross.inference.latent_infer*, see there for a detailed documentation of the arguments.

Parameters

- **estimator** (pyross.inference.SIR_type) – The estimator object of the underlying (non-latent) inference problem.
- **obs** (*np.array*) –
- **fltr** (*2d np.array*) –
- **Tf** (*float*) –
- **param_priors** (*dict*) –
- **init_priors** (*dict*) –
- **contactMatrix** (*callable, optional*) –
- **generator** (*pyross.contactMatrix, optional*) –
- **intervention_fun** (*callable, optional*) –

- **tangent** (*bool, optional*) –

Returns

- *logl* – The log-likelihood of the inference problem.
- *prior* – The prior distribution of the parameters.
- *ndim* – The number of (flat) parameters.

pyross.evidence.**compute_ess**(*weights*)
Compute the effective sample size of a weighted set of samples.

pyross.evidence.**compute_cess**(*old_weights*, *weights*)
Compute the conditional effective sample size as decribed in [Zhou, Johansen, Aston 2016].

pyross.evidence.**resample**(*N*, *particles*, *logl*, *probs*)
Implements the residual resampling scheme, see for example [Doucet, Johansen 2008], https://www.stats.ox.ac.uk/~doucet/doucet_johansen_tutorialPF2011.pdf

pyross.evidence.**evidence_smc**(*logl*, *prior*, *ndim*, *npopulation=200*, *target_cess=0.9*, *min_ess=0.6*, *mcmc_iter=50*, *nprocesses=0*, *save_samples=True*, *verbose=True*)
Compute the evidence using an adaptive sequential Monte Carlo method.

This function computes the model evidence of the inference problem using a sequential Monte Carlo particle method starting at the prior distribution. We implement the method *SMC2* described in [Zhou, Johansen, Aston 2016], https://doi.org/10.1080/10618600.2015.1060885

We start by sampling *npopulation* particles from the prior distribution with uniform weights. The target distribution of the weighted set of particles gets transformed to the posterior distribution by a geometric annealing schedule. The step size is chosen adaptively based on the target decay rate of the effective samples size *target_cess* in each step. Once the effective sample size of the weighted particles goes below *min_cess * npopulation*, we replace the weighted set of samples by a resampled, unweighted set. Between each step, the particles are decorrelated and equilibreated on the current level distribution by running an MCMC chain.

Parameters

- **logl** – Input from *pyross.evidence.get_parameters* or *pyross.evidence.latent_get_parameters*.
- **prior** – Input from *pyross.evidence.get_parameters* or *pyross.evidence.latent_get_parameters*.
- **ndim** – Input from *pyross.evidence.get_parameters* or *pyross.evidence.latent_get_parameters*.
- **npopulation** (*int*) – The number of particles used for the SMC iteration. Higher number of particles increases the accuracy of the result.
- **target_cess** (*float*) – The target rate for the decay of the effective sample size (ess reduces by *1-target_cess* each step). Smaller values result in more iterations and a higher accuracy result.
- **min_ess** (*float*) – The minimal effective sample size of the system. Low effective sample size imply many particles in low probability regions. However, resampling adds variance so it should not be done in every step.
- **mcmc_iter** (*int*) – The number of MCMC iterations in each step of the algorithm. The number of iterations should be large enough to equilibrate the particles with the current distribution, higher iteration numbers will typically not result in more accurate results. Oftentimes, it makes more sense to increase the number of steps (via *target_cess*) instead of increasing the number of iterations. This decreases the difference in distribution between consecutive steps and reduced the error of the final result. This number should however be

large enough to allow equal-position particles (that occur via resampling) to diverge from each other.

- **nprocesses** (*int*) – The number of processes passed to the *emcee* MCMC sampler. By default, the number of physical cores is used.
- **save_samples** (*bool*) – If true, this function returns the interal state of each MCMC iteration.
- **verbose** (*bool*) – If true, this function displays the progress of each MCMC iteration in addition to basic progress information.

Returns

- **log_evidence** (*float*) – The estimate of the log evidence.
- *if save_samples=True –*

  result_samples: list of (float, emcee.EnsembleSampler) The list of samplers and their corresponding step *alpha*.

pyross.evidence.**evidence_path_sampling**(*logl*, *prior*, *ndim*, *steps*, *npopulation=100*, *mcmc_iter=1000*, *nprocesses=0*, *initial_samples=10*, *verbose=True*, *extend_step_list=None*, *extend_sampler_list=None*)

Compute the evidence using path sampling (thermodynamic integration).

This function computes posterior samples for the distributions

p_s propto prior * likelihood^s

for 0<s1, s steps, using ensemble MCMC. The samples can be used to estimate the evidence via

log_evidence = int_0^1 E_{p_s}[log_likelihood] ds

which is know as path sampling or thermodynamic integration.

This function starts with sampling *initial_samples * npopulation* samples from the (truncated log-normal) prior. Afterwards, it runs an ensemble MCMC chain with *npopulation* ensemble members for *mcmc_iter* iterations. To minimise burn-in, the iteration is started with the last sample of the chain with the closest step *s* that has already been computed. To extend the results of this function with additional steps, provide the to-be-extended result via the optional arguments *extend_step_list* and *extend_sampler_list*.

This function only returns the step list and the corresponding samplers. To compute the evidence estimate, use *pyross.evidence.evidence_path_sampling_process_result*.

Parameters

- **logl** – Input from *pyross.evidence.get_parameters* or *pyross.evidence.latent_get_parameters*.
- **prior** – Input from *pyross.evidence.get_parameters* or *pyross.evidence.latent_get_parameters*.
- **ndim** – Input from *pyross.evidence.get_parameters* or *pyross.evidence.latent_get_parameters*.
- **steps** (*list of float*) – List of steps *s* for which the distribution *p_s* is explored using MCMC. Should be in ascending order and not include 0.
- **npopulation** (*int*) – The population size of the MCMC ensemble sampler (see documentation of *emcee* for details).
- **mcmc_iters** (*int*) – The number of iterations of the MCMC chain for each s steps.

- **nprocesses** (*int*) – The number of processes passed to the *emcee* MCMC sampler. By default, the number of physical cores is used.
- **initial_samples** (*int*) – Compute *initial_samples* * *npopulation* independent samples as the result for s = 0.
- **verbose** (*bool*) – If true, this function displays the progress of each MCMC iteration in addition to basic progress information.
- **extend_step_list** (*list of float*) – Extends the result of an earlier run of this function if this argument and *extend_sampler_list* are provided.
- **extend_sampler_list** (*list of emcee.EnsembleSampler*) – Extends the result of an earlier run of this function if this argument and *extend_step_list* are provided.

Returns

- **step_list** (*list of float*) – The steps *s* for which *p_s* has been sampled from (including 0). Always in ascending order.
- **sampler_list** (*list*) – The list of emcee.EnsembleSamplers (and an array of prior samples at 0).

pyross.evidence.**evidence_path_sampling_process_result**(*logl*, *prior*, *ndim*, *step_list*, *sampler_list*, *burn_in=0*, *nprocesses=0*)

Compute the evidence estimate for the result of *pyross.evidence.evidence_path_sampling*.

Parameters

- **logl** – Input from *pyross.evidence.get_parameters* or *pyross.evidence.latent_get_parameters*.
- **prior** – Input from *pyross.evidence.get_parameters* or *pyross.evidence.latent_get_parameters*.
- **ndim** – Input from *pyross.evidence.get_parameters* or *pyross.evidence.latent_get_parameters*.
- **step_list** (*list of float*) – Output of *pyross.evidence.evidence_path_sampling*. The steps *s* for which *p_s* has been sampled from (including 0). Always in ascending order.
- **sampler_list** (*list*) – Output of *pyross.evidence.evidence_path_sampling*. The list of emcee.EnsembleSamplers (and an array of prior samples at 0).
- **burn_in** (*float or np.array*) – The number of initial samples that are discarded before computing the Monte Carlo average.
- **nprocesses** (*int*) – The number of processes used to compute the prior likelihood. By default, the number of physical cores is used.

Returns

- **log_evidence** (*float*) – The estimate of the log evidence.
- **vals** (*list of float*) – The Monte Carlo average of the log-likelihood for each s s step_list.

pyross.evidence.**generate_traceplot**(*sampler*, *dims=None*)

Generate a traceplot for an emcee.EnsembleSampler.

Parameters

- **sampler** (*emcee.EnsembleSampler*) – The sampler to plot the traceplot for.
- **dims** (*list of int, optional*) – Select the dimensions that are plotted. By default, all dimensions are selected.

3.9 TSI

Time since infection models

3.9.1 Simulator

Python Module Index

p

Index

L

M

N

O

P

R

S