eugene Documentation

Release 0.1.dev47+g18715bd

Brett M. Morris

May 07, 2020

CONTENTS

eugene is a generalized simulator for outbreaks based on the work of Riou & Althaus (2020). The primary goal is to probe two of the parameters that describe the epidemic of COVID-19, the basic reproduction number $\mathcal{R}_0$ and the overdispersion factor k. We take an Approximate Bayesian Computation approach to estimating $\mathcal{R}_0$ and k by running thousands of stochastic outbreak simulations and exploring which values of $\mathcal{R}_0$, k, and other parameters accurately reproduce the incidence of COVID-19 on January 18, 2020.

Part I

Installation

To install eugene, run the following:

```
git clone https://github.com/bmorris3/eugene.git
cd eugene
python setup.py install
```

eugene requires numpy, scipy and corner, which you can install with:

```
pip install numpy scipy corner
```

Part II

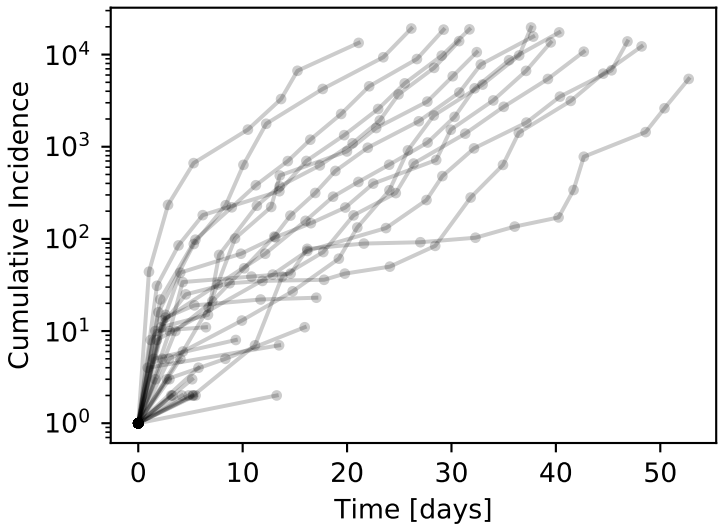
Getting started

At the core of eugene lies a simple outbreak model, which starts with number of index cases n. The user must also specify $\mathcal{R}_0$, k, the generation time between incidences D, the shape of the Gamma distribution parameterized by parameter `gamma_shape`, maximum number of days to simulate `days_elapsed_max` and the maximum number of cases beyond which to stop simulating `max_cases`. We can specify those parameters in code like so:

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2020)

parameters = dict(
    R0 = 2,  # reproduction number
    k = 1,   # overdispersion factor
    n = 1,   # number of index cases
    D = 10,  # generation time interval
    gamma_shape = 2,  # gamma function shape parameter
    max_time = 90,    # maximum simulation time
    days_elapsed_max = 52,  # number of days from index case to measurement
    max_cases = 1e4   # maximum number of cases to simulate
)
```

Now we can simulate 100 outbreaks with these initial parameters:

```
from eugene import simulate_outbreak

fig, ax = plt.subplots(figsize=(4, 3))

for i in range(100):
    times, cumulative_incidence = simulate_outbreak(**parameters)
    ax.semilogy(times, cumulative_incidence, '.-', color='k', alpha=0.2)

ax.set_xlabel('Time [days]')
ax.set_ylabel('Cumulative Incidence')
fig.tight_layout()
plt.show()
```



Every epidemic curve starts at incidence of unity, and the cumulative incidence grows roughly exponentially,

sometimes terminating with zero new cases before it reaches the end of the simulation domain (set by the `days_elapsed_max` parameter).

Part III

Reproducing Riou & Althaus (2020)

CHAPTER

ONE

APPROXIMATE BAYESIAN COMPUTATION

In this short tutorial, we'll show you how to reproduce the results of Riou & Althaus (2020), which showed that the early COVID-19 in Wuhan, China has $\mathcal{R}_0 \sim 2$ using Approximate Bayesian Computation.

First we'll import some of the required packages and eugene:

```
import numpy as np
import matplotlib.pyplot as plt
from glob import glob

from eugene import abc
```

Next we'll set the parameters of the run, which we explain below:

```
params = dict(
    # Grid of R0 and k parameters to iterate over
    R0_grid = np.logspace(np.log10(0.7), np.log10(10), 50),
    k_grid = np.logspace(-2, 1, 10),

    # Maximum number of cases to run the simulation through (should be
    # greater than ``max_number_cases``)
    max_cases = 1e4,

    # Maximum number of days someone might transmit the disease
    max_time = 90,   # days

    # Number of stochastic trials to run at each grid-point
    trials = 1000,

    # Days elapsed since zoonotic transmission
    days_elapsed_min = [46-7, 52-7],  # days
    days_elapsed_max = [46+7, 52+7],  # days

    # Number of cases after ``days_elapsed``
    min_number_cases = [190, 1000],  # cases
    max_number_cases = [5590, 9700],  # cases

    # Initial number of index cases n (day-zero cases)
    n_min = 1,   # cases
    n_max = 100,  # cases

    # Generation interval/Gamma function shape parameter
    gamma_shape_min = 1,
    gamma_shape_max = 5,
```

(continues on next page)

(continued from previous page)

```
    # Generation time interval D
    D_min = 7,   # days
    D_max = 60,  # days

    # Computer parameters
    n_processes = 16,
    n_grid_points_per_process = 2,

    # Formatting string for naming simulation outputs
    samples_path = 'samples/samples{0}.npy'
)
```

The number of processes `n_processes` should be equivalent to the number of processes you can run simultaneously on your machine. `n_grid_points_per_process` determines the number of $\mathcal{R}_0$ grid points distributed to each process. The `samples_path` argument will determine where to put the chains from the rejection sampler – you'll need to create a `samples/` directory for this example to work.

Finally, we can run the simulation with the following:

```
total_trials = (params['trials'] * params['R0_grid'].shape[0] *
                params['k_grid'].shape[0])
print(f'Total number of simulations triggered: {total_trials}')

abc(**params)
```

The Approximate Bayesian Computation `abc` function will run simple parallel processes that each save the accepted chains from a rejection sampler algorithm.

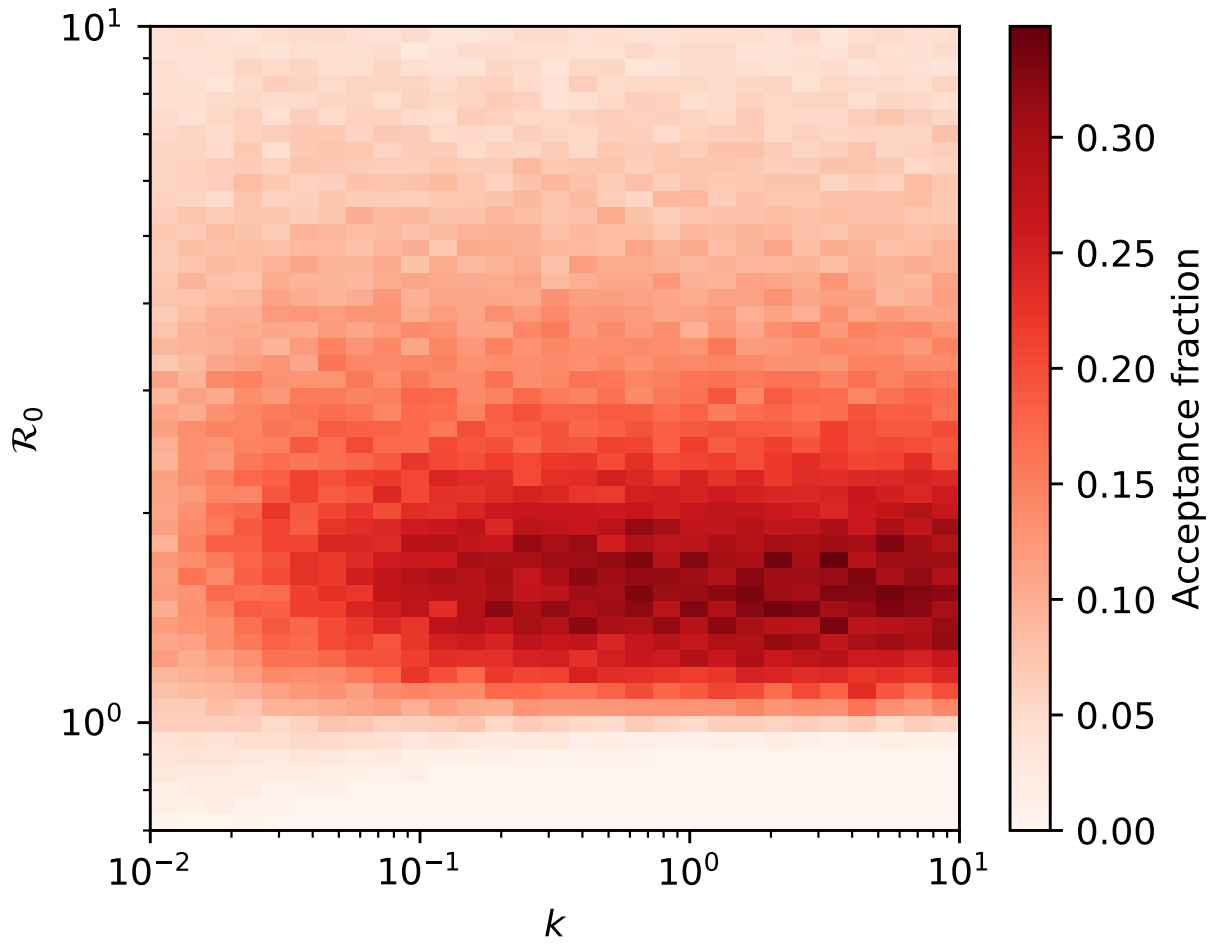
CHAPTER

TWO

VISUALIZING THE RESULTS

We can view the acceptance rate of the chains as a function of $\mathcal{R}_0$ and k with the following commands:

```
R0_grid = params['R0_grid']
k_grid = params['k_grid']
trials = params['trials']

samples = np.vstack([np.load(p) for p in glob('samples/samples*.npy')])

lo, mid, hi = np.percentile(samples[:, 0], [16, 50, 84])
print(f'R0 = {mid:.2f}_{{-{mid-lo:.2f}}}^{{+{hi-mid:.2f}}}')


hist2d, xedges, yedges = np.histogram2d(np.log10(samples[:, 0]),
                                        np.log10(samples[:, 1]),
                                        bins=[R0_grid.shape[0],
                                              k_grid.shape[0]])

fig, ax = plt.subplots(figsize=(5, 4))

X, Y = np.meshgrid(R0_grid, k_grid)

im = ax.pcolor(Y, X, hist2d.T / trials, cmap=plt.cm.Reds)

ax.set_xscale('log')
ax.set_yscale('log')

cbar = plt.colorbar(im, label='Acceptance fraction')

ax.set(xlabel='$k$', ylabel='$\mathcal{R}_0$')
fig.savefig('plots/grid.pdf', bbox_inches='tight')
plt.show()
```



The plot above shows the acceptance rate of the ABC rejection sampler as a function of $\mathcal{R}_0$ and k, darker red represents higher acceptance rates, meaning a better match between the simulated cumulative incidence curves and the observations. The median $\mathcal{R}_0 \sim 2$, meaning for every case of COVID-19 there are approximately two new cases generated, and $k \gtrsim 0.1$.

CHAPTER

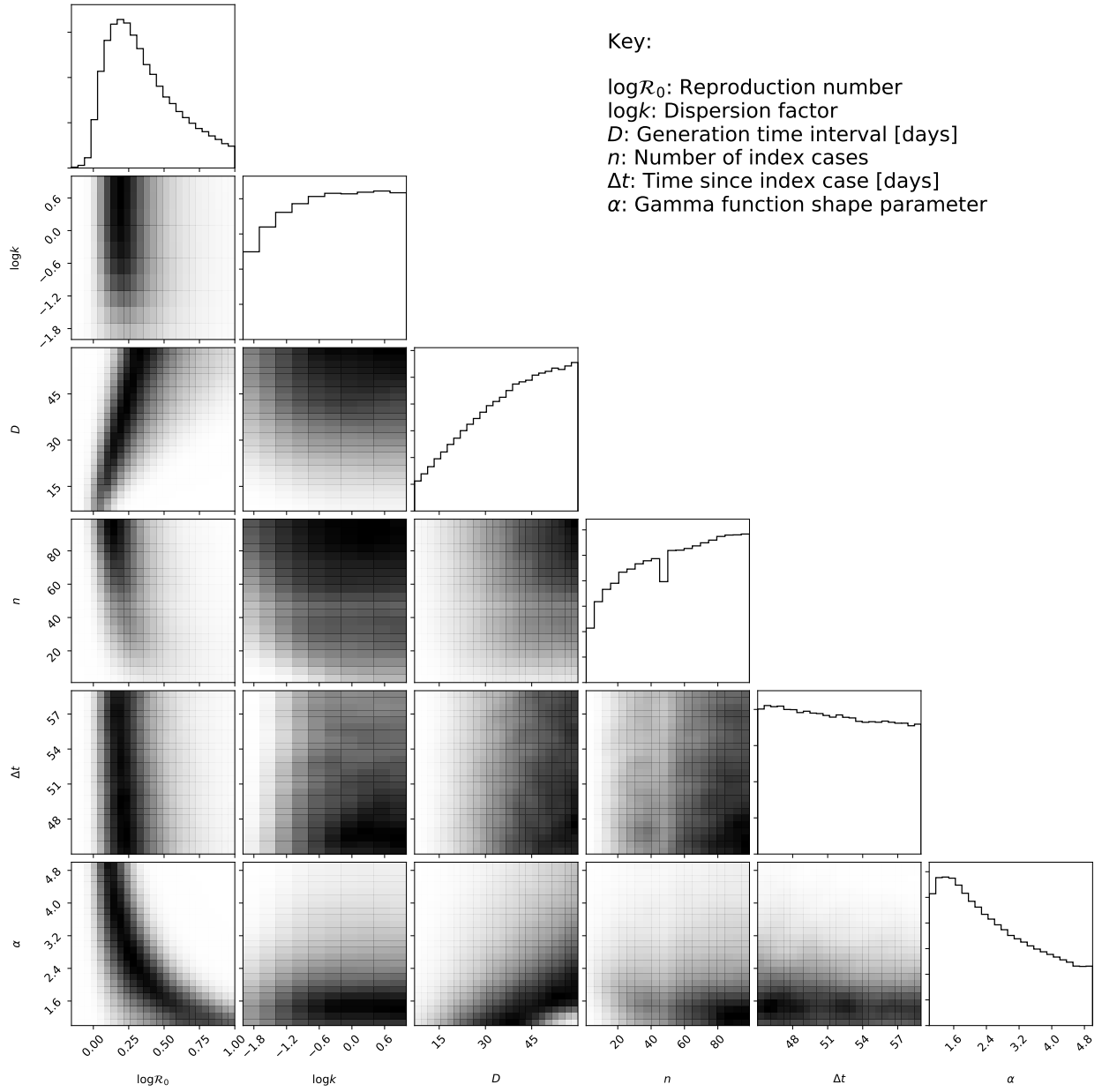
THREE

PARAMETER DEGENERACIES

Since we sampled for a range of $\mathcal{R}_0, k, D, n$, and gamma_shape parameters which we will call α, we can plot the fraction of accepted rejection sampler iterations as a function of each combination of these parameters to examine how the uncertainty on one parameter propagates into uncertainties on the others.

We can generate a *corner plot* with our results like so:

```
from corner import corner

key_text = """Key:

$\log \mathcal{R}_0$: Reproduction number
$\log k$: Dispersion factor
$D$: Generation time interval [days]
$n$: Number of index cases
$\Delta t$: Time since index case [days]
$\\alpha$: Gamma function shape parameter"""

std_bin_size = 25
bins = [std_bin_size, std_bin_size - 15, std_bin_size, std_bin_size - 5,
        std_bin_size, std_bin_size]

samples[:, 0] = np.log10(samples[:, 0])
samples[:, 1] = np.log10(samples[:, 1])

hist_kwargs = dict(plot_contours=False, plot_datapoints=False,
                   no_fill_contours=False, bins=bins)

corner(samples, labels=['$\log \mathcal{R}_0$', '$\log k$', '$D$', '$n$',
                        '$\Delta t$', '$\\alpha$'],
       smooth=True, contour=False, **hist_kwargs)

plt.annotate(key_text, xy=(0.55, 0.8), fontsize=18,
             ha='left', va='bottom', xycoords='figure fraction')

plt.savefig('plots/corner.pdf', bbox_inches='tight')
plt.show()
```



We investigate the larger uncertainties and long tail towards large $\mathcal{R}_0$ with the "corner plot" above. The diagonal elements in the matrix of plots (histograms) represent the posterior distributions for each parameter (see label for each column in the bottom row). The off-diagonal elements represent joint posterior distributions for each pair of model parameters, and darker pixels represent a higher density of posterior samples. Note for example that the 2D histogram in the second row, first column is the same as the figure above (with its axes swapped). The corner plot is useful for examining degeneracies between parameters, which are visible as correlations between model parameters.

There are degeneracies between four pairs of model parameters. First, simulated epidemics can reproduce the observed cumulative incidence on 18 Jan 2020 equally well with small $\mathcal{R}_0$ and small D, or with larger $\mathcal{R}_0$ and larger D. There is degeneracy between $\mathcal{R}_0$ and the Γ-function shape parameter α the observed cumulative incidence is reproduced equally well with $\log_{10} \mathcal{R}_0 = 0.2$ and $\alpha = 5$, or with $\log_{10} \mathcal{R}_0 = 1$ and $\alpha = 1$. There are also degeneracies between $\mathcal{R}_0$ and n, and α and D.

Part IV

eugene

This is the documentation for eugene.

CHAPTER

FOUR

REFERENCE/API

4.1 eugene Package

4.1.1 Functions

abc(n_processes, R0_grid, ...)	
compute(R0_grid, k_grid, trials, D_min, ...)	
simulate_outbreak(R0, k, n, D, gamma_shape, ...)	Simulate an outbreak.
test(**kwargs)	Run the tests for the package.

abc

eugene.**abc**(*n_processes*, *R0_grid*, *n_grid_points_per_process*, ***parameters*)

compute

eugene.**compute**(*R0_grid*, *k_grid*, *trials*, *D_min*, *D_max*, *n_min*, *n_max*, *max_cases*, *gamma_shape_min*, *gamma_shape_max*, *max_time*, *days_elapsed_min*, *days_elapsed_max*, *min_number_cases*, *max_number_cases*, *samples_path*)

simulate_outbreak

eugene.**simulate_outbreak**(*R0*, *k*, *n*, *D*, *gamma_shape*, *max_time*, *days_elapsed_max*, *max_cases*, *seed=None*)

Simulate an outbreak.

Parameters

R0
[float]

k
[float]

n
[float]

D
[float]

gamma_shape
[float]

max_time
[float]

days_elapsed_max
[float]

max_cases
[float]

seed
[int]

Returns

times
[ndarray] Times of incidence measurements

cumulative_incidence
[ndarray] Cumulative incidence (total cases) at each time

test

eugene.**test**(***kwargs*)
Run the tests for the package.

This method builds arguments for and then calls `pytest.main`.

Parameters

package
[str, optional] The name of a specific package to test, e.g. 'io.fits' or 'utils'. Accepts comma separated string to specify multiple packages. If nothing is specified all default tests are run.

args
[str, optional] Additional arguments to be passed to `pytest.main` in the `args` keyword argument.

docs_path
[str, optional] The path to the documentation .rst files.

open_files
[bool, optional] Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Requires the `psutil` package.

parallel
[int or 'auto', optional] When provided, run the tests in parallel on the specified number of CPUs. If parallel is `'auto'`, it will use the all the cores on the machine. Requires the `pytest-xdist` plugin.

pastebin
[('failed', 'all', None), optional] Convenience option for turning on py.test pastebin output. Set to 'failed' to upload info for failed tests, or 'all' to upload info for all tests.

pdb
[bool, optional] Turn on PDB post-mortem analysis for failing tests. Same as specifying `--pdb` in `args`.

pep8
[bool, optional] Turn on PEP8 checking via the pytest-pep8 plugin and disable normal tests. Same as specifying `--pep8 -k pep8` in `args`.

plugins
[list, optional] Plugins to be passed to `pytest.main` in the `plugins` keyword argument.

remote_data
[{'none', 'astropy', 'any'}, optional] Controls whether to run tests marked with @pytest.mark.remote_data. This can be set to run no tests with remote data (`none`), only ones that use data from http://data.astropy.org (`astropy`), or all tests that use remote data (`any`). The default is `none`.

repeat
[`int`, optional] If set, specifies how many times each test should be run. This is useful for diagnosing sporadic failures.

skip_docs
[`bool`, optional] When `True`, skips running the doctests in the .rst files.

test_path
[str, optional] Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.

verbose
[bool, optional] Convenience option to turn on verbose output from py.test. Passing True is the same as specifying `-v` in `args`.

PYTHON MODULE INDEX

INDEX