Bayesian inference of COVID-19

Release 0.1.7

Jonas Dehning, Johannes Zierenberg, F. Paul Spitzner, Michael W

May 26, 2020

CONTENTS

CHAPTER ONE

INSTALLATION

There exists three different possiblities to run the models:

1. Clone the repository, with the latest release:

```
git clone --branch v0.1.7 https://github.com/Priesemann-Group/covid19_inference
```

2. Install the module via pip

```
pip install git+https://github.com/Priesemann-Group/covid19_inference.git@v0.1.7
```

3. Run the notebooks directly in Google Colab. At the top of the notebooks files there should be a symbol which opens them directly in a Google Colab instance.

CHAPTER

TWO

FIRST STEPS

To get started, we recommend to look at one of the currently two example notebooks:

1. **SIR model with one german state** This model is similar to the one discussed in our paper: Inferring COVID-19 spreading rates and potential change points for case number forecasts. The difference is that the delay between infection and report is now lognormal distributed and not fixed.
2. **Hierarchical model of the German states** This builds a hierarchical bayesian model of the states of Germany

We can for example recommend the following articles about bayesian modeling:

As a introduction to Bayesian statistics and the python package (PyMC3) that we use: https://docs.pymc.io/notebooks/api_quickstart.html

This is a good post about hierarchical Bayesian models in general: https://statmodeling.stat.columbia.edu/2014/01/21/everything-need-know-bayesian-statistics-learned-eight-schools/

CHAPTER

THREE

DISCLAIMER

We evaluate the data provided by the John Hopkins University link. We exclude any liability with regard to the quality and accuracy of the data used, and also with regard to the correctness of the statistical analysis. The evaluation of the different growth phases represents solely our personal opinion.

The number of cases reported may be significantly lower than the number of people actually infected. Also, we must point out that week-ends and changes in the test system may lead to fluctuations in reported cases that have no equivalent in actual case numbers.

Certainly, at this stage all statistical predictions are subject to great uncertainty because the general trends of the epidemic are not yet clear. In any case, the statistical trends that we interpret from the data are only suitable for predictions if the measures taken by the government and authorities to contain the pandemic remain in force and are being followed by the population. We must also point out that, even if the statistics indicate that the epidemic is under control, we may at any time see a resurgence of infection figures until the disease is eradicated worldwide.

CHAPTER

FOUR

MODEL

class covid19_inference.model.**Cov19Model**(*new_cases_obs*, *data_begin*, *fcast_len*, *diff_data_sim*, *N_population*, *name=''*, *model=None*)

Model class used to create a covid-19 propagation dynamics model. Parameters below are passed to the constructor. Attributes (Variables) are available after creation and can be accessed from every instance. Some background:

- The simulation starts *diff_data_sim* days before the data.
- The data has a certain length, on which the inference is based. This length is given by *new_cases_obs*.
- After the inference, a forecast takes of length *fcast_len* takes place, starting on the day after the last data point in *new_cases_obs*.
- In total, traces produced by a model run have the length *sim_len = diff_data_sim + data_len + fcast_len*
- Date ranges include both boundaries. For example, if *data_begin* is March 1 and *data_end* is March 3 then *data_len* will be 3.

Parameters

- **new_cases_obs** (*1 or 2d array*) – If the array is two-dimensional, an hierarchical model will be constructed. First dimension is then time, the second the region/country.
- **data_begin** (*datatime.datetime*) – Date of the first data point
- **fcast_len** (*int*) – Number of days the simulations runs longer than the data
- **diff_data_sim** (*int*) – Number of days the simulation starts earlier than the data. Should be significantly longer than the delay between infection and report of cases.
- **N_population** (*number or 1d array*) – Number of inhabitance in region, needed for the S(E)IR model. Is ideally 1 dimensional if new_cases_obs is 2 dimensional
- **name** (*string*) – suffix appended to the name of random variables saved in the trace
- **model** – specify a model, if this one should expand another

Variables

- **new_cases_obs** (*1 or 2d array*) – as passed during construction
- **data_begin** (*datatime.datetime*) – date of the first data point in the data
- **data_end** (*datatime.datetime*) – date of the last data point in the data
- **sim_begin** (*datatime.datetime*) – date at which the simulation begins
- **sim_end** (*datatime.datetime*) – date at which the simulation ends (should match fcast_end)

- **fcast_begin** (*datatime.datetime*) – date at which the forecast starts (should be one day after data_end)
- **fcast_end** (*datatime.datetime*) – data at which the forecast ends
- **data_len** (*int*) – total number of days in the data
- **sim_len** (*int*) – total number of days in the simulation
- **fcast_len** (*int*) – total number of days in the forecast
- **diff_data_sim** (*int*) – difference in days between the simulation begin and the data begin. The simulation starting time is usually earlier than the data begin.

Example

```
with Cov19Model(**params) as model:
    # Define model here
```

covid19_inference.model.**modelcontext**(*model*)

return the given model or try to find it in the context if there was none supplied.

covid19_inference.model.**student_t_likelihood**(*new_cases_inferred*, *pr_beta_sigma_obs=30*, *nu=4*, *offset_sigma=1*, *model=None*, *data_obs=None*, *name_student_t='_new_cases_studentT'*, *name_sigma_obs='sigma_obs'*)

Set the likelihood to apply to the model observations (*model.new_cases_obs*) We assume a `StudentT` distribution because it is robust against outliers [Lange1989]. The likelihood follows:

$$P(\text{data\_obs}) \sim StudentT(\text{mu} = \text{new\_cases\_inferred}, sigma = \sigma, \text{nu} = \text{nu})$$

$$\sigma = \sigma_r \sqrt{\text{new\_cases\_inferred} + \text{offset\_sigma}}$$

The parameter σ_r follows a `HalfCauchy` prior distribution with parameter beta set by `pr_beta_sigma_obs`. If the input is 2 dimensional, the parameter σ_r is different for every region.

Parameters

- **new_cases_inferred** (`TensorVariable`) – One or two dimensonal array. If 2 dimensional, the first dimension is time and the second are the regions/countries
- **pr_beta_sigma_obs** (*float*) – The beta of the `HalfCauchy` prior distribution of σ_r.
- **nu** (*float*) – How flat the tail of the distribution is. Larger nu should make the model more robust to outliers. Defaults to 4 [Lange1989].
- **offset_sigma** (*float*) – An offset added to the sigma, to make the inference procedure robust. Otherwise numbers of `new_cases_inferred` would lead to very small errors and diverging likelihoods. Defaults to 1.
- **model** – The model on which we want to add the distribution
- **data_obs** (*array*) – The data that is observed. By default it is `model.new_cases_ob`
- **name_student_t** – The name under which the studentT distribution is saved in the trace.
- **name_sigma_obs** – The name under which the distribution of the observable error is saved in the trace

Returns *None*

References

`covid19_inference.model.`**`SIR`**(*lambda_t_log*, *mu*, *pr_I_begin=100*, *model=None*, *return_all=False*, *save_all=False*)

Implements the susceptible-infected-recovered model.

$$I_{new}(t) = \lambda_t I(t-1)\frac{S(t-1)}{N}$$
$$S(t) = S(t-1) - I_{new}(t)$$
$$I(t) = I(t-1) + I_{new}(t) - \mu I(t)$$

The prior distribution of the recovery rate μ is set to $LogNormal(\text{log(pr\_median\_mu)), pr\_sigma\_mu})$. And the prior distribution of $I(0)$ to $HalfCauchy(\text{pr\_beta\_I\_begin})$

Parameters

- **`lambda_t_log`** (`TensorVariable`) – time series of the logarithm of the spreading rate, 1 or 2-dimensional. If 2-dimensional the first dimension is time.
- **`mu`** (`TensorVariable`) – the recovery rate μ, typically a random variable. Can be 0 or 1-dimensional. If 1-dimensional, the dimension are the different regions.
- **`pr_I_begin`** (float or array_like or `TensorVariable`) – Prior beta of the Half-Cauchy distribution of $I(0)$.
- **`pr_median_mu`** (*`float or array_like`*) – Prior for the median of the lognormal distrubution of the recovery rate μ.
- **`pr_sigma_mu`** (*`float or array_like`*) – Prior for the sigma of the lognormal distribution of recovery rate μ.
- **`model`** (*`Cov19Model`*) – if none, it is retrieved from the context
- **`return_all`** (*`bool`*) – if True, returns `new_I_t`, `I_t`, `S_t` otherwise returns only `new_I_t`
- **`save_all`** (*`bool`*) – if True, saves `new_I_t`, `I_t`, `S_t` in the trace, otherwise it saves only `new_I_t`

Returns

- **new_I_t** (`TensorVariable`) – time series of the number daily newly infected persons.
- **I_t** (`TensorVariable`) – time series of the infected (if return_all set to True)
- **S_t** (`TensorVariable`) – time series of the susceptible (if return_all set to True)

`covid19_inference.model.`**`SEIR`**(*lambda_t_log*, *pr_beta_I_begin=100*, *pr_beta_new_E_begin=50*, *pr_median_mu=0.125*, *pr_mean_median_incubation=4*, *pr_sigma_median_incubation=1*, *sigma_incubation=0.4*, *pr_sigma_mu=0.2*, *model=None*, *return_all=False*, *save_all=False*, *name_median_incubation='median_incubation'*)

Implements a model similar to the susceptible-exposed-infected-recovered model. Instead of a exponential decaying incubation period, the length of the period is lognormal distributed. The complete equation is:

$$
\begin{aligned}
E_{\text{new}}(t) &= \lambda_t I(t-1)\frac{S(t)}{N} \\
S(t) &= S(t-1) - E_{\text{new}}(t) \\
I_{\text{new}}(t) &= \sum_{k=1}^{10} \beta(k) E_{\text{new}}(t-k) \\
I(t) &= I(t-1) + I_{\text{new}}(t) - \mu I(t) \\
\beta(k) &= P(k) \sim LogNormal(\log(d_{\text{incubation}})), \text{sigma\_incubation})
\end{aligned}
$$

The recovery rate μ and the incubation period is the same for all regions and follow respectively:

$$
\begin{aligned}
P(\mu) &\sim LogNormal(\text{log(pr\_median\_mu)), pr\_sigma\_mu}) \\
P(d_{\text{incubation}}) &\sim Normal(\text{pr\_mean\_median\_incubation, pr\_sigma\_median\_incubation})
\end{aligned}
$$

The initial number of infected and newly exposed differ for each region and follow prior `HalfCauchy` distributions:

$$
\begin{aligned}
E(t) &\sim HalfCauchy(\text{pr\_beta\_E\_begin}) \quad \text{for } t \in \{-9, -8, ..., 0\} \\
I(0) &\sim HalfCauchy(\text{pr\_beta\_I\_begin}).
\end{aligned}
$$

Parameters

- **`lambda_t_log`** (`TensorVariable`) – time series of the logarithm of the spreading rate, 1 or 2-dimensional. If 2-dimensional, the first dimension is time.
- **`pr_beta_I_begin`** (*`float or array_like`*) – Prior beta of the `HalfCauchy` distribution of $I(0)$.
- **`pr_beta_new_E_begin`** (*`float or array_like`*) – Prior beta of the `HalfCauchy` distribution of $E(0)$.
- **`pr_median_mu`** (*`float or array_like`*) – Prior for the median of the `Lognormal` distribution of the recovery rate μ.
- **`pr_mean_median_incubation`** – Prior mean of the `Normal` distribution of the median incubation delay $d_{\text{incubation}}$. Defaults to 4 days [Nishiura2020], which is the median serial interval (the important measure here is not exactly the incubation period, but the delay until a person becomes infectious which seems to be about 1 day earlier as showing symptoms).
- **`pr_sigma_median_incubation`** – Prior sigma of the `Normal` distribution of the median incubation delay $d_{\text{incubation}}$. Default is 1 day.
- **`sigma_incubation`** – Scale parameter of the `Lognormal` distribution of the incubation time/ delay until infectiousness. The default is set to 0.4, which is about the scale found in [Nishiura2020], [Lauer2020].
- **`pr_sigma_mu`** (*`float or array_like`*) – Prior for the sigma of the lognormal distribution of recovery rate μ.
- **`model`** (*`Cov19Model`*) – if none, it is retrieved from the context
- **`return_all`** (*`bool`*) – if True, returns `new_I_t`, `new_E_t`, `I_t`, `S_t` otherwise returns only `new_I_t`
- **`save_all`** (*`bool`*) – if True, saves `new_I_t`, `new_E_t`, `I_t`, `S_t` in the trace, otherwise it saves only `new_I_t`

- **name_median_incubation** (*str*) – The name under which the median incubation time is saved in the trace

Returns

- **new_I_t** (TensorVariable) – time series of the number daily newly infected persons.
- **new_E_t** (TensorVariable) – time series of the number daily newly exposed persons. (if return_all set to True)
- **I_t** (TensorVariable) – time series of the infected (if return_all set to True)
- **S_t** (TensorVariable) – time series of the susceptible (if return_all set to True)

References

`covid19_inference.model.`**`delay_cases`**(*new_I_t*, *pr_median_delay=10*, *pr_sigma_median_delay=0.2*, *pr_median_scale_delay=0.3*, *pr_sigma_scale_delay=None*, *model=None*, *save_in_trace=True*, *name_delay='delay'*, *name_delayed_cases='new_cases_raw'*, *len_input_arr=None*, *len_output_arr=None*, *diff_input_output=None*)

Convolves the input by a lognormal distribution, in order to model a delay:

$$y_{\text{delayed}}(t) = \sum_{\tau=0}^{T} y_{\text{input}}(\tau) LogNormal[log(\text{delay}), \text{pr\_median\_scale\_delay}](t-\tau)$$
$$log(\text{delay}) = Normal(log(\text{pr\_sigma\_delay}), \text{pr\_sigma\_delay})$$

For clarification: the $LogNormal$ distribution is a function evaluated at $t-\tau$.

If the model is 2-dimensional, the $log(\text{delay})$ is hierarchically modelled with the *hierarchical_normal()* function using the default parameters except that the prior σ of delay_{L2} is HalfNormal distributed (`error_cauchy=False`).

Parameters

- **new_I_t** (TensorVariable) – The input, typically the number newly infected cases $I_{new}(t)$ of from the output of *SIR()* or *SEIR()*.
- **pr_median_delay** (*float*) – The mean of the `normal` distribution which models the prior median of the `LogNormal` delay kernel.
- **pr_sigma_median_delay** (*float*) – The standart devaiation of `normal` distribution which models the prior median of the `LogNormal` delay kernel.
- **pr_median_scale_delay** (*float*) – The scale (width) of the `LogNormal` delay kernel.
- **pr_sigma_scale_delay** (*float*) – If it is not None, the scale is of the delay is kernel follows a prior `LogNormal` distribution, with median `pr_median_scale_delay` and scale `pr_sigma_scale_delay`.
- **model** (*Cov19Model*) – if none, it is retrieved from the context
- **save_in_trace** (*bool*) – whether to save y_{delayed} in the trace
- **name_delay** (*str*) – The name under which the delay is saved in the trace, suffixes and prefixes are added depending on which variable is saved.

- **name_delayed_cases** (*str*) – The name under which the delay is saved in the trace, suffixes and prefixes are added depending on which variable is saved.
- **len_input_arr** – Length of `new_I_t`. By default equal to `model.sim_len`. Necessary because the shape of theano tensors are not defined at when the graph is built.
- **len_output_arr** (*int*) – Length of the array returned. By default it set to the length of the cases_obs saved in the model plus the number of days of the forecast.
- **diff_input_output** (*int*) – Number of days the returned array begins later then the input. Should be significantly larger than the median delay. By default it is set to the `model.diff_data_sim`.

Returns **new_cases_inferred** (`TensorVariable`) – The delayed input $y_{\text{delayed}}(t)$, typically the daily number new cases that one expects to measure.

`covid19_inference.model.`**`week_modulation`**(*new_cases_raw*, *week_modulation_type='abs_sine'*, *pr_mean_weekend_factor=0.3*, *pr_sigma_weekend_factor=0.5*, *week_end_days=(6, 7)*, *model=None*, *save_in_trace=True*)

Adds a weekly modulation of the number of new cases:

$$\text{new\_cases} = \text{new\_cases\_raw} \cdot (1 - f(t)) \,, \qquad \text{with}$$
$$f(t) = f_w \cdot \left(1 - \left|\sin\left(\frac{\pi}{7}t - \frac{1}{2}\Phi_w\right)\right|\right),$$

if `week_modulation_type` is `"abs_sine"` (the default). If `week_modulation_type` is `"step"`, the new cases are simply multiplied by the weekend factor on the days set by `week_end_days`

The weekend factor f_w follows a Lognormal distribution with median `pr_mean_weekend_factor` and sigma `pr_sigma_weekend_factor`. It is hierarchically constructed if the input is two-dimensional by the function *`hierarchical_normal()`* with default arguments.

The offset from Sunday Φ_w follows a flat `VonMises` distribution and is the same for all regions.

Parameters

- **new_cases_raw** (`TensorVariable`) – The input array, can be one- or two-dimensional
- **week_modulation_type** (*str*) – The type of modulation, accepts `"step"` or `"abs_sine` (the default).
- **pr_mean_weekend_factor** (*float*) – Sets the prior mean of the factor f_w by which weekends are counted.
- **pr_sigma_weekend_factor** (*float*) – Sets the prior sigma of the factor f_w by which weekends are counted.
- **week_end_days** (*tuple of ints*) – The days counted as weekend if `week_modulation_type` is `"step"`
- **model** (*Cov19Model*) – if none, it is retrieved from the context
- **save_in_trace** (*bool*) – If True (default) the new_cases are saved in the trace.

Returns **new_cases** (`TensorVariable`)

`covid19_inference.model.`**`make_change_point_RVs`**(*change_points_list*, *pr_median_lambda_0*, *pr_sigma_lambda_0=1*, *model=None*)

Parameters

- **priors_dict** –
- **change_points_list** –
- **model** –

covid19_inference.model.**lambda_t_with_sigmoids**(*change_points_list*, *pr_median_lambda_0*, *pr_sigma_lambda_0=0.5*, *model=None*)

Parameters

- **change_points_list** –
- **pr_median_lambda_0** –
- **pr_sigma_lambda_0** –
- **model** (*Cov19Model*) – if none, it is retrieved from the context

covid19_inference.model.**hierarchical_normal**(*name*, *name_sigma*, *pr_mean*, *pr_sigma*, *len_L2*, *w=1.0*, *error_fact=2.0*, *error_cauchy=True*)

Implements an hierarchical normal model:

$$
\begin{aligned}
x_\text{L1} &= Normal(\text{pr\_mean}, \text{pr\_sigma})\\
y_{i,\text{L2}} &= Normal(x_\text{L1}, \sigma_\text{L2})\\
\sigma_\text{L2} &= HalfCauchy(\text{error\_fact} \cdot \text{pr\_sigma})
\end{aligned}
$$

It is however implemented in a non-centered way, that the second line is changed to:

$$y_{i,\text{L2}} = x_\text{L1} + Normal(0,1) \cdot \sigma_\text{L2}$$

See for example https://arxiv.org/pdf/1312.0906.pdf

Parameters

- **name** (*str*) – Name under which x_L1 and y_L2 saved in the trace. `'_L1'` and `'_L2'` is appended
- **name_sigma** (*str*) – Name under which σ_L2 saved in the trace. `'_L2'` is appended.
- **pr_mean** (*float*) – Prior mean of x_L1
- **pr_sigma** (*float*) – Prior sigma for x_L1 and (muliplied by `error_fact`) for σ_L2
- **len_L2** (*int*) – length of y_L2
- **error_fact** (*float*) – Factor by which `pr_sigma` is multiplied as prior for *sigma_text{L2}*
- **error_cauchy** (*bool*) – if False, a $HalfNormal$ distribution is used for σ_L2 instead of $HalfCauchy$

Returns

- **y** (`TensorVariable`) – the random variable y_L2
- **x** (`TensorVariable`) – the random variable x_L1

`covid19_inference.model.`**`make_prior_I`**(*lambda_t_log*, *mu*, *pr_median_delay*, *pr_sigma_I_begin=2*, *n_data_points_used=5*, *model=None*)

Builds the prior for I begin by solving the SIR differential from the first data backwards. This decorrelates the I_begin from the lambda_t at the beginning, allowing a more efficient sampling. The example_one_bundesland runs about 30% faster with this prior, instead of a HalfCauchy.

Parameters

- **`lambda_t_log`** (`TensorVariable`) –
- **`mu`** (`TensorVariable`) –
- **`pr_median_delay`** (*`float`*) –
- **`pr_sigma_I_begin`** (*`float`*) –
- **`n_data_points_used`** (*`int`*) –
- **`model`** (*`Cov19Model`*) – if none, it is retrieved from the context

Returns **I_begin** (`TensorVariable`)

CHAPTER

FIVE

DATA RETRIEVAL

Table of Contents

5.1 Utility

`covid19_inference.data_retrieval.retrieval.`**`set_data_dir`**(*fname=None*, *permissions=None*)

Set the global variable _data_dir. New downloaded data is placed there. If no argument provided we try the default tmp directory. If permissions are not provided, uses defaults if fname is in user folder. If not in user folder, tries to set 777.

`covid19_inference.data_retrieval.retrieval.`**`backup_instances`**(*trace=None*, *model=None*, *fname='latest_'*)

helper to save or load trace and model instances. loads from *fname* if provided traces and model variables are None, else saves them there.

5.2 Johns Hops University

class covid19_inference.data_retrieval.**JHU**(*auto_download=False*)

This class can be used to retrieve and filter the dataset from the online repository of the coronavirus visual dashboard operated by the Johns Hopkins University.

Features

- download all files from the online repository of the coronavirus visual dashboard operated by the Johns Hopkins University.
- filter by deaths, confirmed cases and recovered cases
- filter by country and state
- filter by date

Example

```
jhu = cov19.data_retrieval.JHU()
jhu.download_all_available_data()

#Acess the data by
jhu.data
#or
jhu.get_new("confirmed","Italy")
jhu.get_total(filter)
```

__init__(*auto_download=False*)

On init of this class the base Retrieval Class __init__ is called, with jhu specific arguments.

Parameters **auto_download** (*bool, optional*) – Whether or not to automatically call the download_all_available_data() method. One should explicitly call this method for more configuration options (default: false)

download_all_available_data(*force_local=False, force_download=False*)

Attempts to download from the main urls (self.url_csv) which was set on initialization of this class. If this fails it downloads from the fallbacks. It can also be specified to use the local files or to force the download. The download methods get inhereted from the base retrieval class.

Parameters

- **force_local** (*bool, optional*) – If True forces to load the local files.
- **force_download** (*bool, optional*) – If True forces the download of new files

get_total_confirmed_deaths_recovered(*country: str = None, state: str = None, begin_date: datetime.datetime = None, end_date: datetime.datetime = None*)

Retrieves all confirmed, deaths and recovered cases from the Johns Hopkins University dataset as a DataFrame with datetime index. Can be filtered by country and state, if only a country is given all available states get summed up.

Parameters

- **country** (*str, optional*) – name of the country (the “Country/Region” column), can be None if the whole summed up data is wanted (why would you do this?)
- **state** (*str, optional*) – name of the state (the “Province/State” column), can be None if country is set or the whole summed up data is wanted

- **begin_date** (*datetime.datetime, optional*) – intial date for the returned data, if no value is given the first date in the dataset is used
- **end_date** (*datetime.datetime, optional*) – last date for the returned data, if no value is given the most recent date in the dataset is used

Returns *pandas.DataFrame*

get_new(*value='confirmed'*, *country: str = None*, *state: str = None*, *data_begin: datetime.datetime = None*, *data_end: datetime.datetime = None*)

Retrieves all new cases from the Johns Hopkins University dataset as a DataFrame with datetime index. Can be filtered by value, country and state, if only a country is given all available states get summed up.

Parameters

- **value** (*str*) – Which data to return, possible values are - "confirmed", - "recovered", - "deaths" (default: "confirmed")
- **country** (*str, optional*) – name of the country (the "Country/Region" column), can be None
- **state** (*str, optional*) – name of the state (the "Province/State" column), can be None
- **begin_date** (*datetime.datetime, optional*) – intial date for the returned data, if no value is given the first date in the dataset is used
- **end_date** (*datetime.datetime, optional*) – last date for the returned data, if no value is given the most recent date in the dataset is used

Returns *pandas.DataFrame* – table with new cases and the date as index

get_total(*value='confirmed'*, *country: str = None*, *state: str = None*, *data_begin: datetime.datetime = None*, *data_end: datetime.datetime = None*)

Retrieves all total/cumulative cases from the Johns Hopkins University dataset as a DataFrame with datetime index. Can be filtered by value, country and state, if only a country is given all available states get summed up.

Parameters

- **value** (*str*) – Which data to return, possible values are - "confirmed", - "recovered", - "deaths" (default: "confirmed")
- **country** (*str, optional*) – name of the country (the "Country/Region" column), can be None
- **state** (*str, optional*) – name of the state (the "Province/State" column), can be None
- **begin_date** (*datetime.datetime, optional*) – intial date for the returned data, if no value is given the first date in the dataset is used
- **end_date** (*datetime.datetime, optional*) – last date for the returned data, if no value is given the most recent date in the dataset is used

Returns *pandas.DataFrame* – table with total/cumulative cases and the date as index

filter_date(*df*, *begin_date: datetime.datetime = None*, *end_date: datetime.datetime = None*)

Returns give dataframe between begin and end date. Dataframe has to have a datetime index.

Parameters

- **begin_date** (*datetime.datetime, optional*) – First day that should be filtered

- **end_date** (*datetime.datetime, optional*) – Last day that should be filtered

Returns *pandas.DataFrame*

get_possible_countries_states()
Can be used to get a list with all possible states and couties.

Returns *pandas.DataFrame in the format*

5.3 Robert Koch Institute

class covid19_inference.data_retrieval.**RKI**(*auto_download=False*)
This class can be used to retrieve and filter the dataset from the Robert Koch Institute Robert Koch Institute. The data gets retrieved from the arcgis dashboard.

Features

- download the full dataset
- filter by date
- filter by bundesland
- filter by recovered, deaths and confirmed cases

Example

```
rki = cov19.data_retrieval.RKI()
rki.download_all_available_data()

#Acess the data by
rki.data
#or
rki.get_new("confirmed","Sachsen")
rki.get_total(filter)
```

__init__(*auto_download=False*)
On init of this class the base Retrieval Class __init__ is called, with rki specific arguments.

Parameters **auto_download** (*bool, optional*) – Whether or not to automatically call the download_all_available_data() method. One should explicitly call this method for more configuration options (default: false)

download_all_available_data(*force_local=False, force_download=False*)
Attempts to download from the main url (self.url_csv) which was given on initialization. If this fails download from the fallbacks. It can also be specified to use the local files or to force the download. The download methods get inhereted from the base retrieval class.

Parameters

- **force_local** (*bool, optional*) – If True forces to load the local files.
- **force_download** (*bool, optional*) – If True forces the download of new files

get_total(*value='confirmed', bundesland: str = None, landkreis: str = None, data_begin: datetime.datetime = None, data_end: datetime.datetime = None, date_type: str = 'date'*)
Gets all total confirmed cases for a region as dataframe with date index. Can be filtered with multiple arguments.

Parameters

- **value** (*str*) – Which data to return, possible values are - "confirmed", - "recovered", - "deaths" (default: "confirmed")
- **bundesland** (*str, optional*) – if no value is provided it will use the full summed up dataset for Germany
- **landkreis** (*str, optional*) – if no value is provided it will use the full summed up dataset for the region (bundesland)
- **data_begin** (*datetime.datetime, optional*) – initial date, if no value is provided it will use the first possible date
- **data_end** (*datetime.datetime, optional*) – last date, if no value is provided it will use the most recent possible date
- **date_type** (*str, optional*) – type of date to use: reported date 'date' (Meldedatum in the original dataset), or symptom date 'date_ref' (Refdatum in the original dataset)

Returns *pandas.DataFrame*

get_new(*value='confirmed', bundesland: str = None, landkreis: str = None, data_begin: datetime.datetime = None, data_end: datetime.datetime = None, date_type: str = 'date'*)

Retrieves all new cases from the Robert Koch Institute dataset as a DataFrame with datetime index. Can be filtered by value, bundesland and landkreis, if only a country is given all available states get summed up.

Parameters

- **value** (*str*) – Which data to return, possible values are - "confirmed", - "recovered", - "deaths" (default: "confirmed")
- **bundesland** (*str, optional*) – if no value is provided it will use the full summed up dataset for Germany
- **landkreis** (*str, optional*) – if no value is provided it will use the full summed up dataset for the region (bundesland)
- **data_begin** (*datetime.datetime, optional*) – intial date for the returned data, if no value is given the first date in the dataset is used, if none is given could yield errors
- **data_end** (*datetime.datetime, optional*) – last date for the returned data, if no value is given the most recent date in the dataset is used

Returns *pandas.DataFrame* – table with daily new confirmed and the date as index

filter(*data_begin: datetime.datetime = None, data_end: datetime.datetime = None, variable='confirmed', date_type='date', level=None, value=None*)

Filters the obtained dataset for a given time period and returns an array ONLY containing only the desired variable.

Parameters

- **data_begin** (*datetime.datetime, optional*) – initial date, if no value is provided it will use the first possible date
- **data_end** (*datetime.datetime, optional*) – last date, if no value is provided it will use the most recent possible date
- **variable** (*str, optional*) – type of variable to return possible types are: "confirmed" : cases (default) "AnzahlTodesfall" : deaths "AnzahlGenesen" : recovered
- **date_type** (*str, optional*) – type of date to use: reported date 'date' (Meldedatum in the original dataset), or symptom date 'date_ref' (Refdatum in the original dataset)

- **level** (*str, optional*) –

  possible strings are: "None" : return data from all Germany (default) "Bundesland" : a state "Landkreis" : a region

- **value** (*None, optional*) – string of the state/region e.g. "Sachsen"

Returns *pd.DataFrame* – array with ONLY the requested variable, in the requested range. (one dimensional)

filter_all_bundesland(*begin_date: datetime.datetime = None, end_date: datetime.datetime = None, variable='confirmed', date_type='date'*)

Filters the full RKI dataset

Parameters

- **df** (*DataFrame*) – RKI dataframe, from get_rki()
- **begin_date** (*datetime.datetime*) – initial date to return
- **end_date** (*datetime.datetime*) – last date to return
- **variable** (*str, optional*) – type of variable to return: cases ("AnzahlFall"), deaths ("AnzahlTodesfall"), recovered ("AnzahlGenesen")
- **date_type** (*str, optional*) – type of date to use: reported date 'date' (Meldedatum in the original dataset), or symptom date 'date_ref' (Refdatum in the original dataset)

Returns *pd.DataFrame* – DataFrame with datetime dates as index, and all German regions (bundesländer) as columns

5.4 Robert Koch Institute situation reports

class covid19_inference.data_retrieval.**RKIsituationreports**(*auto_download=False*)

As mentioned by Matthias Linden, the daily situation reports have more available data. This class retrieves this additional data from Matthias website and parses it into the format we use i.e. a datetime index.

Interesting new data is for example ICU cases, deaths and recorded symptoms. For now one can look at the data by running

Example

```
rki_si_re = cov19.data_retrieval.RKIsituationreports(True)
print(rki_si_re.data)
```

Todo: Filter functions for ICU, Symptoms and maybe even daily new cases for the respective categories.

__init__(*auto_download=False*)

On init of this class the base Retrieval Class __init__ is called, with rki situation reports specific arguments.

Parameters **auto_download** (*bool, optional*) – Whether or not to automatically call the download_all_available_data() method. One should explicitly call this method for more configuration options (default: false)

download_all_available_data(*force_local=False, force_download=False*)

Attempts to download from the main url (self.url_csv) which was given on initialization. If this fails download from the fallbacks. It can also be specified to use the local files or to force the download. The download methods get inhereted from the base retrieval class.

Parameters

- **force_local** (*bool, optional*) – If True forces to load the local files.
- **force_download** (*bool, optional*) – If True forces the download of new files

5.5 Google

class covid19_inference.data_retrieval.**GOOGLE**(*auto_download=False*)

This class can be used to retrieve the mobility dataset from Google.

Example

```
gl = cov19.data_retrieval.GOOGLE()
gl.download_all_available_data()

#Acess the data by
gl.data
#or
gl.get_changes(filter)
```

__init__(*auto_download=False*)

On init of this class the base Retrieval Class __init__ is called, with google specific arguments.

Parameters **auto_download** (*bool, optional*) – Whether or not to automatically call the download_all_available_data() method. One should explicitly call this method for more configuration options (default: false)

download_all_available_data(*force_local=False, force_download=False*)

Attempts to download from the main url (self.url_csv) which was given on initialization. If this fails download from the fallbacks. It can also be specified to use the local files or to force the download. The download methods get inhereted from the base retrieval class.

Parameters

- **force_local** (*bool, optional*) – If True forces to load the local files.
- **force_download** (*bool, optional*) – If True forces the download of new files

get_changes(*country: str, state: str = None, region: str = None, data_begin: datetime.datetime = None, data_end: datetime.datetime = None*)

Returns a dataframe with the relative changes in mobility to a baseline, provided by google. They are separated into "retail and recreation", "grocery and pharmacy", "parks", "transit", "workplaces" and "residental". Filterable for country, state and region and date.

Parameters

- **country** (*str*) – Selected country for the mobility data.
- **state** (*str, optional*) – State for the selected data if no value is selected the whole country is chosen
- **region** (*str, optional*) – Region for the selected data if no value is selected the whole region/country is chosen
- **data_end** (*data_begin,*) – Filter for the desired time period

Returns *pandas.DataFrame*

get_possible_counties_states_regions()
Can be used to obtain all different possible countries with there corresponding possible states and regions.

Returns *pandas.DataFrame*

5.6 Our World in Data

class covid19_inference.data_retrieval.**OWD**(*auto_download=False*)
This class can be used to retrieve the testings dataset from Our World in Data.

Example

```
owd = cov19.data_retrieval.OWD()
owd.download_all_available_data()
```

__init__(*auto_download=False*)
On init of this class the base Retrieval Class __init__ is called, with google specific arguments.

Parameters **auto_download** (*bool, optional*) – Whether or not to automatically call the download_all_available_data() method. One should explicitly call this method for more configuration options (default: false)

download_all_available_data(*force_local=False*, *force_download=False*)
Attempts to download from the main url (self.url_csv) which was given on initialization. If this fails download from the fallbacks. It can also be specified to use the local files or to force the download. The download methods get inhereted from the base retrieval class.

Parameters

- **force_local** (*bool, optional*) – If True forces to load the local files.
- **force_download** (*bool, optional*) – If True forces the download of new files

get_possible_countries()
Can be used to obtain all different possible countries in the dataset.

Returns *pandas.DataFrame*

get_total(*value='tests'*, *country=None*, *data_begin=None*, *data_end=None*)
Retrieves all new cases from the Our World in Data dataset as a DataFrame with datetime index. Can be filtered by value, country and state, if only a country is given all available states get summed up.

Parameters

- **value** (*str*) – Which data to return, possible values are - “confirmed”, - “tests”, - “deaths” (default: “confirmed”)
- **country** (*str*) – name of the country
- **begin_date** (*datetime.datetime, optional*) – intial date for the returned data, if no value is given the first date in the dataset is used
- **end_date** (*datetime.datetime, optional*) – last date for the returned data, if no value is given the most recent date in the dataset is used

Returns *pandas.DataFrame* – table with new cases and the date as index

get_new(*value='tests'*, *country=None*, *data_begin=None*, *data_end=None*)

Retrieves all new cases from the Our World in Data dataset as a DataFrame with datetime index. casesn be filtered by value, country and state, if only a country is given all available states get summed up.

Parameters

- **value** (*str*) – Which data to return, possible values are - "confirmed", - "tests", - "deaths" (default: "confirmed")
- **country** (*str*) – name of the country
- **begin_date** (*datetime.datetime, optional*) – intial date for the returned data, if no value is given the first date in the dataset is used
- **end_date** (*datetime.datetime, optional*) – last date for the returned data, if no value is given the most recent date in the dataset is used

Returns *pandas.DataFrame* – table with new cases and the date as index

5.7 Base Retrieval Class

class covid19_inference.data_retrieval.retrieval.**Retrieval**(*name*, *url_csv*, *fallbacks*, *update_interval=None*, ***kwargs*)

Each source class should inherit this base retrieval class, it streamlines alot of base functions. It manages downloads, multiple fallbacks and local backups via timestamp. At init of the parent class the Retrieval init should be called with the following arguments, these get saved as attributes.

An example for the usage can be seen in the _Google, _RKI and _JHU source files.

__init__(*name*, *url_csv*, *fallbacks*, *update_interval=None*, ***kwargs*)

Parameters

- **name** (*str*) – A name for the Parent class, mainly used for the local file backup.
- **url_csv** (*str*) – The url to the main dataset as csv, if an empty string if supplied the fallback routines get used.
- **fallbacks** (*array*) – Fallbacks can be filepaths to local or online sources or even methods defined in the parent class.
- **update_interval** (*datetime.timedelta*) – If the local file is older than the update_interval it gets updated once the download all function is called.

_download_csv_from_source(*filepath*, ***kwargs*)

Uses pandas read csv to download the csv file. The possible kwargs can be seen in the pandas documentation.

These kwargs can vary for the different parent classes and should be defined there!

filepath [str] Full path to the desired csv file

Returns *bool* – True if the retrieval was a success, False if it failed

_fallback_handler()

Recursivly iterate over all fallbacks and try to execute subroutines depending on the type of fallback.

_timestamp_local_old(*force_local=False*) → bool

1. Get timestamp if it exists
2. compare with the date today
3. update if data is older than set intervall -> can be parent dependant

`_save_to_local``()`

Creates a local backup for the self.data pandas.DataFrame. And a timestamp for the source.

CHAPTER

SIX

PLOTTING

covid19_inference.plotting.**get_all_free_RVs_names**(*model*)

Returns the names of all free parameters of the model

Parameters **model** (*pm.Model instance*) –

Returns *list* – all variable names

covid19_inference.plotting.**get_prior_distribution**(*model*, *x*, *varname*)

Given a model and variable name, get the prior that was used for modeling.

Parameters

- **model** (*pm.Model instance*) –
- **x** (*list or array*) –
- **varname** (*string*) –

Returns *array* – the prior distribution evaluated at x

covid19_inference.plotting.**plot_hist**(*model*, *trace*, *ax*, *varname*, *colors=('tab:blue', 'tab:orange')*, *bins=50*)

Plots one histogram of the prior and posterior distribution of the variable varname.

Parameters

- **model** (*pm.Model instance*) –
- **trace** (*trace of the model*) –
- **ax** (*matplotlib.axes instance*) –
- **varname** (*string*) –
- **colors** (*list with 2 colornames*) –
- **bins** (*number or array*) – passed to np.hist

Returns *None*

covid19_inference.plotting.**plot_cases**(*trace*, *new_cases_obs*, *date_begin_sim*, *diff_data_sim*, *start_date_plot=None*, *end_date_plot=None*, *ylim=None*, *week_interval=None*, *colors=('tab:blue', 'tab:orange')*, *country='Germany'*)

Plots the new cases, the fit, forecast and lambda_t evolution

Parameters

- **trace** (*trace returned by model*) –
- **new_cases_obs** (*array*) –

- **date_begin_sim** (*datetime.datetime*) –
- **diff_data_sim** (*float*) – Difference in days between the begin of the simulation and the data
- **start_date_plot** (*datetime.datetime*) –
- **end_date_plot** (*datetime.datetime*) –
- **ylim** (*float*) – the maximal y value to be plotted
- **week_interval** (*int*) – the interval in weeks of the y ticks
- **colors** (*list with 2 colornames*) –

Returns *figure, axes*

VARIABLES SAVED IN THE TRACE

The trace by default contains the following parameters in the SIR/SEIR hierarchical model. XXX denotes a number.

Name in trace	Dimensions	Created by function
lambda_XXX_L1	samples	lambda_t_with_sigmoids/make_change_point_RVs
lambda_XXX_L2	samples x regions	lambda_t_with_sigmoids/make_change_point_RVs
sigma_lambda_XXX_L2	samples	lambda_t_with_sigmoids/make_change_point_RVs
transient_day_XXX_L1	samples	lambda_t_with_sigmoids/make_change_point_RVs
transient_day_XXX_L2	samples x regions	lambda_t_with_sigmoids/make_change_point_RVs
sigma_transient_day_XXX_L2	samples	lambda_t_with_sigmoids/make_change_point_RVs
transient_len_XXX_L1	samples	lambda_t_with_sigmoids/make_change_point_RVs
transient_len_XXX_L2	samples x regions	lambda_t_with_sigmoids/make_change_point_RVs
sigma_transient_len_XXX_L2	samples	lambda_t_with_sigmoids/make_change_point_RVs
delay_L1	samples	delay_cases
delay_L2	samples x regions	delay_cases
sigma_delay_L2	samples	delay_cases
weekend_factor_L1	samples	week_modulation
weekend_factor_L2	samples x regions	week_modulation
sigma_weekend_factor_L2	samples	week_modulation
offset_modulation	samples	week_modulation
new_cases_raw	samples x time x regions	week_modulation
mu	samples	SIR/SEIR
I_begin	samples x regions	SIR/SEIR
new_cases	samples x time x regions	SIR/SEIR
sigma_obs	samples x regions	SIR/SEIR
new_E_begin	samples x 11 x regions	SEIR
median_incubation_L1	samples	SEIR
median_incubation_L2	samples x regions	SEIR
sigma_median_incubation_L2	samples	SEIR

For the non-hierchical model, variables with _L2 suffixes are missing, and _L1 suffixes are removed from the name.

CHAPTER

EIGHT

CONTRIBUTING

We always welcome contributions. Here we gather some guidelines to make the process as smooth as possible.

8.1 Beginning

To see where help is needed, go to the issues page on Github. If you want to begin on an issue, make a comment below and begin a draft pull request: https://github.blog/2019-02-14-introducing-draft-pull-requests/ You can link the pull request on the right side of the commit to it.

When you have finished working on the issue, change it to a regular pull request. Check that there are no conflicts to the current master (https://www.digitalocean.com/community/tutorials/how-to-rebase-and-update-a-pull-request)

8.2 Code formatting

We use black https://github.com/psf/black as automatic code formatter. Please run your code through it before you open a pull request.

We do not check for formatting in the testing (travis) but recommend to set up black as a pre-commit hook.

```
conda install -c conda-forge pre-commit
pre-commit install
```

Try to stick to PEP 8. You can use type annotations if you want, but it is not necessary or encouraged.

8.3 Testing

We use travis and pytest. To check your changes locally:

```
python -m pytest --log-level=INFO --log-cli-level=INFO
```

It would be great if anything that is added to the code-base has an according test in the `tests` folder. We are not there yet, but it is on the todo. Be encouraged to add tests :)

8.4 Documentation

The documentation is built using Sphinx from the docstrings. To test it before submitting, navigate with a terminal to the docs/ directory. Install if necessary the packages listed in `piprequirements.txt` run `make html`. The documentation can then be accessed in `docs/_build/html/index.html`. As an example you can look at the documentation of *`covid19_inference.model.SIR()`*

CHAPTER

NINE

DEBUGGING

This is some pointer to help debugging models and sampling issues

9.1 General approach for nans/infs during sampling

The idea of this approach is to sample from the prior and then run the model. If the log likelihood is then -inf, there is a problem, and the output of the theano functions is inspected.

Sample from prior:

```
from pymc3.util import (
    get_untransformed_name,
    is_transformed_name)

varnames = list(map(str, model.vars))

for name in varnames:
    if is_transformed_name(name):
        varnames.append(get_untransformed_name(name))

with model:
    points = pm.sample_prior_predictive(var_names = varnames)
    points_list = []
    for i in range(len(next(iter(points.values())))):
        point_dict = {}
        for name, val in points.items():
                point_dict[name] = val[i]
        points_list.append(point_dict)
```

points_list is a list of the starting points for the model, sampled from the prior. Then to run the model and print the log-likelihood:

```
fn = model.fn(model.logpt)

for point in points_list[:]:
    print(fn(point))
```

To monitor the output and save it in a file (for use in ipython). Learned from: http://deeplearning.net/software/theano/tutorial/debug_faq.html#how-do-i-step-through-a-compiled-function

```
%%capture cap --no-stderr
def inspect_inputs(i, node, fn):
    print(i, node, "input(s) value(s):", [input[0] for input in fn.inputs],
```

(continues on next page)

(continued from previous page)

```
          end='')

def inspect_outputs(i, node, fn):
    print(" output(s) value(s):", [output[0] for output in fn.outputs])

fn_monitor = model.fn(model.logpt,
                      mode=theano.compile.MonitorMode(
                           pre_func=inspect_inputs,
                           post_func=inspect_outputs).excluding(
                                 'local_elemwise_fusion', 'inplace'))

fn = model.fn(model.logpt)

for point in points_list[:]:
    if fn(point) < -1e10:
        print(fn_monitor(point))
        break
```

In a new cell:

```
with open('output.txt', 'w') as f:
    f.write(cap.stdout)
```

Then one can open output.txt in a text editor, and follow from where infs or nans come from by following the inputs and outputs up through the graph

9.2 Sampler: MCMC (Nuts)

9.2.1 Divergences

During sampling, a significant fraction of divergences are a sign that the sampler doesn't sample the whole posterior. In this case the model should be reparametrized. See this tutorial for a typical example: https://docs.pymc.io/notebooks/Diagnosing_biased_Inference_with_Divergences.html

And these papers include some more details: https://pdfs.semanticscholar.org/7b85/fb48a077c679c325433fbe13b87560e12886.pdf https://arxiv.org/pdf/1312.0906.pdf

9.2.2 Bad initial energy

This typically occurs when some distribution in the model can't be evaluated at the starting point of chain. Run this to see which distribution throws nans or infs:

```
for RV in model.basic_RVs:
    print(RV.name, RV.logp(model.test_point))
```

However, this is only evaluates the test_point. When PyMC3 starts sampling, it adds some jitter around this test_point, which then could lead to nans. Run this to add jitter and then evaluate the logp:

```
chains=4
for RV in model.basic_RVs:
    print(RV.name)

```

(continues on next page)

(continued from previous page)

```
    for _ in range(chains):
        mean = {var: val.copy() for var, val in model.test_point.items()}
        for val in mean.values():
            val[...] += 2 * np.random.rand(*val.shape) - 1
        print(RV.logp(mean))
```

This code could potentially change in newer versions of PyMC3 (this is tested in 3.8). Read the source code, to know which random jitter PyMC3 currently adds at beginning.

9.2.3 Nans occur during sampling

Run the sampler with the debug mode of Theano.

```
from theano.compile.nanguardmode import NanGuardMode
mode = NanGuardMode(nan_is_error=True, inf_is_error=False, big_is_error=False,
                    optimizer='o1')
trace = pm.sample(mode=mode)
```

However this doesn't lead to helpful messages if nans occur during gradient evaluations.

9.3 Sampler: Variational Inference

There exist some ways to track parameters during sampling. An example:

```
with model:
    advi = pm.ADVI()
    print(advi.approx.group)

    print(advi.approx.mean.eval())
    print(advi.approx.std.eval())

    tracker = pm.callbacks.Tracker(
        mean=advi.approx.mean.eval,  # callable that returns mean
        std=advi.approx.std.eval  # callable that returns std
    )

    approx = advi.fit(100000, callbacks=[tracker],
                      obj_optimizer=pm.adagrad_window(learning_rate=1e-3),)
                     #total_grad_norm_constraint=10) #constrains maximal gradient,␣
↪could help


    print(approx.groups[0].bij.rmap(approx.params[0].eval()))

    plt.plot(tracker['mean'])
    plt.plot(tracker['std'])
```

For the tracker, the order of the parameters is saved in:

```
approx.ordering.by_name
```

and the indices encoded there in the slc field. To plot the mean value of a given parameter name, run:

```
plt.plot(np.array(tracker['mean'])[:, approx.ordering.by_name['name'].slc]
```

The debug mode is set with the following parameter:

```
from theano.compile.nanguardmode import NanGuardMode
mode = NanGuardMode(nan_is_error=True, inf_is_error=False, big_is_error=False,
                    optimizer='o1')
approx = advi.fit(100000, callbacks=[tracker],
             fn_kwargs={'mode':mode})
```

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Lange1989] Lange, K., Roderick J. A. Little, & Jeremy M. G. Taylor. (1989). Robust Statistical Modeling Using the t Distribution. Journal of the American Statistical Association, 84(408), 881-896. doi:10.2307/2290063

[Nishiura2020] Nishiura, H.; Linton, N. M.; Akhmetzhanov, A. R. Serial Interval of Novel Coronavirus (COVID-19) Infections. Int. J. Infect. Dis. 2020, 93, 284–286. https://doi.org/10.1016/j.ijid.2020.02.060.

[Lauer2020] Lauer, S. A.; Grantz, K. H.; Bi, Q.; Jones, F. K.; Zheng, Q.; Meredith, H. R.; Azman, A. S.; Reich, N. G.; Lessler, J. The Incubation Period of Coronavirus Disease 2019 (COVID-19) From Publicly Reported Confirmed Cases: Estimation and Application. Ann Intern Med 2020. https://doi.org/10.7326/M20-0504.

PYTHON MODULE INDEX

INDEX